Database querying on the World Wide Web: the design and implementation of domain-specific database search engines

Carlos Fernando Enguix

University of Wollongong

Follow this and additional works at: https://ro.uow.edu.au/theses

Recommended Citation
Database Querying on the World Wide Web: The Design and Implementation of Domain-specific Database Search Engines

A thesis submitted in fulfilment of the requirements for the award of the degree

Honours Master in Commerce

University of Wollongong

by

Carlos Fernando Enguix,
Diplomate in Informatics UPV Spain,
Grad. Dip. in Business UoW

Faculty of Commerce - Business Systems Department
1999
Certificate of Originality

Decision Systems Laboratory

Business Systems Department
University of Wollongong, March 1999

I hereby declare that this submission is completely original as it does not contain material previously published or written by other individuals and that the research proposed in this thesis may possibly contribute significantly to the research domain being studied with some innovative ideas and concepts.

Carlos Fernando Enguix
Conferences and Workshops Presented:

Parts of this thesis have been presented in the following Conferences, Workshops or Decision Systems Lab technical reports:

7th World Wide Web Conference: Doctoral Consortium

Published in: Computer Networks and ISDN systems 30 (1998) 567-572
Available at: http://www7.conf.au/programme/docpapers/1891/com1891.htm

7th World Wide Web Conference: Workshop on Reuse of Web information

Published by the CSIRO as:
Proceedings of the Workshop on Reuse of Web Information held in conjunction with the 7th WWW Conference, 14 April 1998, Brisbane, Australia, Anne-Marie Vercoustre, Maria Milosavljevic and Ross Wilkinson. Editors, CSIRO Report Number CMIS 98-11, 107 pages.
Available at: http://www.mel.dit.csiro.au/~vercous/REUSE/accepted-papers.htm

Decision Systems Laboratory, Technical Report TR98/5/101:
Available at: http://budhi/uow.edu/postgrad/carlos/tr98_5_101.htm
Abstract

The World Wide Web can be considered to be a very large semi-structured database that holds vast amounts of useful information. However, our ability to query, search, and reuse the information on the Web is limited at present. Existing search techniques suffer from critical deficiencies with respect to robustness, flexibility, and precision. This research attempts to develop domain-centred alternatives to keyword and subject directory search engines. By focusing on well-demarcated logical domains whose ontologies can be modelled using third-generation, object-relational data models, we demonstrate how relevant Web data can be restructured using object-relational databases, against which SQL-type queries can be issued. A prototype implementation for the ‘universities’ domain entitled ‘UniGuide’ is presented in this dissertation.
Capitan
Viento que me arrastrabas,
Contra corriente siempre voy
Huyendo de mi mismo,
Huyendo de mi propia dirección
Por mucho que alce las velas
No sigo el rumbo que quiero yo
Por poco que descuido mi timón
Barco a la deriva soy...
Capitán de mi vida intento ser
Mas viento traicionero por la
Espalda me hiciste ver
Cuan duro es navegar por el mundo,
Cuan duro es...

Luz de Luces
Luces albinas durmiendo en su soporte
Centros de gravedad yacen inmóviles
Luces amarillas señalan hacia el norte
Proyectan en mí la dimensión de la incertidumbre
Crepúsculos misteriosos paren entre los árboles
Ensaye más fuertes que la mugre
Que crece por este mundanal camino
Luces de la ciudad emulan un paraíso divino
Mas en verdad es el desenlace de un espejismo
Engañabobos de una ficción y no el albedrío
Luces de colores quisiera yo en mi entorno
Vida misma, encima de la naturaleza, río
De esperanzas y de retornos
La luz que engendraste materia
Dícese lugar al Universo arterias
La luz de vida brillar en cada ser vivo
La mística energía sin fin ni principio
Luz inmortal brillar, brilla...brilla omnipotente
Sobre el profano, el agnóstico y el alma creyente

Light of Lights
Albinos lights sleeping on their support
Centres of Gravity lie motionless
Yellow lights pointing to the North
Project in me the dimension of uncertainty
Mysterious twilights bearing between the trees
Stronger fantasies than the dirt
That grows around this worldly path
The city lights emulate a divine paradise
The truth is that it is the outcome of a mirage
Trickster of a fiction and not free will
Coloured lights I would like around me
Life itself, enzyme of the nature, river
Of hopes and of returns
Light that engendered matter
You gave the universe arteries
Light of life shines in each living being
The mystic energy without end or beginning
Shine immortal light, shine...shine omnipotent over the profane, the agnostic and the faithful soul

Maria Cervera
I would like to give thanks to:

First of all to my mother, Maria Enguix aka Maria Cervera, who has been giving me moral support during the whole Master. There is a special relation between us, not only we have a relation of mother-son but we are also friends, and even sometimes we discuss like little kids. As a matter of fact, we are both very young in spirit. If I have to describe what is love, then I shall say that true love implies to be with the loved ones in the good and the bad times, to forgive and be forgiven, to support, help understand and believe each other, and to never let down the loved ones. All this and more the two “Marias of my life”, my mother and my sister have demonstrated me along these years the concept of true love.

To my advisor, the Associate Professor Dr. Joseph G. Davis, who believed in me since the very first moment and offered me to apply for the Honours Master in Commerce.

To Professor Graham Winley, head of the department of Business Systems, who advised by Dr. Davis gave me the opportunity to receive a scholarship from the department.

To David Dodds, the Computer Systems Officer, who has been giving me technical support all the time. Maintaining ILLUSTRATM is not an easy task and more if you have not been trained to do so.

To Dr. Aditya Ghose who encouraged me to send articles to WWW conferences, and also gave me references to relevant articles.

To Joshua Fan, who suggested me some techniques for handling pointers in C.
To the Decision Systems Laboratory who has given me the opportunity to meet people from different cultures: Alex (Philippines), Soon Nam (Korea), Boon (Malaysia), Justin (Denmark), Dadong (China), Elaine (Taiwan), Mario (Australia), etc.

To Cindy, a pretty & petite young JAVA-NESE™ lady who has given to my heart very good moments … and encouraged me to have faith… in Jesus and God

To Erwin Kalvelagen, from the Netherlands, who developed the JDBC driver for ILLUSTR A and facilitated generously to us. These are the kind of people that disinterestedly help others, and fortunately we can find lots of them in the Information Universe, the Web.

To Informix for the help provided in some technical questions

To Jesus and God
For helping me in difficult moments…
For giving me strength…
For protecting the stubborn child inside of me, a child that needs to know, that needs to learn, that needs to discover, that is sustained and addicted to knowledge…

The more we know the more we acknowledge how ignorant we are…only God knows!
Overview of Contents

Chapter 1. Background and Introduction - 1

1.1 WWW Search Engines: current situation - 2
1.2 The design and implementation of an alternative: database querying on the WWW - 5
1.3 Research Objectives - 7
1.4 Organisation of the dissertation - 7

Chapter 2. Review of the Literature – 9

2.1 Introduction - 10
2.2 Early approaches to searching the Internet - 17
2.3 General classification of Search Engines - 20
2.4 Keyword-based Web Search - 21
2.5 Database Approaches to searching the WWW - 36
2.6 Other related work - 45
2.7 Summary - 53
Chapter 3. Research Methodology – 55

3.1 Introduction - 56
3.2 Proof of concept Research Methodology - 58
3.3 Design Framework - 62
3.4 Implementation environment - 63
3.5 Evaluation Methodology - 64
3.6 Summary - 65

Chapter 4. Design and Implementation of a Domain-specific Object-Relational Database Search Engine – 66

4.1 Introduction - 67
4.2 Modelling specific domains on the WWW - 69
4.3 Object-Relational Model of UniGuide - 75
4.4 Transformation of the model into an object-relational database - 79
4.5 Architecture - 83
4.6 End-user perspective: User Interface Sections - 183
4.7 Summary - 195

Chapter 5. Generalised Methodological Guidelines – 196

5.1 Introduction - 197
5.2 Methodology - 198
5.3 Summary - 200
Chapter 6. Contributions, Implications, Limitations and Future Research – 201

6.1 Introduction - 202
6.2 Research contributions - 202
6.3 Implications - 203
6.4 Limitations - 206
6.5 Future Research - 207
6.6 Summary - 210

7. References - 211
Table of Contents

Certificate of Originality - ii
Conferences and Workshops presented - iii
Abstract - iv
Poems - v
Acknowledgements - vi
Overview of Contents - viii
Table of Contents - xi
List of Tables - xvii
List of Figures - xviii
Acronyms and Abbreviations - xx

Chapter 1. Background and Introduction – 1

1.1 WWW Search Engines: current situation - 2
   1.1.1 The Query Problem: Deficiencies in current implementations - 3
   1.1.2 Problems to be solved - 4

1.2 The design and implementation of an alternative: database querying on the WWW - 5
1.3 Research Objectives - 7
1.4 Organisation of the dissertation – 7
Chapter 2. Review of the Literature – 9

2.1 Introduction - 10
  2.1.1 History of the Internet and the World Wide Web - 11
  2.1.2 Current situation of the WWW – 16

2.2 Early approaches to searching the Internet - 17
  2.2.1 WHOIS - 17
  2.2.2 X.500 - 18
  2.2.3 Archie - 19
  2.2.4 WAIS - 19
  2.2.5 Veronica – 20

2.3 General classification of Search Engines - 20

2.4 Keyword-based Web Search - 21
  2.4.1 Client-based/real-time - 21
    2.4.1.1 The Fish-Search - 21
    2.4.1.2 WebSQL - 23
  2.4.2 Distributed indexing - 25
    2.4.2.1 Harvest: a Scalable Resource Discovery System - 26
  2.4.3 Robot-based centralised indexing - 28
  2.4.4 Meta-search engines - 34
  2.4.5 Subject Directory - 35

2.5 Database Approaches to searching the WWW - 36
  2.5.1 A Multi-layered Approach: MLDB - 36
  2.5.2 The Araneus Data Model and View Language - 40
    2.5.2.1 The Araneus Data Model - 40
    2.5.2.2 The Araneus database view language: ULIXES - 41
    2.5.2.3 The Araneus hypertextual view language: PENEOPE - 42
  2.5.3 Virtual Database Technology - 43

2.6 Other related work - 45
  2.6.1 Internet Metadata formats - 47
    2.6.1.1 IAFA Templates - 49
    2.6.1.2 Dublin Core - 50
    2.6.1.3 Resource Description Framework – 52

2.7 Summary – 53

Chapter 3. Research Methodology – 55
Chapter 4. Design and Implementation of a Domain-specific Object-Relational Database Search Engine – 66

4.1 Introduction – 67

4.2 Modelling specific domains on the WWW - 69
  4.2.1 Modelling ontologies - 69
  4.2.2 Relational Model - 70
  4.2.3 Object Oriented Model - 71
  4.2.4 Hybrid model: object-relational model - 73
  4.2.5 A comparison of different modelling approaches - 74

4.3 Object-Relational Model of UniGuide - 75
4.4 Transformation of the model into an object-relational database - 79

4.5 Architecture - 83
  4.5.1 High-level overview of components - 83
  4.5.2 Server-side components - 87
    4.5.2.1 ILLUSTRORDBMS - 88
      4.5.2.1.1 Prototype table attributes - 90
      4.5.2.1.2 Prototype set attributes - 91
    4.5.2.2 ILLUSTRASQL3 - 92
      4.5.2.2.1 DDL statements - 93
        4.5.2.2.1.1 CREATE TYPE statement - 93
        4.5.2.2.1.2 CREATE TABLE - 94
        4.5.2.2.1.3 CREATE INDEX - 95
        4.5.2.2.1.4 CREATE RULE - 95
        4.5.2.2.1.5 CREATE FUNCTION - 96
      4.5.2.2.2 DML statements - 96
        4.5.2.2.2.1 INSERT INTO table statement - 97
        4.5.2.2.2.2 UPDATE table statement - 97
        4.5.2.2.2.3 DELETE FROM table statement - 98
      4.5.2.2.3 SELECT statements - 98
        4.5.2.2.3.1 Simple Queries - 99
        4.5.2.2.3.2 Summarised Information - 100
        4.5.2.2.3.3 Joins - 100
        4.5.2.2.3.4 Fetching sets: an alternative - 101
        4.5.2.2.3.5 Simulation of transitive closure - 102
4.5.2.4 UniGuide Rules Subsystem - 113
  4.5.2.4.1 Rules-based Check Constraints - 114
  4.5.2.4.2 Rules-based Referential Integrity - 115
  4.5.2.4.3 Timestamp update Rules - 116
  4.5.2.4.4 URL Foreign key update rules - 116
  4.5.2.4.5 Parent Object Reference update Rules - 118
  4.5.2.4.6 Insert/Update Valid set instance and Synchronised insert/update/delete Pseudo Primary keys Rules - 119

4.5.2.5 UniGuide SQL3 Generator - 122
  4.5.2.5.1 DML Generation Statement Modules - 127
  4.5.2.5.2 Query Statement Generation Modules - 133
  4.5.2.5.3 Query filtering modules - 136

4.5.2.6 ILLUSTRATA Web-Datablade Applications - 138
  4.5.2.6.1 Insert/Update Applications - 147
  4.5.2.6.2 Query Generation Applications - 149
  4.5.2.6.3 Query filtering Applications - 151

4.5.3 Client-side: Javascripts - 153
  4.5.3.1 Insert/Update modules - 155
  4.5.3.2 Query Generation modules - 157
  4.5.3.3 Query filtering modules - 159

4.5.4 Integration and interaction between server-side and client-side - 160
  4.5.4.1 Insert/Update modules - 161
  4.5.4.2 Query Generation modules - 162
  4.5.4.3 Query filtering modules - 165

4.5.5 Meta data design: UniGuide scheme meta-tags - 166
  4.5.5.1 UniGuide Meta tag Generator - 168

4.5.6 UniGuide Indexing Agent - 169
  4.5.6.1 app_Index_Robot_HD - 174
  4.5.6.2 Robot_Stats - 175
  4.5.6.3 URL_Scanner - 175
  4.5.6.4 JDBC_MetaData Saver - 177
  4.5.6.4.1 UniGuide SQL3 Generator: DML Generation Statement Modules Indexing Agent Version - 178
  4.5.6.5 JDBC_URL_Saver - 181
  4.5.6.6 ThreadLimit - 182
  4.5.6.7 Stop_URL_Scanner - 182

4.6 End-user perspective: User Interface Sections - 183
  4.6.1 Submit URL Section - 184
  4.6.2 Simple Queries Section - 187
  4.6.3 Free-SQL Section - 189
  4.6.4 Meta Tag Generator Section - 191
  4.6.5 Query Taxonomy: examples of possible queries - 192
Chapter 5. Generalised Methodological Guidelines – 196

5.1 Introduction - 197
5.2 Methodology - 198
5.3 Summary - 200

Chapter 6. Contributions, Implications, Limitations and Future Research – 201

6.1 Introduction - 202
6.2 Research contributions - 202
6.3 Implications - 203
  6.3.1 Academic/Research community - 203
  6.3.2 Developers - 204
  6.3.3 Other End-users - 205
6.4 Limitations - 206
6.5 Future Research - 207
6.6 Summary - 210

7. References - 211

Appendix A. UniGuide SQL3 Generator Library: Source Code – 216

A.1 DML Statement Generation Modules - 216
  A.1.1 DML Statement Generation Modules: Row instances - 216
  A.1.2 DML Statement Generation Modules: Set instances - 223

A.2 Query Statement Generation Modules - 240
  A.2.1 Query Statement Generation Modules: Row instances - 240
  A.2.2 Query Statement Generation Modules: Set instances - 244

A.3 Query Filtering Modules - 251
Appendix B. Web-Datablade Applications and Embedded JavaScripts - 254

B.1 Insert/Update Web-Datablade Applications - 254
   B.1.1 Insert/Update Web-Datablade Applications: State transition diagram - 254
   B.1.2 Insert/Update Web-Datablade Applications: Source Code - 255

B.2 Query Web-Datablade Applications - 267
   B.2.1 Query Web-Datablade Applications: State transition diagram - 267
   B.2.2 Query Web-Datablade Applications: Source Code - 267

B.3 Query Filtering Web-Datablade Applications - 275
   B.3.1 Query Filtering Web-Datablade Applications: State transition diagram - 275
   B.3.2 Query Filtering Web-Datablade Applications: Source Code - 276

B.4 Meta tag generator Web-Datablade Application - 285
   B.4.1 Meta tag generator Web-Datablade Application: State transition Diagram - 285
   B.4.2 Meta tag generator Web-Datablade Application: Source Code - 286

Appendix C. UniGuide Indexing Agent – 294

C.1 UniGuide Indexing Agent SQL3 Generator modules: Source Code - 294
   C.1.1 DML Statement Generation Modules: Row instances - 294
   C.1.2 DML Statement Generation Modules: Set instances - 304

C.2 UniGuide Indexing Agent: Hard Drive Version - 318
   C.2.1 JAVA™ Source Code: app_Index_Robot_HD.java - 318
List of Tables

Chapter 2. Review of the Literature
2.1 Internet Statistics - 16
2.2 NetQL operators - 39
2.3 Dublin Core Elements - 51

Chapter 4. Design and Implementation of a Domain-specific Object-Relational Database Search Engine
4.1 Column descriptions table _tables - 105
4.2 An example of a given row instance from the table _tables - 106
4.3 Column descriptions table _columns - 108
4.4 Column_type values - 110
4.5 An example of a given row of data from the _columns table - 112
4.6 Interpretation of the keywords current and new in rules - 113
4.7 Correspondence between ILLUOSTRA API data types and standard C and SQL3 data types - 123
4.8 List of most important Web Datablade module tags - 140
4.9 List of TLNC elements/symbols - 144
4.10 List of elements/symbols included in WebApp structure charts - 146
4.11 UniGuide Table Browser icons - 185
4.12 Query Taxonomy - 193
List of Figures

Chapter 2. Review of the Literature
2.1 Example of a SOIF record - 27
2.2 Example of a robots.txt file - 29
2.3 Entity-Relationship Model of the WISE resource database - 31
2.4 Main components VDB technology - 44
2.5 Example of Description and Keyword meta tags - 48
2.6 Sample IAFA template record (partial) representing a document - 50
2.7 Example of Dublin Core meta-tags - 52
2.8 Example of abbreviated RDF syntax - 53

Chapter 3. Research Methodology
3.1 Relation between Research Process and Body of Knowledge - 58
3.2 A Multi-methodological approach to IS Research - 59
3.3 Principal parts of a system development research methodology - 60

Chapter 4. Design and Implementation of a Domain-specific Object-Relational Database Search Engine
4.1 Synthesised graphical representation of the Object Relational Model of UniGuide - 76
4.2 UML Object-Relational Model of UniGuide
4.3 Models obtained in a structured modelling process - 79
4.4 High-level overview of UniGuide components - 85
4.5 Harvest of metadata (meta-tags) in UniGuide - 86
4.6 ILLUSTRATA Components involved in a client-server request - 89
4.7 Relational Model of the Auxiliary Catalogue System - 104
4.8 Identification of column types in a DML statement involving a set instance - 109
4.9 Example of a rules-based check constraint - 114
4.10 Chaining rules to control uniqueness of set instances - 121
4.11 Phases in the execution of a DML/SELECT statement in UniGuide - 126
4.12 Web Datablade Module Architecture - 139
4.13 TLNC main_ins Web application - 148
4.14 Server-side WebApp structure chart main_ins Web application - 149
4.15 TLNC main_squery Web application - 150
4.16 Server-side WebApp structure chart main_squery Web application - 151
4.17 TLNC free_sql Web application - 152
4.18 Server-side WebApp structure chart free_sql Web application - 153
4.19 JavaScript Object Hierarchy (partial view) - 155
4.20 Client-side WebApp structure chart main_ins Web application - 157
4.21 Client-side WebApp structure chart main_squery Web application - 158
4.22 Client-side WebApp structure chart free_sql Web application - 159
List of Figures (continued)

4.23 Client-Server interaction WebApp structure chart main_ins Web application - 162
4.24 Client-Server interaction WebApp structure chart main_squery Web application - 164
4.25 Client-Server interaction WebApp structure chart free_sql Web application - 166
4.26 Formal syntax of a UniGuide meta tag - 167
4.27 An example of a UniGuide meta tag - 168
4.28 Client-Server interaction WebApp structure chart main_meta Web application - 169
4.29 Double daemon architecture in the ILLUSTRATI ODBC Driver - 171
4.30 UniGuide Indexing Agent: GUI - 173
4.31 UniGuide Indexing Agent: UniGuide Scheme Meta tags detected - 173
4.32 UML Class Diagram app_index_robot_HD - 183
4.33 UniGuide Database Search Engine: Main Menu - 183
4.34 UniGuide Submit URL Section - 186
4.35 UniGuide Simple Query Section: Query Formulation - 188
4.36 UniGuide Simple Query Section: Query Results - 188
4.37 UniGuide FreeSQL Section: Query Formulation - 190
4.38 UniGuide FreeSQL Section: Query Results - 190
4.39 UniGuide Meta Tag Generator Section: Data Input - 191
4.40 UniGuide Meta Tag Generator Section: Data Output - 192

Appendix A. UniGuide SQL3 Generator Library: Source Code
A1 Structure Chart GF_simple_insert.c - 216
A2 Structure Chart GF_complex_insert.c - 223
A3 Structure Chart get_complex_insert_statement module - 224
A4 Structure Chart get_complex_update_statement module - 224
A5 Structure Chart: GF_squery_simple - 240
A6 Structure Chart: GF_squery_complex - 244
A7 Structure Chart: GF_free_sql - 251

Appendix B. Web-Datablade Applications and Embedded JavaScripts
B1 Insert/Update Web-Datablade Applications: State transition diagram - 254
B2 Query Web-Datablade Applications: State transition diagram - 267
B3 Query Filtering Web-Datablade Applications: State transition diagram - 275
B4 Meta tag generator Web-Datablade Application: State transition diagram - 285

Appendix C. UniGuide Indexing Agent
C1 Structure Chart GF_robot_simple_insert.c - 294
C2 Structure Chart GF_robot_complex_insert.c - 304
C3 Structure Chart get_complex_insert_statement - 305
C4 Structure Chart get_complex_update_statement - 305
### Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADT</td>
<td>Abstract Data Type</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>CCITT</td>
<td>Consultative Committee on International Telephone and Telegraphy</td>
</tr>
<tr>
<td>CERN</td>
<td>European Organisation for Nuclear Research</td>
</tr>
<tr>
<td>CGI</td>
<td>Common Gateway Interface</td>
</tr>
<tr>
<td>CML</td>
<td>Chemical Markup Language</td>
</tr>
<tr>
<td>DARPA</td>
<td>Defense Advanced Research Projects Agency</td>
</tr>
<tr>
<td>DDL</td>
<td>Data Definition Language</td>
</tr>
<tr>
<td>DML</td>
<td>Data Manipulation Language</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>E-Commerce</td>
<td>Electronic Commerce</td>
</tr>
<tr>
<td>E-Mail</td>
<td>Electronic Mail</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>HSR</td>
<td>The Harvest Server Registry</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IAFA</td>
<td>Internet Anonymous FTP Archives</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MIME</td>
<td>Multi-purpose Internet Mail Extension</td>
</tr>
<tr>
<td>MLDB</td>
<td>Multi-Layered Database</td>
</tr>
<tr>
<td>MsqI</td>
<td>Interactive Query Processor</td>
</tr>
<tr>
<td>NCSA</td>
<td>National Centre For Supercomputer Applications</td>
</tr>
<tr>
<td>NIC</td>
<td>Network Information Center</td>
</tr>
<tr>
<td>NNTP</td>
<td>Network News Transfer Protocol</td>
</tr>
<tr>
<td>NSAPI</td>
<td>Netscape Application Program Interface</td>
</tr>
<tr>
<td>OCLC</td>
<td>Online Computer Library Center</td>
</tr>
<tr>
<td>OID</td>
<td>Object Identifier</td>
</tr>
<tr>
<td>OO</td>
<td>Object Oriented</td>
</tr>
<tr>
<td>OODB</td>
<td>Object Oriented Database</td>
</tr>
<tr>
<td>OODM</td>
<td>Object Oriented Data Model</td>
</tr>
<tr>
<td>OOM</td>
<td>Object Oriented Model</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>OOPL</td>
<td>Object Oriented Programming Language</td>
</tr>
<tr>
<td>ORDB</td>
<td>Object Relational Database</td>
</tr>
<tr>
<td>ORDBMS</td>
<td>Object Relational Database Management System</td>
</tr>
<tr>
<td>ORDM</td>
<td>Object Relational Data Model</td>
</tr>
<tr>
<td>ORM</td>
<td>Object Relational Model</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PK</td>
<td>Primary Key</td>
</tr>
<tr>
<td>POID</td>
<td>Object Identifier parent table</td>
</tr>
<tr>
<td>PseudoPK</td>
<td>Pseudo Primary Key</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>SGML</td>
<td>Standard Generalized Markup Language</td>
</tr>
<tr>
<td>SOIF</td>
<td>Summary Object Interchange Format</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SQL92</td>
<td>Structured query Language 92 version</td>
</tr>
<tr>
<td>SQL3</td>
<td>Structured Query Language Version 3</td>
</tr>
<tr>
<td>STD</td>
<td>State transition diagram</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet Protocol</td>
</tr>
<tr>
<td>Telnet</td>
<td>Remote login</td>
</tr>
<tr>
<td>TFxIDF</td>
<td>Term Frequency x Inverse Document Frequency</td>
</tr>
<tr>
<td>TLNC</td>
<td>Top level Navigational Chart</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VRML</td>
<td>Virtual Reality Markup Language</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>Web</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>WebApp</td>
<td>Web Application</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1.
Background and Introduction

1.1 WWW Search Engines: current situation

1.2 The design and implementation of an alternative: database querying on the WWW

1.3 Research Objectives

1.4 Organisation of the dissertation
Chapter 1. Background and Introduction

This chapter is organised as follows: in the first section we describe the current situation respect to search engines. In the second section we outline deficiencies associated to current technologies. Next we list the problems to be solved in order to provide an alternative to current implementations. What follows next, is the description of the alternative proposed in this research: the deployment of a domain-specific database search engine that facilitates more intelligent query formulation. We conclude explaining the research objectives to be solved and the organisation of the thesis.

1.1 WWW Search Engines: current situation

The advent of the World Wide Web (WWW, Web) in 1990 changed drastically the configuration of the Internet, giving the possibility to everybody to become a potential end-user thanks to providing a user-friendly interface to displaying information. This desirable feature encouraged information providers to push huge amounts of information on the WWW, converting it up-to-date in the most successful "Information Universe". Since 1993 it has been growing at an exponential rate, impacting dramatically on the resource discovery problem. The searching and browsing of information on the WWW sometimes becomes so frustrating, that it is common to acquire the well-known “lost in hyperspace” syndrome. As a result, search engines appeared to overcome this problematic issue of trying to find information on the Web. Currently, keyword-based search engines are the most prominent technology on the WWW. Keyword-based search engines offer free-text based queries, where end-users
can specify vague queries. End-users are overwhelmed by the number of hits returned by particular queries. These search engines are logically based on word weighting algorithms and pattern matching. Also, they are physically based on the use of huge indexes that map Uniform Resource Locators (URLs) to words.

The bulk of information being stored on the Web may be classified as semi-structured, opposed to the highly structured data stored in traditional databases. Traditional Database Managements Systems (DBMS) deal with structured data offering great precision and more focused queries as compared with keyword-based search techniques. In order to achieve those features present in traditional DBMS we must provide a mechanism that can map, convert or restructure semistructured data from the Web into structured DBMS.

1.1.1 The Query Problem: Deficiencies in current implementations

Most of the existing search engines provide a very simple interface to querying, a simple text box for entering keywords. We list below some of the more common deficiencies of current implementations (Enguix et al. 1998a):

- **Most of the search engines are keyword-based** (Poulter 1997), constrained to very limited structured querying thus providing more syntactic and less semantic precision

- **The lack of control in querying Web data**: the boundaries of the query are unknown and the output of a query is hard to predict

- **The ability to establish associations between data elements is scarce** at best or non-existent
Many of the above problems can be traced to the absence of a conceptual model that can cover the whole semantics of the WWW. Keyword-based search engines rely on the use of huge indexes that are mapped to URLs. Their power typically resides in efficient statistical algorithms for matching keywords with the contents of Web pages (Yuwono and Lee 1996). The lack of semantics leads to the situation in which it is almost impossible to establish relationships or logical associations between concepts or entities. Depending on the internal algorithms of the different search engines we can expect different results in response to queries. Also, since the search is based on syntactic precision, we usually end up with a very large number of unrelated hits, which makes it difficult to find the required information.

1.1.2 Problems to be solved

- **Provide an alternative to keyword-based search engines:** keyword-based search engines offer free text-based queries which are appropriate for vague searches but inappropriate for more focused queries that require more precision. On the other hand traditional databases give the possibility of defining more structured and precise queries. This precision can be gained switching from keyword-based technology to highly structured databases. A highly structured database requires the discovery of semantics in data, and the definition and construction of schemas capable of representing in an organised way data semantics.

- **Discovery of semantics:** the discovery of semantics associated to the logical structure of homogeneous sets of Web pages. Some parts of the Web present high regularity in their intellectual content and even in their logical structure. In fact, it is possible to capture common concepts or semantics represented by common sets of Web pages.
• **Schema definition:** the definition of schemas associated to well-demarcated logical domains on the WWW that permit the discovery of semantics. Schemas can represent not only homogeneous set of Web pages but also the inter-relationships among them. Further, schemas are based on the ontology of the particular logical domain being modelled, which may span many Web sites. Subsequently, the captured schema must be transformed into appropriate structures capable of storing related information on the Web.

• **Data conversion:** the conversion of semi-structured data into structured data, dealing with the design of metadata that can describe efficiently and effectively the semi-structured data. *From our viewpoint, Web pages are considered as containers of both structured data (metadata) and semistructured information (document contents)*

• **Data collection:** the implementation of autonomous software modules (indexing robots/agents) that automatically retrieve, restructure and store metadata associated to Web pages on the WWW into a highly structured database.

• **Query precision:** the definition of more focused and precise queries with the aid of an advanced query language such as SQL3.

1.2 The design and implementation of an alternative: database querying on the WWW

Mapping from a semi-structured layer of data in the WWW to a structured layer will give us the chance to solve the problems identified above. A common approach to providing such mapping is obtained with text restructuring programs or wrappers that can transform semi-structured data into structured data. *The common deficiency presented in the wrapper approach is the non-existence of universal wrappers. In*
other words, wrappers are site-specific, and need to be regenerated every time a given Web page changes, to adapt to the current format of the particular Web page. Another strategy consists of attaching metadata that describes more efficiently the data represented in particular Web pages and extract this metadata to store it finally in a structured database. We consider the metadata technique as format neutral, changes in the format of the document do not affect at all. Only changes in the type of content represented in the document may affect the validity of the metadata attached to the Web page. Besides, the negative aspect of metadata is the requirement of a consensus between information providers to follow a given standard. In this thesis, we advocate the metadata strategy, feasible for environments where metadata standards can be applied, such as corporate intranets, government organisations and so on. Automatic data collectors, more commonly known as indexing robots, will be responsible for capturing, parsing, extracting and storing into a database the metadata associated with Web pages. Once the database is populated, more complex and constrained queries can be issued, which supposes the achievement of a higher level of resource discovery. The query language used in our research is a more advanced structured query language (SQL3) that permits us to manage both relational and object oriented constructs. Not only can be issued structured queries against attributes but also queries that span multiple Web servers, distinguishing between intra-site (single server) and inter-site (multiple servers) queries.

1.3 Research Objectives

Mainly, our attempt is to solve the problems identified in section 1.1.2. The range and diversity of data on the web is so extensive that ontologies may have to be constructed separately for each relatively well defined domain. This strategy of "Divide and
Conquer” involves identifying and isolating reasonably well-demarcated subsets of the Web consisting of a collection of homogeneous Web sites whose structure can be modelled.

The main goal of this research is the construction of domain-specific database search engines capable of offering traditional database queries on the WWW, beyond the scope of keyword-based search engines. Also we present a general methodology that describes step by step the deployment of this kind of search engines, involving the identification and isolation of logical domains on the WWW. The term logical domain refers to parts of the WWW that are common in content and structure or present high regularity. A good example of these well-demarcated logical domains is the University logical domain. The majority of University WWW servers store information about departments, faculties, schools, staff members, courses, subjects, publications, projects and so on. There are many other logical domains such as software companies, government departments, schools, etc.

1.4 Organisation of the dissertation

The dissertation is organised as follows:

A review of the relevant literature and the research methodology involved is presented in chapter 2 and chapter 3 respectively. A model that captures the core constructs and their interrelationships (ontology model) for the ‘Australian Universities’ domain and the architecture and implementation of a prototype entitled as ‘UniGuide’ and the usage of the prototype from an end-user perspective are presented in chapter 4. In chapter 5 we describe a series of methodological guidelines which are required in the deployment of database search engines. Chapter 6 is devoted to contributions, implications,
limitations and future research. Finally we conclude the thesis with a list of appendixes and references.
Chapter 2.
Review of the Literature

2.1 Introduction
2.2 Early approaches to searching the Internet
2.3 General classification of Search Engines
2.4 Keyword-based Web Search
2.5 Database Approaches to searching the WWW
2.6 Other related work
2.7 Summary
Chapter 2. Review of the Literature

2.1 Introduction

In this chapter we begin explaining the origin of the Internet and the World Wide Web. An introduction to the current situation with some abbreviated statistics about the estimated size of the Internet and the Web is included. Also we list and describe the most remarkable early Internet search engines, which have been designed.

A taxonomy of WWW search engines is presented, differentiating mainly between keyword-based search engines and database approaches to searching the Web. From the perspective of data gathering, a different classification is outlined, distinguishing client-based/real-time search engines, distributed indexing, robot-based centralised indexing, meta-search engines and subject directory based search engines. In the database search engine section we present some of the most influential research papers from our viewpoint, especially the multi-layered database approach and the Araneus Data Model. These approaches present alternative strategies to keyword-based search engines, modelling the WWW as a highly structured database.

Furthermore, metadata formats are cited and described as a critical auxiliary data structure/object that allows the presentation of information in a compact and efficient way and can serve as an interface to the semi-structured data stored on the WWW to the highly structured information stored in databases.
This chapter is organised as follows: in this section we present an overview of the history of the Internet and the WWW and the current situation. Next, a summary of early approaches to searching the Internet is included (section 2.2). A general classification of search engines that distinguishes from a high-level viewpoint keyword-based search engines and database search engines is presented in section 2.3. A detailed taxonomy of keyword-based search engines and some of the most important database search engines in presented in section 2.4 and 2.5 respectively. The main focus of section 2.6 is on Internet meta data formats, which are critical in the deployment of the UniGuide system. This chapter terminates with a summary of the review of the literature.

2.1.1 History of the Internet and the World Wide Web

The Internet has become a world-wide mass-media for global dissemination of information and a medium that facilitates the interaction between individuals in an unprecedented scale. The Internet today is a widespread infrastructure, the initial prototype of a Global Information Infrastructure.

It could be said that the Internet got started in 1973 when the US Advanced Research Projects Agency (DARPA) initiated a research project to interlink packet networks of different types. The objective was to develop the transparent communication of computers across multiple linked packet networks (Cerf 1998). The initial set of interlinked networks was known as the “Internet”. From the research two communications protocols were developed: Transmission Control Protocol (TCP) and the Internet Protocol (IP).

The protocols were designed in order to provide (Leiner et al. 1998):
• A seamless integration of existent networks to the Internet
• A reliable transmission of information
• A decentralised control at the operations level
• The introduction of a global addressing system
• The interfacing to a range of different operating systems

The IP protocol provided control for addressing and forwarding of individual packets, while the TCP protocol was concerned with the reliable transmission of packets.

File transfer protocol (FTP) and remote login (telnet) were one of the most important applications that worked on top of the TCP/IP protocol although the electronic mail application (e-mail) had the most significant impact. E-mail gave the possibility to people to communicate and collaborate without regard of the geographic location.

In the 1980s the widespread of local area networks (LAN), Ethernet networks, personal computers (PC) and workstations influenced the initial flourishing of the Internet. Three different network classes appeared to accommodate the range of networks:

• **Class A** Large scale networks
• **Class B** Medium scale networks
• **Class C** Local Area Networks (LAN)

Another important development was the invention of the domain name systems (DNS) which permitted a scalable distributed mechanism for resolving hierarchical host names into Internet addresses.
The integration of the TCP/IP protocols into a well-known operating system in the research community (Berkeley Unix BSD) was also a critical factor in the general widespread of the Internet.

By 1985 the Internet was a well-established technology supporting a broad community of researchers, developers, and other communities. After 1989 it began to integrate support for other protocols such as Open systems Interconnection (OSI) protocols. By the end of 1991, the Internet included more than 5,000 networks and 700,000 hosts.

The beginning of a new flourishing in the Internet occurred in 1989 when the architecture of the World Wide Web was proposed by Tim Berners-Lee, a researcher at the European Particle Physics Laboratory (CERN) in Geneva, Switzerland. The goal of the World Wide Web was to be a shared information space where people could communicate (Berners-Lee 1996).

The architecture of the WWW included (Berners-Lee et al. 1994):

- The definition of a universal address system: Uniform Resource Identifier (URI)
- The invention of a markup language: Hypertext Markup Language (HTML) and
- The creation of a network protocol: Hypertext Transfer Protocol (HTTP)

URIs are strings that represent universal addresses of objects on the Web. They are universal in the sense that they are unique in the abstract space defined by the Web, which means that the Web requires a global space of identifiers. This is similar to the concept of object identifier (OID) in any object oriented database system (ODBMS) that requires the OID to be unique for a particular database. URIs can point to any kind
of resource not only using HTTP, but also using other common network protocols such as file transfer protocol (FTP), network news transfer protocol (NNTP), etc. Therefore there are URIs for Web pages, newsgroups articles, for e-mail addresses, telnet destinations, etc.

The general syntax of a URI is:

```
Resource_Type: [Resource_Parameters]
```

We list some of the main resource types that can be addressed:

- **http** for any type of document/resource: Web pages, ascii files, postscript documents, images, etc.
- **news** for news articles
- **mailto** for mail messages
- **ftp** for files retrieved via the FTP protocol
- **telnet** for remote login
- **file** for accessing files in the local file system

The resource parameters are resource dependent and may be optional.

The specific syntax for the HTTP protocol is:

```
http://server_address/[directory]
```

Server address represents the address of the Web server being accessed via the HTTP protocol and the directory indicates how to locate a particular resource inside the Web server's space.
One of the critical issues that the WWW influenced in the explosion of the growth of
the Internet, was the facility to address an available resource without consultation; in
other words URIs provided scalability to the WWW.

The HTTP protocol is a request/response protocol that can transfer HTML documents
and an extensible set of formats such as postscript files, word documents, ascii text, etc.
When these resources are transferred, additional information is transferred in HTTP
headers so the client application that receives the information can resolve which kind of
resource is about to receive. The set of headers is an extension of the Multipurpose
Internet Mail Extensions (MIME) set. HTTP is a stateless protocol in the sense that
connections between client and server are maintained only for the duration of one
operation.

The Hypertext Markup Language (HTML) was designed to be simple and adherent to
the Standard Generalized Markup Language (SGML) standard, including Document
Type Definitions (DTDs) which can be thought of the logical schema and syntax of a
valid HTML document. HTML has undergone a continuous evolution due to public and
market pressure allowing the inclusion of features such as frames, tables, applets,
scripts, plug-ins, cascading style sheets and so on.

2.1.2 Current situation of the Internet and the WWW
The World Wide Web has become the most important protocol on the Internet,
displacing other services that were previously created with the same purpose such as
Gopher and WAIS. Netnews Discussions groups still subsist as a dynamic paradigm to
interchange messages and FTP for transmission of files, especially software. The Web has become much more sophisticated with the integration of sound, video-conferencing, virtual reality (VRML), client browsers with specialized plug-ins for different types of applications, electronic commerce (E-Commerce), more advanced markup language, and so on. The integration of all these new features has accelerated even more the current growth of the Internet and the WWW. Some reports have appeared from time to time with the intention of trying to solve the non-trivial task of measuring the Internet. It is very difficult to estimate the current size, turnover and growth of the public Internet because of its dynamic nature (Kahle 1996). The following table (table 2.1 Internet Statistics) summarizes the total number of sites, total data stored and the data change rate for the most important services or protocols available in the Internet at 1996.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Number of Sites</th>
<th>Total Data</th>
<th>Data Change Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>WWW</td>
<td>400,000</td>
<td>1,500 GB</td>
<td>600 GB/Month</td>
</tr>
<tr>
<td>Gopher</td>
<td>5,000</td>
<td>100 GB</td>
<td>Declining</td>
</tr>
<tr>
<td>FTP</td>
<td>10,000</td>
<td>5,000 GB</td>
<td>Unknown</td>
</tr>
<tr>
<td>Netnews</td>
<td>20,000 Discussions</td>
<td>240 GB</td>
<td>16 GB/Month</td>
</tr>
</tbody>
</table>

Table 2.1 Internet Statistics (Kahle 1996)

The World Wide Web is growing rapidly, the number of pages is reported to be doubling every year. The average Web page size is 30 kbytes, including graphics. It was estimated that there were 50 million of pages bringing the total size of the Web to 1,500 GB. A more recent study (Bharat and Broder 1998) estimated the total number of static Web pages in the Public WWW to be approximately 200 million as of November 1997, which is 4 times the amount shown previously. Considering the average size of a Web page proposed in (Kahle 1996), this will give us a total amount of 6,000 GB of information stored on the WWW. These numbers were obtained analyzing statistically the coverage on the WWW of the four most important WWW search engines (AltaVista, Hotbot, Excite, Infoseek). Some of the results reported in this study are
really astonishing, as it is estimated that the intersection of common Web pages covered by all the four is roughly 2.25 million pages (only 1.4%). As a matter of fact, this leads us to suspect that these calculations are not very consistent.

### 2.2 Early approaches to searching the Internet

Before the explosion of the WWW starting around 1993, the Internet was in a different state as known today. In 1992 it was estimated that the Internet provided direct connectivity to more than 1 million computers world-wide. Widely accessible resources were made available such as software, documents, images, and other types of information. As a matter of fact, the location and retrieval of Internet resources have become a necessity for the whole community of users. A number of systems were developed to satisfy these needs (Schwartz et al. 1992). The focus of the early Internet search engines was on locating and retrieving files (i.e. Archie) or locating different kinds of resources and offering specialised databases (i.e. WAIS) or locating people, addresses and getting information about Internet domains (i.e. WHOIS).

#### 2.2.1 WHOIS

The WHOIS service used by Network Information Centers (NICs) and other organisations maintained databases of registered users, network numbers and domains across the Internet. WHOIS could also be used to find the existence of a particular domain name or to obtain site contact information for an Internet domain. Each WHOIS server collected geographically distributed information into single databases containing small subsets of registered Internet users and sites associated to a given NIC. Because each WHOIS server run independently without coordination among themselves, users had to deal with the distribution and inconsistencies between servers.
2.2.2 X.500

The Consultative Committee on International Telephone and Telegraphy (CCITT) and the International Organization for Standardization (ISO) developed a directory service standard named as X.500 describing a hierarchical name space. X.500 can be considered as a global White Pages Directory where directories are organised as part of a single global directory. The X.500 directory is organised hierarchically by country, organisation, organisational unit and person. In X.500, each local directory is called a Directory System Agent (DSA) which can represent one or more organisations. The Directory Information Tree (DIT) interconnects different DSAs. Directory User Agents (DUAs) are used as an interface to one or more DSAs allowing end-users to query and retrieve information. The general use of X.500 has been as a user directory. Query parameters can include the name of a person, country, organisation, etc. returning matching records containing e-mail address, phone number, address and so on. X.500 supported sub-tree searches where users can browse organisations and then issue queries to a server for that part of a tree.

2.2.3 Archie

Archie allowed the search of files of Internet FTP servers that offered anonymous FTP access. It maintained a list of thousands of UNIX anonymous FTP archives world-wide in an indexed database by performing recursive directory listings at each site periodically. Users can query this database via telnet or a forms interface available on the WWW (ArchiePlex) from any world-wide replicated archie server, using regular
expressions. End-users can query any of these databases in order to find a given software item and finally retrieve the file from the closest place to the end-user. It is still used to search software in shareware repositories.

2.2.4 Wide-Area Information Servers: WAIS

WAIS servers allowed the search and retrieval of information on the Internet from indexed specialised subject databases called sources. World-wide distributed WAIS servers offered access to hundreds of databases containing technical reports, mailing lists, weather maps, and so on. WAIS indices contained keywords mapping words to textual documents. Also WAIS could extract keywords from other different kinds of documents, based on the knowledge of particular document types. Instead of using a centralised index, WAIS distributed its indices among the servers that provide information. The decentralised set of WAIS indices provided more scalability to the WAIS architecture. The existence of a top-level index operated by Thinking Machines registered information available on each server. As a result a query must be executed in two phases, first search the directory of servers and then select a particular server to search the required information. To obtain relevant documents, WAIS ranked matches using a word weighting algorithm supporting relevance feedback where users can find documents similar in keyword occurrences to previously located documents. Users can access WAIS servers via WAIS clients, by using telnet to connect to a public WAIS client or on the WWW with a forms-based interface via a WWW-WAIS gateway.

2.2.5 Veronica

Before appearing the WWW the most popular service to browse and locate information on the Internet was Gopher, now in disuse. Gopher provided a simple menu-driven user
interface, that linked to other menu hierarchies or to a particular file. A centralised “Mother Gopher” was the first menu in the hierarchy providing a repository of links to others gophers located world-wide. Due to its popularity the number of gopher severs increased world-wide and an associated keyword-based search engine that supported boolean operators was developed in order to search menu item descriptions world-wide. The search engine named as Veronica traversed Gopher graphs reading all directory and file names and then indexed them into a large index. Veronica could be accessed with telnet or through the WWW Veronica Gateway with a forms-based interface.

2.3 General classification of Search Engines

From the semantics and data organisation perspective the taxonomy of search engines includes at its highest levels keyword-based search engines and database search engines. Keyword-based search engines are characterised by the non-existence of a schema associated to Web pages, by the lack of semantics and by free text-based queries. On the other hand database search engines try to model the Web or parts of the Web as any traditional database system associating some kind of schema to a given set of Web pages. Also a mechanism is provided to map from the semi-structured layer, such as the WWW, to a structured layer, the resource database. Once a given schema is obtained, more focused and structured queries can be issued against the database using a particular query language.

2.4 Keyword-based Web Search

From the gathering of data point of view, we can distinguish distributed indexing, client-based/real-time, robot-based centralised indexing, meta-search engines and subject directory search engines. This classification is based partially in the
classification done by Koch in the article Internet search services (Koch 1996). This is not an exhaustive classification of current search engines but a classification of the “current real players”, most probable future survivors, and the most influential ones, which are included as a base for future implementations.

2.4.1 Client-based/real-time

The search process is initiated, monitored and computed locally in the end-user’s machine, allowing in some configurations real-time search (recursive search through the hypertext structure).

2.4.1.1 The Fish-Search

One of the first client-based search engines was the fish-search for mosaic (DeBra and Post 1994) which provided automated navigation and real-time search acting as a private indexing robot. The fish search algorithm navigates through individual WWW documents. It offers a client-based search tool that does automated navigation emulating the behaviour of a browsing user, being much faster and following an optimised strategy. The search is performed on-line covering small fractions of the WWW at once and parsing the entire contents of the retrieved documents. When a search process is initiated every document is retrieved and scanned for relevant information. Links pertaining to a given document are also scanned and traversed a-posteriori. In the fish search a document is relevant if it contains the words the search tries to find. A search process can be effective and efficient depending on a given number of factors such as:

- Find a starting document from which relevant documents can be found traversing links (robot-based search engines provide good stating points)
• The algorithm must avoid downloading irrelevant documents. It is assumed that relevant documents may point to other relevant documents as well.

• With the use of auxiliary caches the retrieving of popular documents can be accelerated.

The search algorithm allows the use of regular expressions, case insensitive search, approximate match using the agrep tool, etc. It also permits the limitation of the search process to a given server, subnetwork or to sites that share part of their domain name (i.e. All servers of a given university). Depth of search specifies the maximum number of forward links (tree levels) through irrelevant documents that can be followed before backtracking on that direction. Width of search specifies the maximum number of selected outgoing links from a retrieved document that will be scanned to find relevant documents.

The fish search can be considered as the precursor for other interesting client-based search tools such as WebSQL (Arocena et al. 1997).

2.4.1.2 WebSQL

WebSQL (Arocena et al. 1997) is a high-level declarative query language for extracting information from the WWW. WebSQL queries can be issued against popular search engines or performed in real-time over subnetworks on the WWW. The WebSQL query engine consists of a library of JAVA™ classes, which allows the embedding of queries into JAVA programs.
In WebSQL keyword-based and topology-based queries can be integrated seamlessly. Keyword-based/content search ignores the structure of a hypermedia network whereas topology-based/structure search examines the hypermedia structure for subnetworks that match a given pattern. The query language allows the combination of queries with controlled automated navigation. Queries that combine content and structure are necessary when applications on the Web require accessing the structure defined by links. Some examples include checking for dangling links, unreachable documents, etc.

WebSQL models the WWW as a relational database composed of two virtual relations, Document and Anchor. The Document relation has one row for each document in the Web and the Anchor relation contains one tuple for each link in each document on the Web.

The Document relation has the following attributes:

- **url**: URL of the document
- **title**: title tag of the document
- **text**: text associated to the document, not including HTML tags
- **length**: length of the document
- **type**: format of the document i.e. html
- **modif**: date of last modification of the document

The Anchor relation has the following attributes:

- **Base**: the base URL address needed to form the absolute URL address of the link
- **Label**: text associated to the link
- **Href**: contains a relative or absolute URL address
Queries can be issued to search engines or be issued to sub-networks of the Web as navigational queries or combine both. WebSQL queries include path regular expressions that define internal links within a document (=>$), paths of zero length (=), paths to local servers (->) or paths to external servers (=>).

An abbreviated general syntax of the query language includes (WebSQL Homepage 1997) to define WebSQL queries:

```
SELECT {Attributes}
FROM {Document | Anchor} TableVar
SUCH THAT RangeCondition
WHERE {Condition}
```

The following figure demonstrates an example of a simple query that retrieves the URLs of documents that mentions the WWW in their contents, starting from the technical reports section of a given university department.

```
SELECT d.url
FROM Document d
WHERE d.text CONTAINS "WWW" OR d.title CONTAINS "WWW"
```

The main components of the WebSQL system are the compiler, a query engine and the user interface. The WebSQL compiler and query engine are implemented as a set of JAVA™ classes forming the WebSQL class library, which can be used within JAVA™ programs. The compiler parses queries and translates them into object programs that are
executed by an interpreter. The query engine evaluates conditions that can trigger the submission of requests to search engines or the depth-first traversal of subWebs. The user interface is implemented as an HTML form connected to a CGI script, as an applet using client-server architecture or as a standalone application. WebSQL was conceived as a tool not for end-users but for helping programmers to develop Web-based applications.

2.4.2 Distributed indexing

Indexes are distributed/replicated through different sites and collected by index servers called “brokers” (Bowman et al. 1995), combined to form a centralised database (ALIWEB now in disuse (Koster 94a)), or forming special topologies by creating links between resources that are topically related (Ingrid (Francis et al. 1995)). We will only describe the most important one, Harvest.

2.4.2.1 Harvest: a Scalable Resource Discovery System

Harvest is an integrated system that includes a set of customisable tools for gathering information from diverse repositories, building topic-specific content indexes widely replicated, searching indexes and caching of objects. The system interacts with WWW clients, HTTP, FTP, Gopher and newsgroups resources. The main components of the Harvest system architecture are gatherers and brokers. Gatherers collect and extract indexing information from various providers. Gatherers can collect and extract information from HTML documents, postscript files, new articles and so on. Brokers provide the indexing and query interface to the information captured by Gatherer modules. Brokers can retrieve information incrementally from various Gatherers or other Brokers. A Broker can collect information from many Gatherers so it can build
indexes of widely distributed data and a Gatherer can cater information to many Brokers. By constructing topic-specific brokers, Brokers can avoid many of the vocabulary and scaling problems of non-focused global indexes. The Harvest Server Registry (HSR) is a special broker used to register information about each Gatherer, Broker, Cache and Replicator available in the Internet. The HSR constitutes the starting point to search for a particular Broker.

In order to permit an efficient communication between Gatherers and Brokers an attribute-value stream protocol named as the Summary Object Interchange Format (SOIF) is defined. SOIF is based on Internet Anonymous FTP Archives (IAFA) templates and BibTeX. It has been designed to be sufficiently expressive to handle different kinds of objects and to handle streams of object summaries. Each SOIF record contains a type, a URL and a list of byte-count delimited attribute value pairs.

The following figure (fig. 2.1) shows an example of a SOIF record:

![Example of SOIF record](image)

Figure 2.1 Example of a SOIF record

The main search engine used in the Harvest system is Glimpse. Glimpse supports space-efficient indexes and flexible interactive queries allowing boolean expressions, regular expressions and approximate matching (agrep tool). The index used by Glimpse is
similar in principle to inverted indexes. Glimpse supports incremental indexing of information and allows the definition of more structured and complex queries with the aid of attributes gathered from objects.

The Harvest project ended officially on August 1996, but the original team is still maintaining and developing some parts (i.e. a very interesting light version of Harvest: Webglimpse with the ability to find topological neighbours of a given Web page (Manber et al. 1997).

The problem associated to the distributed approach is the required infrastructure, the installation and maintenance of a given number of modules or subsystems distributed world-wide, and more important a general consensus between information providers to adopt such technology in a world-wide scale. This can be the more reasonable answer to justify that, although it can be considered as the most promising strategy for resource discovery, why they have lost ground to robot-based centralised indexing search engines which do not required such infrastructure.

2.4.3 Robot-based centralised indexing
The principal objectives of centralized indexing robots is to include as many servers as possible, to index the whole contents of documents to provide high quality indexes and to create broad indexes normally with breadth-first search strategies (Pinkerton 1994).

The general steps performed by an indexing robot consist of retrieving documents, index the content of the currently retrieved document and parse links to other documents. In the architecture of the WebCrawler indexing robot the search engine is
responsible for deciding the next documents to be retrieved and scanned. A database stores document metadata, links between documents, and full-text indexes. Autonomous modules or agents are responsible for retrieving documents from the network directed by the search engine.

Search engines discover new documents by starting from a list of known Web pages, scanning the links that belong to the document, retrieving the documents pointed by the links and repeating the whole process on and on. Search engines consider the WWW as a large directed graph that must be traversed to retrieve all the stored information. Search engines determine not only the Web pages that they visit but also the types of documents to be scanned and retrieved. As mentioned above indexing robots try to build indexes that cover many servers on the WWW. A breadth-first algorithm ensures that every server can have at least some Web pages represented in the index. In the breadth-first strategy every immediate descendant from all stored pages are retrieved. Other strategies such as depth-first can slow down servers with repeated requests, as this strategy retrieves from a list of stored Web pages every descendant Web pages traversing all the existent levels of the directed graph (Poulter 1997). Also there is a series of guidelines proposed by an unofficial standard for robot exclusion that indicates to indexing robots which parts of a given server should not be accessed such as deep virtual trees, duplicated or temporary information or CGI-scripts with side-effects (Koster 1994b). Indexing robots access via HTTP a file named robots.txt, normally situated in the root directory of a given server (from a URL perspective). The following figure (fig. 2.2) shows an example of a robots.txt file that disallows the access to the directory /clients/pso/ to all indexing robots:
In the WebCrawler, Agents are invoked by the search engine ordering them to retrieve a given Web page. Agents can access different types of contents with several protocols such as HTTP and FTP. They respond to the search engine with an object containing the Web page or with specific error messages. The database comprises of a full-text index and additional information that represents the Web as a graph. The index is inverted with words pointing to documents that contain those words. The indexing process of a document activates a lexical analyzer that transforms the document into a collection of tokens or words. Common words are filtered through stop lists to prevent them being indexed and the rest of words are weighted by their frequency in the document divided by their frequency in a reference domain.

Other data stored in the database includes information about documents, links and servers. URLs are decomposed into objects that describe the server and the document referenced. Links in a document are treated as pointers to other documents. Each object type, documents, links and servers are stored in separate BTREE indexes.

In WISE (A World Wide Web Resource Database System) (Yuwono and Lee 1996) resources are kept updated by comparing the date of last modification of the stored Web page with the date of last modification obtained from the HTTP header of the original source Web page. The index is a table, which maps keywords to URLs containing the term. The word position attribute is used to store information for phrase searching.
Other attributes store information about title, date of last update and list of links of a given Web page. The entity-relational model of WISE is outlined in the following figure (fig. 2.3):

![Entity-Relationship Model of the WISE resource database](image)

Figure 2.3 Entity-Relationship Model of the WISE resource database

The main components are the index database, the indexing robot, the search engine and the user interface. The indexing robot in this case communicates with servers using only the HTTP protocol and retrieves only object of text format from remote servers, including non-HTML documents. Interactive (forms) and dynamically generated resources (i.e. CGI scripts), non-textual and application-specific objects are not processed by the indexing robot. Hyperlinks to other textual resources are followed in a breadth-first manner. The indexing robot respects the proposed standard for robot exclusion and also recognizes loops caused by circular symbolic links. The indexing robot starts crawling the WWW from a list of target sites to visit. Every time the robot finds a link that references to a new site, the URL is stored into a special list, which is appended to the target list for the next processing batch. The inverted file index is built upon extracting keywords from page titles, headings, anchors, words in italics, etc. Stop
words are removed and word stemming is performed on the remaining words to remove suffices.

In the Webcrawler the query server interacts with the index so users can enter keywords as their query and the titles and URLs of the documents containing at least some of the words are retrieved and presented to the user ordered by relevance.

In WISE query preprocessing involves removal of stop words, word stemming, and identification of phrases. The resulting keywords are scanned against the inverted file index to create a list of potentially relevant URLs that are processed by a ranking algorithm (TF\texttimes IDF). The relevance score of a document is based on the sum of the weights of the query terms that appear in the document. The weight of a term is a function of the term's occurrence frequency in the document (TF) and the number of documents containing the term in the collection (IDF). This algorithm gives higher weights to keywords that occur frequently in small sets of documents.

Currently there is been a new revival in the research of indexing robots with some interesting work proposed in (Cho et al. 1998). Due to the exponential growth of the WWW it is claimed that most indexing robots are not capable of covering the whole Web limited by their storage capacity, not being able to cope with the at least 1.5 TB of information available on the WWW. Another reason is the indexing process takes so much time that at some point the indexing robot will have to stop and revisit previously scanned Web pages to check for changes. As a result some parts of the Web may never be visited. Their research proposes to visit the most important part of the Web,
introducing algorithms and strategies capable of determining the importance of a given Web page.

The importance of a Web page can be defined by applying any of the following techniques:

- **Similarity to a Driving Query Q:** a query Q determines the indexing process, and the importance of a given Web page P is defined to be the textual similarity between the Web page P and the query Q. It is based on the term frequency, inverse document frequency (TFxIDF) algorithm mentioned in WISE.
- **Backlink count:** the importance factor is determined by the number of links that point to a given Web page P on the entire covered Web.
- **PageRank:** recursively defines the importance of a Web page to be the weighted sum of the backlinks to it. In other words, the importance of a given Web page P is dependent on the importance of the Web pages that point to P and the number of total outgoing links of every Web page that points to P.
- **Forward Link Count:** importance is determined by the number of outgoing links of a given Web page P. Permits to identify directory or index pages.
- **Location metric:** importance is a function of the location of a given Web page P. URLs ending with ".com" or containing the string "home" can be considered as more important. Web pages that are situated closer to the root directory of a given server are considered as more important.

Centralised indexing robots cover large spaces of the Web giving high recall (fraction of relevant documents retrieved by a query) and less precision (match between documents retrieved and a query). Precision is the limiting factor as many
of the documents retrieved by a given query can be irrelevant. Other limitation is that
database attributes as such are non-existent in Web pages because there is no restricted
vocabulary, all different kinds of topics and subjects are covered. As a result the only
generic attributes that can be defined are derived from the decomposition of URLs into
attributes that store information about protocols handled, servers, type of domain and
country. Other attributes can be defined from HTML tags such as title, meta-tags, etc. or
by meta-data specified by HTTP headers such as date of last modification, length, type
of document and so on.

With the advent of the Extensible Markup Language (XML) and the Resource
Description Framework (RDF), specific vocabularies can be defined for specific
communities and intelligent metadata can be associated to Web pages. As a matter of
fact, more structured and focused queries will be possible facilitating higher recall and
precision in this kind of search engines.

### 2.4.4 Meta-search engines

Meta-search engines provide a central interface to an arbitrary number of search
engines, posting queries in parallel, collecting, integrating and filtering the results.
Meta-search engines add an additional level of abstraction to Web searching. Meta-
search engines can simultaneously query several Internet search engines, interpret the
results, and display them into a uniform format.

The general Architecture of Meta-search Engines consists of three components: a
query dispatcher, interface agents or site wrappers and a display mechanism or
result integrator (Dreilinger and Howe 1997). The query dispatcher determines to
which search engines a specific query is sent. The interface agents are self-contained programs that interact with specific search engines. The interface agent adapts the user’s query format to the required format for a particular search engine and interprets native result formats returned by specific search engines. The display mechanism integrates and reformats results from individual search engines removing duplicates or verifying links. End-users submit queries via the meta-search engine’s user interface. The dispatcher determines which search engines to send the query. The interface agents of the selected search engines are activated submitting the query to their respective search engines. The results are converted into a uniform internal format, and finally integrated, filtered and formatted by a displayer module and the output is sent to user’s browser.

2.4.5 Subject Directory

Indexes are organised by subject or topic, which results in a more semantic indexing. Subject directory search engines differ in the range of subjects that they cover (general or specific: Subject Based Information Gateways), the classification system/scheme applied to organise the indexes and the information stored in order to describe the resources. Currently, the most popular subject directory search engine on the WWW is Yahoo. Yahoo classification’s system is proprietary, classifying resources by subject hierarchies (Callery 1997). Apart from the URL, Yahoo includes as a resource description of each Web page, the title and a brief description. Yahoo! allows keyword-based searches as any other search engine but its strength lies primarily in the subject hierarchy which it is organised. Yahoo! classification system is proprietary, non-compliant to any official standard although it can be compared with the early Dewey Decimal Classification. Thousands of submissions are received everyday, being examined by human beings. Specialised cataloguers check categories suggested by
users and organise the submissions. Each submission stores information about title, URL, a short description and category suggested. Categories and subcategories are added as the hierarchy is being developed, maintaining a consistent vocabulary in the naming of common subcategories. Sites can be placed under specific regions constraining the scope of a given subcategory. Users can search for information by browsing the subject tree hierarchy or specifying keywords that are directed to the database or redirected to keyword-based search engines.

2.5 Database Approaches to searching the WWW

The literature presented in this section is the most relevant to our research. The section describes the most influential approaches in attempting to convert the WWW into a highly structured database. Database approaches to searching the Web, opposed to keyword-based search engines, focus on the discovery of semantics on the Web and the generation of schemas associated to particular domains. One of the first theoretical research papers related to the subject was the Multi-Layered Database approach proposed by Han et al. in 1995. Up-to date the most serious and consolidated study is the one presented by Paolo Atzeni et al. in 1997, the Araneus data model and view language. The section concludes with an introduction to Virtual Database technology (Rajamaran 1998), which consists of a commercial system adapted to convert source data in HTML format into structured relational tuples.

2.5.1 A Multi-layered Approach: MLDB

In (Han et al. 1995) a Multiple Layered Database (MLDB) approach is proposed to facilitate resource discovery in the Web. An MLDB is a database stratified in several layers with the lowest layer named as layer-0 which corresponds to the WWW and the
higher layers from layer-1 to above that store generalised information extracted from the lower layers. By transforming the semi-structured layer, the WWW into structured databases, database techniques can be applied to manage and retrieve information at this layer. In their proposal it is claimed that the layer-1 database is commonly very large and widely distributed. Therefore to obtain an efficient resource discovery this layer should be generalised into higher layers. All non-primitive layers (i.e. layer-1 and above) can be constructed based on an extended-relational model with the capability to deal with complex data types such as sets, lists, etc.

The major components of an MLDB are:

- A database schema
- A set of concept hierarchies and
- A set of generalised database relations at the non-primitive layers mapping to files in the Web.

The database schema stores metadata describing the types, ranges and structures used, relations at different layers, attributes associated, data statistics about the relations at different layers and so on. The set of concept hierarchies consists of a set of predefined concepts and the definition of hierarchies among them which assist the system to generalise lower layer information into higher ones and address queries to the appropriate layer.

Layer-1 is a detailed abstraction of the WWW, substantially smaller but rich enough to preserve most of the interesting pieces of information of the primitive layer. However it is still too large and too widely distributed for efficient storage, management and search
on the Web and must be generalised into higher layered databases. The construction of layer-1 requires standardisation of the schema and the development of software that can build it automatically. Mechanisms must be developed to handle complex structures (list, sets, etc.), missing attribute values, inconsistent values, system generated object identifiers to identify objects independent of URL addresses, and so on. Distributed indexers named as “local software robots” must be developed to automate the construction process of layer-1 and released in regional networks.

A set of standard concept hierarchies will serve as a common reference for the construction of higher layers and the retrieval of information between layers. Hierarchies are built with the definition of contains-lists and alias-lists. Contains-list specifies a concept and its immediate subconcepts; alias-list specifies the list of synonyms of a given concept. These definitions are critical for the declaration of additional relational operations that extend the semantics of traditional selection and join. Concepts are generalised at different layers and search conditions may not match exactly the concept level of the current available layer. Keyword-based search in an MLDB precises four additional relational operations:

- **coverage**: a concept A covers concept B if A or A’s synonym is an ancestor of B or B’s synonym
- **covered_by**: a concept A is covered by concept B if A or A’s synonym is a descendant of B or B’s synonym
- **synonym**: a concept A is a synonym of concept B if A and B are in the same alias-list
- **approximation**: A concept A is an approximation of concept B if A or A’s synonym is a sibling of B or B’s synonym
A new query language, NetQL, is described with the incorporation of four new operators corresponding to the operations mentioned above (see Table 2.2):

<table>
<thead>
<tr>
<th>NetQL Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covers</td>
<td>Coverage</td>
</tr>
<tr>
<td>Coveredby</td>
<td>Covered_by</td>
</tr>
<tr>
<td>Like</td>
<td>Synonym</td>
</tr>
<tr>
<td>Closeto</td>
<td>approximation</td>
</tr>
</tbody>
</table>

Table 2.2 NetQL operators

The syntax of NetQL is:

```
(SELECT | LIST | DESCRIBE) ({Attribute} | *)
FROM {Relation}
[RELATED-TO {Concept}]
[IN {location}]
[WHERE {Conditions}]
```

LIST is used to retrieve summaries; DESCRIBE can be used to discover and describe general characteristics of the data; RELATED-TO locates related subjects and IN locates geographical regions. The conditions specified in the WHERE clause can include NetQL operators.

In spite of the fact that the research proposed in (Han et al. 1995) did not end up with any implementation it can be considered as one of the most interesting theoretical proposals trying to transform the WWW into a structured database.

2.5.2 The Araneus Data Model and View Languages

The Araneus Data Model proposes the definition of database-like schemas that model homogeneous sets of HTML documents situated in structured servers on the Web (Atzeni et al. 1997a) (Atzeni et al. 1997b). A page oriented data model called the Araneus Data Model (ADM) is used to model the schema associated to pages that are
similar not only in their hypertextual structure (link structure) but also in their textual organisation (HTML tags). Furthermore, both a database and hypertextual view language is proposed to support a two-way data restructuring process. The ULIXES language is used to build relational views over Web data, and the PENELOPE language allows the conversion and reorganisation of structured data into HTML format giving the perception of the construction of virtual Web servers.

2.5.2.1 The Araneus Data Model

In the ADM each Web page is considered as an entity identified by its URL with a set of attributes. The notion of page scheme is introduced in order to model sets of homogeneous Web pages. Each page scheme is identified by a page name. Page scheme attributes can be monovalued corresponding to text, images or links to other pages, or multivalued corresponding to collection of objects modelled as lists of tuples which in turn can be nested. Some pages modelled in an ADM scheme may be unique, including normally at least the home page of each structured server. The general syntax for describing a page scheme is:

```
PAGE SCHEME Page_ID [UNIQUE]:
    {Attribute: (TEXT | IMAGE | LINK TO Page_ID | LIST OF ({Attribute: (….);})) ;}
END PAGE SCHEME
```

The scheme can be represented as a directed multigraph, having page schemes as nodes in the scheme graph. Page instances can be viewed as nodes of a tree connected by links with simple or complex values.

2.5.2.2 The Araneus database view language: ULIXES
ULIXES has been designed to extract data from the Web based on ADM schemes. The data extraction process is based on the notion of navigation in the site using navigational expressions representing paths in the site graph. These navigations allow to follow links between different pages and to explore the hierarchical structure of a page. The general form of a navigational expression is:

```
Page_ID.Attribute.Link to Page_ID -> Page_ID.[Attribute.Link to Page_ID]
```

The dot operator (.) indicates navigation inside pages and the link operator (->) navigation between pages. A navigational expression corresponds to all paths in the server graph starting from the unique instance of a page scheme and traversing the graph up to nodes corresponding to attributes of the destination page. Each of these navigations can be represented as a tuple of values, one value for each monovalued attribute associated to nodes in the navigation. Each navigational expression can be represented as a relation. The DEFINE TABLE statement is used to define relational views over ADM schemes which are based on navigational expressions. The syntax of a DEFINE TABLE statement is:

```
DEFINE TABLE Relation_ID({Attributes}) AS Navigational_Expression IN ADM_Scheme_ID USING {Monovalued_Page_ID_Attributes} WHERE {Condition(Monovalued_Page_ID_Attributes)}
```

Each DEFINE TABLE statement generates a relation which can be imported into a RDBMS and be queried using SQL. Each page is considered as a nested relation, in which list attributes are modelled using tables. For each page involved in a navigational expression, a table is generated and then the resulting tables are joined and projected using SQL.
2.5.2.3 The Araneus hypertextual view language: PENELOPE

PENELOPE is used to transform relational views into Web pages with a new structure that does not appear in the original server. This restructuring process is used to define derived or virtual servers that materialise new views over the original data. Query results are organised according to the definition of new page schemes.

The restructuring process in Araneus is composed of the following steps:

- Navigational expressions over the structured servers are defined and relational views as well with DEFINE TABLE statements.
- Additional views are defined in SQL with CREATE VIEW statements
- New pages are specified with DEFINE PAGE statements

In order to create pages from tuples in nested relations two important concepts are introduced, the notion of local URL and structures. Local URL is a local file name used to identify pages and can be either a string constant or can be generated dynamically with the use of the URL function over attributes in relations. Structures are basically ADM types that describe how page structures can be created starting from attributes in relations. The syntax of a DEFINE page statement is:

```
DEFINE PAGE Page_ID [UNIQUE]
AS URL (Constant | URL(<Relation_ID_Attribute>));
{Attribute: (TEXT<Relation_ID_Attribute> | LINK TO Page_ID (<Relation_ID_Attribute> | URL(<Relation_ID_Attribute>)) | LIST OF [{Attribute: …;}] );}
FROM Relation_ID
```

The relation Relation_ID is extended by adding local URLs to each tuple as new attributes. A page URL that identifies the page is generated. Also, a link will be
generated, for every structure of the form of LINK TO Page_ID in the DEFINE PAGE statement. This relation is projected onto the attributes defined in the page structure and will include the page URL. The projected relation is nested according to the defined structures. At the end an HTML document is generated for each tuple in the resulting relation.

In essence the Araneus system not only allows the extraction and conversion of semi-structured data into a relational structure but also the restructuring of the captured data into hypertext views.

### 2.5.3 Virtual Database Technology

Virtual Database (VDB) technology makes the World Wide Web and other external data sources behave as an extension of a RDBMS (Rajaraman 1998). VDBs gather, structure and integrate data from disparate data sources providing the application programmer with the appearance of a single, unified relational database system. The main components of the Virtual Database management system (VDBMS) are the Virtual Database Server and a family of specific site wrappers that interface to external data sources. Figure 2.4 shows diagrammatically the main components of a VDB system.
VDBs are accessed from applications through the VDB server, which directs queries to the required wrappers and finally integrates query results from the set of wrappers. Wrappers interface with the target Web sites via the HTTP protocol and HTML. Client applications that can issue SQL queries access wrappers with the aid of the JDBC API. An SQL query issued to the wrapper provokes in some cases the automatic filling of HTML forms on a particular site, the navigation and parsing of the results in HTML format and the transformation of data into relational tuples. **Wrappers use extraction rules to obtain attributes values from Web pages, data transformation/normalisation rules to adapt the data to the schema, and data integrity rules to force integrity constraints.**

Client JAVA™ applications can interact with several data sources, interfacing directly with the required wrappers. The application treats each data source as a separate JDBC source with its own schema, and therefore must connect to each source separately and integrate the data dynamically as needed. The VDBMS exposes tables in multiple data sources as virtual tables in a single VDB supporting full RDBMS functionality over
virtual tables such as view definitions and query processing across sources. From a higher level of abstraction the VDBMS receives queries that are decomposed by the query processor which sends the query to the required Web sites and finally integrates the received results.

In synthesis, VDBs converts, restructures and integrates semi-structured data from different sites into highly structured databases. It is assumed that this conversion demands the definition of schemas associated to the data extracted, which serve as a guide to the extraction process.

2.6 Other related work

We shall mention briefly a totally different strategy for data modelling on the WWW. In (Abiteboul 1997) the data stored on the Web is generally considered as semi-structured, neither raw data nor strictly typed, not table-oriented as in any relational model or sorted-graph as in object databases. Semi-structured data presents many anomalies respect to structured data:

- The structure is irregular, may consist of heterogeneous elements, some of them incomplete
- The structure is partial where parts of the data may lack structure.
- The notion of schema is often posterior to the existence of data
- The schema may be very large as a consequence of heterogeneity
- The schema is rapidly evolving and may be updated as easily as data
- The distinction between schema and data is blurred
In order to deal with all these anomalies, the concept of data guide was introduced in the Lorel Query Language for semi-structured data, to focus on non-conventional approaches to typing found in most semi-structured domains. A data guide provides some information about the current type of data, so accuracy is traded in for simplicity. All new data is accepted, eventually at the cost of modifying the data guide.

In synthesis, a data guide provides a loose description of the structure of data. In the context of semi-structured data, query languages have to be more flexible than in conventional DBMS and data typing should be less strict. These queries languages should enable standard database-style query primitives, navigational queries, pattern searching, and give the possibility to query both data and schema at the same time. Another interesting approach has been proposed in the WebOQL system (Arocena and Mendelzon 1998). The WebOQL data model supports not only semi-structured data, but also record-based data and structured documents as well. The query language permits the restructuring of any instance type into an instance of any of the rest. In WebOQL a Web is an abstraction of the data model capable of modelling a collection of related pages, a subnetwork of Web pages or even the entire WWW.

WebOQL views the Web as a database that can be queried using a declarative language exploiting document structure. WebOQL is based on a schema-free data model based on ordered trees, viewing the Web as a graph of trees. The query language is able to navigate, query and restructure graphs of trees. The structure of pages is captured in the queries themselves, being capable of querying pages with irregular structure and pages whose structure is not fully known.

Both approaches attack the problem of converting the WWW into a database with a totally different strategy compared to the one proposed in this thesis. The data model is
either loosely typed or schema-free. In the spirit of databases, we advocate the
discovery and definition of schemas associated to specific domains on the Web and we
precise of significantly typed data structures.

Another field closely related to the research proposed in this thesis consists of Internet
metadata formats. In our methodology, the transformation of the WWW into a highly
structured database it is entirely based on the use of metadata. For this reason we cite
some of the main uses of metadata and we mention and describe some of the most
influential Internet metadata formats.

2.6.1 Internet Metadata formats

Metadata is information about data. Metadata describes some aspects of data on the
Internet (Ianella and Waugh 1997). Metadata can consist of records that refer to digital
resources available across the Internet that may be capable of existing in a separate
physical form from the metadata record itself (Heery 1996). Metadata can be used to
present in a compact form the meaning of the data being stored, to allow users to
search for the data and to establish relationships with other resources.

The basic model used for metadata is known as the "attribute type and value" model
(Ianella and Waugh 1997). Metadata refers to a set of facts about resources
represented as attributes. An attribute type identifies what kind of information the
attribute contains and an attribute value contains the metadata itself.
A number of Internet metadata sets have been proposed in order to facilitate the discovery of resources on the Internet and the WWW. Metadata standards are used to define:

- The set of attributes which can be used to describe resources
- The set of attributes that are mandatory or optional
- The meaning or semantics of each attribute and
- The syntax or set of permitted values for particular attributes.

The definition of Internet Metadata standards has been a very conflictive issue, due to the heterogeneous nature of the information stored, the disparity of communities involved in pushing information on the Internet and the non-existence of a common vocabulary that can represent universally information and communities. Currently the most popular use of metadata within Web pages consists of a limited metadata set that contains two elements, Description and Keywords (see Fig. 2.5). The majority of search engines scan Web pages giving precedence to the keywords found in these meta tags.

```html
<META name="Description" content="Database Querying on the World Wide Web">
<META name="keywords" content="search engine, database, metadata, WWW, ontologies">
```

**Figure 2.5** Example of Description and Keyword meta tags

The use of this limited set of meta tags can improve slightly the location of resources on the WWW, although the Web precisely extensible and adaptable metadata formats capable of representing the diversity of information stored. May 1998 be considered in the future as an important year for the WWW thanks to the adoption of XML and the first universal WWW metadata standard approved by the W3C? The standard we are referring above is the Resource Description Framework (RDF) which may have a
dramatic effect on how the Web is indexed, increasing the level of precision and recall in WWW search engines.

We shall mention some of the Internet metadata formats that have influenced more in our research: IAFA templates and the Dublin core set. Also we provide a small introduction to RDF in the contents of this section.

2.6.1.1 IAFA Templates

Internet Anonymous FTP Archive (IAFA) Templates were designed to facilitate effective access to FTP archives describing the contents and services available from the archive. FTP archive administrators were responsible of describing the different kinds of resources available from their own archives. Different template formats were designed, adapted to the various categories of information stored in FTP archives such as images, documents, software and so on. FTP archive sites that were compliant to a common set of indexing and cataloguing guidelines permitted automated data collectors to gather sets of records from FTP anonymous sites. One of the first search systems to adopt IAFA templates was the ALIWEB system (Koster 94a). The effort required by FTP administrators to create record descriptions associated to each FTP site, ended up in the final disuse of ALIWEB.

The format of IAFA records is in simple ASCII text. Data elements are defined as attribute/value pairs and are of variable length. Templates include simplified bibliographic fields such as title, author, publisher and language and other non-bibliographic information. Additionally, URL and e-mail links to authors and publishers
are included. The following figure (fig. 2.6) shows an example of an IAFA template record (partial) used to describe an article (document) about children with cancer.

<table>
<thead>
<tr>
<th>Template-Type:</th>
<th>DOCUMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle:</td>
<td>83381296-7713</td>
</tr>
<tr>
<td>Title:</td>
<td>Better care of the child with cancer</td>
</tr>
<tr>
<td>URI-v1:</td>
<td><a href="http://www.dundee.ac.uk/MedEd/Webupdate/child/cancer.htm">http://www.dundee.ac.uk/MedEd/Webupdate/child/cancer.htm</a></td>
</tr>
<tr>
<td>URI-v2:</td>
<td><a href="http://www.dundee.ac.uk/MedEd/Webupdate/child/cancer.htm">http://www.dundee.ac.uk/MedEd/Webupdate/child/cancer.htm</a></td>
</tr>
<tr>
<td>Author-Name-v1:</td>
<td>Richard Stevens MRCP MRCPPath</td>
</tr>
<tr>
<td>Author-Job-Title-v1:</td>
<td>Consultant Oncologist</td>
</tr>
<tr>
<td>Admin-Email-v1:</td>
<td><a href="mailto:update@dundee.ac.uk">update@dundee.ac.uk</a></td>
</tr>
<tr>
<td>Description:</td>
<td>An article on childhood cancers from Web Update, covering a Summary of the various types of malignancy……</td>
</tr>
<tr>
<td>Publisher-Name-v1:</td>
<td>Centre for Medical Education, University of Dundee</td>
</tr>
<tr>
<td>Record-Created-Email:</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.6 Sample IAFA template record (partial) representing a document

2.6.1.2 Dublin Core

The Dublin Metadata Core Element Set is a core list of metadata elements agreed at the OCLC/NCSA Metadata Workshop in May 1995. The objective was to promote and develop metadata elements required to facilitate the discovery of resources on the Internet. The current metadata set consist of 15 elements falling into three groups:

- **Elements related to the content of the resource**: title, subject and keywords, description, source, language, relation and coverage.

- **Elements related to intellectual property**: author or creator, publisher, other contributors and rights management

- **Elements related to the instantiation of the resource**: date, resource type, format and resource identifier

The following table (table 2.3) summarises the information stored by each element:

<table>
<thead>
<tr>
<th>Element</th>
<th>Information stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>The name given by the creator to the resource</td>
</tr>
<tr>
<td>Subject and Keywords</td>
<td>The topic of the resource expressed as keywords or phrases that describe the content of the resource</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Description</td>
<td>Textual description, including abstracts of document-like objects, or content descriptions in the case of visual resources</td>
</tr>
<tr>
<td>Publisher</td>
<td>The entity responsible for making the resource available in its present form</td>
</tr>
<tr>
<td>Other Contributor</td>
<td>A secondary person or organisation who has contributed intellectually to the resource</td>
</tr>
<tr>
<td>Date</td>
<td>Date of creation of the resource</td>
</tr>
<tr>
<td>Resource Type</td>
<td>The category of the resource: home page, novel, poem, technical report, essay, etc.</td>
</tr>
<tr>
<td>Format</td>
<td>The data format of the resource</td>
</tr>
<tr>
<td>Resource Identifier</td>
<td>A string or number that identifies uniquely the resource. For Internet resources normally the URL of the resource</td>
</tr>
<tr>
<td>Source</td>
<td>Information about a second resource from which the current resource is derived</td>
</tr>
<tr>
<td>Language</td>
<td>An abbreviation indicating the language of the contents of the resource</td>
</tr>
<tr>
<td>Relation</td>
<td>An identifier of a second resource and its relationship to the current resource</td>
</tr>
<tr>
<td>Coverage</td>
<td>The spatial or temporal characteristics of the intellectual content of the resource. Spatial coverage can be specified with the use of coordinates or site names, for temporal coverage date/time ranges</td>
</tr>
<tr>
<td>Rights Management</td>
<td>A rights management statement, or a link to a rights management statement, etc.</td>
</tr>
</tbody>
</table>

Table 2.3 Dublin Core Elements

Each element is repeatable and optional, and may also have a sub-type and sub-scheme information providing additional semantics to the metadata values.

An example of Dublin Core meta tags describing the current thesis is shown below:

```xml
<META NAME="DC.title" CONTENT="Database Querying on the WWW">
<META NAME="DC.subject" CONTENT="(SCHEME=keyword) Search Engine, Database, Metadata">
<META NAME="DC.author" CONTENT="(TYPE=name) Carlos F. Enguix">
<META NAME="DC.author" CONTENT="(TYPE=email) cfe01@uow.edu.au">
<META NAME="DC.author" CONTENT="(TYPE=affiliation) DSL">
<META NAME="DC.publisher" CONTENT="(TYPE=name) University of Wollongong">
<META NAME="DC.date" CONTENT="(TYPE=creation) (SCHEME=ISO31) 1998-06-02">
<META NAME="DC.date" CONTENT="(TYPE=current) (SCHEME=ISO31) 1998-06-02">
<META NAME="DC.format" CONTENT="(SCHEME=imt) text/html">
<META NAME="DC.language" CONTENT="(SCHEME=ISO639) en">
```

Figure 2.7 Example of Dublin Core meta-tags
2.6.1.3 Resource Description Framework

The Resource Description Framework (RDF) is a metadata standard designed to provide interoperability between applications that exchange machine-understandable information on the Web and to enable automated processing of Web resources (W3C 1998). RDF metadata can be used in resource discovery to provide more accurate descriptions and semantics to resources improving recall and precision in search engines.

In the context of RDF, metadata is data describing Web resources. RDF uses the Extensible Markup Language (XML) as the encoding syntax for the metadata. RDF is capable of describing any kind of resource identified by a given Uniform Resource Identifier (URI). RDF data consists of nodes and attached attribute/value pairs. Nodes may refer to any Web resource identified by a given URI or other instances of metadata. Attributes are named properties of the nodes and the associated values can hold atomic values (strings, numbers, etc.), other resources or metadata instances. Fig. 2.8 shows an example of metadata in abbreviated RDF/Dublin Core syntax describing the current thesis (Miller and Iannella 1998).

```xml
<?xml:namespace href="http://www.w3.org/TR/WD-rdf-syntax#" as "DC"/>
<RDF:RDF>
                DC:Title  ="Database Querying on the WWW"
                DC:Creator ="Carlos F. Enguix"
                DC:Subject ="Search Engine, Database, Metadata"
                DC:Description ="This thesis is about database search engines"
                DC:Publisher ="University of Wollongong"
                DC:Format  ="text/html"
                DC:Type    ="Thesis"
                DC:Language ="en"
                DC:Date    ="1998-06-02" />
</RDF:RDF>
```

Figure 2.8 Example of abbreviated RDF syntax
RDF does not contain any predefined vocabulary for authoring metadata, although it is expected that standard vocabularies will appear adapted to certain communities or domains.

2.7 Summary

In the review of the literature, we have been travelling in time, having as a starting point the beginning of the Internet, describing some of the first Internet search engines, explaining how important was the invention of the WWW. We presented a taxonomy of WWW search engines, differentiating two types of classifications. The first classification is determined by the discovery of semantics and the definition of schemas associated with certain domains on the Web. This classification "divides the Web" into keyword-based search engines and database approaches to searching the Web. Keyword-based search engines do not precise of the definition of schemas representing parts of the Web, whereas database approaches do. The second classification is subjugated to a very particular viewpoint, the gathering of data. This classification includes client-based/real-time search engines, distributed indexing search engines, robot-based centralised indexing, meta-search engines and subject directory search engines. We consider as the most relevant section, the section entitled as Database Approaches to Searching the Web, specially the Araneus data model. The restructuring of Web pages starting from relational tuples (PENELOPE) is out of the scope of this thesis. On the other hand, the discovery of homogeneous set of Web pages, the definition of schemas associated with these Web pages, the type of attributes that must be stored are common phases in both approaches. The divergence of ideas appear when considering the methodology involved. The Araneus data model is entirely based on the use of site-specific wrappers, whereas our methodology is entirely based on
metadata. The Araneus data model focuses on restricted domains (i.e. Web servers), on the contrary we focus on logical domains capable of covering large sets of Web servers.

In the Other Related Work section an introduction to loosely typed or schema-free database approaches to searching the Web was presented. Finally we concluded with the notion of metadata and we described some of the most important metadata formats. May XML and RDF resolve the resource discovery issue on the WWW? May XML and RDF affect drastically the way search engines index information? These questions may be answered in a very close future.
Chapter 3.
Research Methodology

3.1 Introduction
3.2 Proof of concept Research Methodology
3.3 Design Framework
3.4 Implementation environment
3.5 Evaluation Methodology
3.6 Summary
Chapter 3. Research Methodology

3.1 Introduction

As cited in section 1.3 (Research Objectives), the purpose of this research is to provide an alternative to keyword-based search engines, in the form of domain-specific database search engines that gives us the possibility of defining more focused and structured queries using a predicate calculus-based structured query language such as SQL. A prerequisite to realising our proposed alternative is the demonstration that certain domains on the Web present high regularity either in their content or organisation, which implicitly indicates the presence of an underlying schema. The output of the research will be the development of an ORDB search engine, which represents the materialisation or proof that our hypothesis is feasible. Nunamaker et al. (1991) clearly states that the building of a system in and of does not constitute research unless new technologies or concepts are introduced in the construction of the system. We believe that the construction of UniGuide constitutes an innovative way of restructuring and organising Web information.

Nunamaker et al. 1991, mentions that research follows a pattern of "problem, hypothesis, analysis, argument". From our viewpoint we can synthesise our research process as follows:

- **Problem:** current keyword-based search engines do not address properly the resource discovery problem on the WWW. As the Web expands exponentially, the problem expands as well.
• **Hypothesis:** it is possible to provide an alternative to keyword-based technology. One of the alternative is to build database search engines that offer more focused and structured query capability.

• **Analysis:** the materialisation of the hypothesis is obtained with the definition of schemas associated to logical domains and the development of the database search engine, based on the alternative conceptualisation of the problem.

• **Argument:** the feasibility of the construction of database search engines, and the definition of more focused and structured queries will confirm the validity of the hypothesis.

This view of research methodology permits system development to be an acceptable piece of evidence in support of the proof, whereas proof can be considered as any convincing argument in support of a valid hypothesis. System development can be thought of as a "proof by demonstration".

In addition (Hartmanis 1995) states that computer science advances are often demonstrated and documented by a "dramatic demonstration" rather than a dramatic experiment as in physical sciences. A demo constitutes the proof that demonstrates the feasibility of what was thought to be impossible or not feasible. A demo implicates the development of a research prototype system.

This chapter is organised as follows: firstly we explain in more detail what constitutes the proof-of-concept research methodology. Secondly, we present the stages that we have followed in our research: analysis and design, prototype construction and system evaluation.
3.2 Proof of Concept Research Methodology

A research process involves understanding research domains, asking meaningful questions and applying valid research methodologies to address these questions. A research domain is the subject under study in a research project, whereas a research methodology consists of the combination of the process, methods, and tools that are used in conducting research in a research domain. Results from a successful research project may contribute to the body of knowledge by expanding knowledge in a given domain. The following figure (fig. 3.1) depicts diagramatically the relation between research process and the body of knowledge:

![Figure 3.1 Relation between Research Process and Body of Knowledge](image)

Without a thorough and complete understanding of a research domain, researchers may formulate meaningless hypothesis, which can lead to incorrect conclusions. Where valid hypothesis can be obtained, system development can be used as a viable research methodology. Systems must be developed in order to test and measure the underlying concepts. Therefore system development is a key element of Information Systems (IS) research. A multi-methodological approach to IS research consists of four research strategies, depicted in figure 3.2:
• **Theory building**: includes the development of new ideas and concepts and the construction of new methods or models. Theories are used to suggest research hypothesis, guide the design of experiments and so on.

• **Experiments**: includes research strategies such as computer and experimental simulations. Experimental designs are guided by theories and facilitated by systems development. Results from experimentation may guide the refinement of theories and the evolutionary improvement of developed systems.

• **Observation**: includes research methodologies such as case studies and sample surveys. Used to help researchers in the formulation of specific hypothesis to be tested through experimentation.

• **Systems development**: includes concept design, the construction of the architecture of the system, prototyping, product development and technology transfer. Concept design consists of the adaptation of technological and theoretical advances into practical applications. Prototyping is used as a proof-of-concept to demonstrate feasibility. Successful prototypes are expanded into fully operational production.
systems. The transfer of technology to end-users implies the consolidation of the system development life cycle.

System development can be considered as a "super-methodology" that includes a hierarchy of sub-methodologies, depicted in figure 3.3:

Figure 3.3 Principal parts of a system development research methodology

- **Construct a conceptual framework**: researchers must find meaningful research questions, investigate the system functionalities and requirements, etc. The conceptual framework leads to theory building.
- **Develop a system architecture**: includes the definition of system components and the interaction among them, identification of environmental constraints and objectives of the development efforts and the definition of functionalities of the resulting system to achieve the stated objectives.
• **Analyse and design the system:** involves the understanding of the research domain, the application of relevant scientific and technical knowledge (knowledge domain), the creation, synthesis and evaluation of proposed alternative solutions.

• **Build the prototype:** used to demonstrate the feasibility of the design and the usability of the functionalities of a system development research project.

• **Observe and evaluate the system:** includes the testing of the performance and usability of the system, and the observation of the impact on individuals, groups or organisations.

Only two stages, design, implementation have been applied in our research. The evaluation stage is included in this chapter for future work. In fact, in the introduction chapter we discuss the current situation of the research domain, the problems to be solved, and the research objectives. In the review of the literature we present an overview of the past, present and state-of-the-art of search engines. In the following chapter (4. Design and Implementation of an Object-Relational Database Search Engine) we present a detailed explanation of the development of a research prototype database search engine.

As indicated in (Nunamaker et. al 1991) the use of system development as a research methodology in IS should conform to the following criteria (Nunamaker et al. 1991):

• The purpose is to study an important phenomenon in areas of IS through system building

• Results must make a significant contribution to the domain

• The system is testable against all the stated objectives and requirements

• The new system provides better solutions than existing systems

• Experience and design expertise gained from building the system can be generalised
3.3 Design Framework

The information stored on the WWW is so extensive, that the extraction of abstract models that represent in a coherent form part of the Web poses in a serious challenging research problem. Basing ourselves on the concept of ontologies or "existence of things in a domain", data models can be extracted from well-demarcated Web domains that present high regularity in their contents and organisation.

The nature of the relatively less structured, irregular and idiosyncratic data on the Web, forces us in some ways to use more powerful models that are capable of representing the type of information stored on the WWW. Due to the fact that the WWW constitutes a very dynamic environment, and that the information stored on Web pages can be considered as semi-structured and loosely typed, our data model is based on a hybrid object-relational model capable of representing Web information in a consistent way. An object-relational model (ORM) permits us to define abstract data types (ADT's), complex objects, inheritance, rules, and so on. In addition, an ORM is more suited for a loosely typed environment such as the Web because it gives the possibility to define more dynamic or evolutionary data integrity constraints. In addition, we have modelled a new type of metadata that is capable of representing core information associated to the contents of Web pages, in a precise and compact form. This information represents the interface between the Web and the UniGuide object-relational database (ORDB).

In the next section we shall explain the implementation environment, which is based on the integration of diverse technologies. The complexity of the system is such, that a graphical model that is capable of representing in an intuitive way the integration
between different components is required as a serious documentation methodology that can assist in the understanding and evolution of the system. After studying a series of graphical models that represent Web-based systems, few of them seemed to be suitable for our needs. As a result, we were forced to invent a new type of notation capable of depicting in a compact and intuitive way the complexity of Web-based systems. The graphical model in question is named as "WebApp Structure chart", which stands for Web Application Structure Chart. WebApp structure charts are backed up by Top Level Navigational Charts (TLNC) that present in a very simple way the possible navigations on a Web application. WebApp structure charts are capable of differentiating client and server components.

3.4 Implementation environment

It would be a fallacy to say that ILLUSTRÁ™ was chosen as the ideal technology to deal with the research proposed in this thesis. As a matter of fact, the order of the factors are altered, the Business Systems department received a grant from Informix in order to experiment with some feasible research project. Discovering the features available in ILLUSTRÁ, enlightened us to focus our research towards Web technology. The architecture imposed by ILLUSTRÁ, gave us retroactively the possibility of defining more complex models capable of representing information stored on the Web. ILLUSTRÁ is characterised mainly for the use of hybrid object-relational constructs, rules, dynamically loadable executable external C functions, inheritance, and so on. In addition, the Web DataBlade® module is included, which provides the interface between the WWW and the UniGuide ORDB. A review of hybrid object-relational database technologies in general and ILLUSTRÁ in particular is presented in chapter 4 (Design and Implementation of an Object-Relational Database Search Engine).
3.5 Evaluation Methodology

Evaluation is included in this section as future work. A part of the evaluation of search engines can be performed in an unobtrusive way, by simply analysing Web log files, the use of cookies, by defining auditing rules that are fired automatically when end-users query the database or by extending external C functions called by *UniGuide* Web applications:

- **Web log files**: Web server log files track the hosts that access a given Web site. A typical line includes the hostname or the Internet Protocol (IP) address of the browser that connected the database search engine, the date and time of the request, etc.

- **Cookies**: a cookie is a small command string that the Navigator browser stores in the cookie.txt file, and serves as a general mechanism to store and retrieve information on the client machine. It is possible to store the number of accesses to the database search engine in this file and other information.

- **Auditing Rules**: when an end-user specifies a given query, ILLUSTRATA gives us the possibility to define certain operations before, on or after a SELECT statement is executed. A simple way of defining an auditing rule will consist on the increment of a counter that stores the number of total queries against tables.

- **Extending external C functions**: all Web applications interact with ILLUSTRATA databases, and the majority of them rely on the execution of dynamically loadable executable external C functions. The parameters passed to these external functions, in the case of queries, includes the table/set table being accessed, a list of column names involved in the query, and a list of associated values. It can be pretty straightforward to capture those values and insert them in an auditing table, which
may allow us to generate statistics over frequency of terms and attributes being queried.

In addition, we may test the usability of the prototype with an experimental group of end-users in order to determine the consistency and efficiency of the graphical user interface (GUI). By combining all these methodologies, we will be able to generate accurate statistics of the usability of the system. Other evaluations may include database or indexing robot performance issues.

3.6 Summary

In the first chapter it is clearly stated that the building of a system does not constitute research by itself. Only when new technologies or concepts are introduced in the construction of the system can the process be considered as legitimate research. We argue that the development of UniGuide contributes to research in view of the innovative strategies to structuring and searching information on the WWW. Only three of the multi-methodological approaches involved in the system development research methodology are used: analysis and design, implementation of the prototype and evaluation.
Chapter 4.
Design and Implementation of a Domain-specific Object-Relational Database Search Engine

4.1 Introduction

4.2 Modelling specific domains on the WWW

4.3 Object-Relational Model of UniGuide

4.4 Transformation of the model into an object-relational database

4.5 Architecture

4.6 End-user perspective: User Interface Sections

4.7 Summary
Chapter 4. Design and Implementation of a Domain-specific Object-Relational Database Search Engine

4.1 Introduction

This thesis began as an exercise in attempting to develop semantic models that covered different portions of the Web. The first phase included a generic low-level model similar in concept to the one adopted by WebSQL (Arocena et al. 1997) differentiating Web pages, Links, Images, Mime types, and Meta tags. This initial approach was discarded, as it was conceived to capture the internal components of a given HTML document and not the semantics associated with the intellectual contents of the document itself. The next move consisted in identifying a series of Web servers that presented similar content. Influenced by the approach given by (Han et al. 1995) in their Multi-Layered Database Approach, different logical domains were being modelled in parallel. The initial prototype domains covered software companies and the Australian Universities. As the modelling process was progressing we realised the complexity involved, especially in the University domain. As a matter of fact, the lesson learned from the Han et al. simulation, is that it is practically impossible to follow such strategy. The Web is so heterogeneous and so complex, that any attempt to develop a model to represent the semantics of the entire Web is unlikely to yield useful results. We need to focus more realistically on restricted logical domains whose ontologies can be explicated and modelled. This strategy of "divide and conquer" involves identifying and isolating reasonably well-demarcated subsets of the Web consisting of a collection of homogeneous Web sites whose structure can be modelled. This implies the existence of generic entities or concepts in the domain and the stable and predictable inter-
relationships among them. One should not lose sight of the fact that any modelling is a process of abstraction. Hence it is impossible to model all the different variations of a given entity or to incorporate all possible entities in a "characteristic" domain. In view of the similarity in structure that exists within the domain, it is our contention that all significant and common entities and their inter-relationship can be represented by our approach which is consistent with the observation by Atzeni et al. (1997) that in structured servers and intranet applications, the hypertext organisation of the Web pages tend to mirror the underlying structure of the organisation (or domain).

This chapter is organised as follows:

Different modelling strategies are introduced in the subsection 4.2 (Modelling specific domains on the WWW), concluding with an explanation that justifies over the use of an object-relational model for the development of a schema-based object-relational database search engine entitled as UniGuide. In sections 4.3 and 4.4, the object-relational model for the "Australian Universities Domain" and the transformation of the model into an object-relational database are presented. In the following section, section 4.5 is presented a detailed analysis of the main components used in the construction of the database search engine. Server-side and client-side components that form part of the architecture are tightly integrated. Server-side components include the ORDBMS itself, the UniGuide Rules Subsystem, a query language (SQL3), a critical component in the development of UniGuide such as the Auxiliary Catalogue System, the SQL Generator composed by a library of modules that generate SQL3 statements and page-based Web DataBlade applications. Client-side components include only Javascript modules and custom objects. Also, we explain the type of metadata being captured, parsed and stored by a customised UniGuide indexing agent. Finally, we conclude with
a description of the database search engine from an end-user perspective, indicating the type of queries end-users can perform.

### 4.2 Modelling specific domains on the WWW

#### 4.2.1 Modelling ontologies

The framework proposed in this research is predicated on two significant assumptions:

- **Ontologies or models of concepts and their relationships** (Mahalingam and Huhns 1997) represent powerful means to structure the global information base on the Web.

- **The range and diversity of data on the Web** is so extensive that ontologies may have to be constructed separately for each relatively well defined domain.

Ontology is a term with a long pedigree in philosophy. It refers to things that ‘exist’ (in the domain). It can be thought of as a generic description of the concepts and relationships that always exist, enabling knowledge sharing and reuse. For instance, it is reasonable to expect that the university domain will always have information regarding research entities, academic departments, courses, research outputs, and so on. Furthermore, these are likely to be inter-related in similar and predictable ways. As well, ontologies can grow and shrink based on the context in which they are used (Mahalingam and Huhns 1997).

Our proposed method involves isolating a distinct domain, modelling its ontology using an object-relational data model (ORDM), and extracting and storing all relevant metadata from the domain Web pages in database tables corresponding to the objects in the model. This database becomes a resource that can be queried by end-users for a
A wide range of information specific to the domain in a fashion that current search engines cannot match. Populating the database can be automated using suitable indexing agents.

*UniGuide* is a demonstration prototype implementation of the above approach. The ontology for all Australian Universities Web pages is modelled as an ORDM. This model is then mapped to ILLUSTRA database tables and a set of queries that can be issued against this database.

In the next subsections we compare different modelling paradigms in order to justify our selection, a hybrid object-relational model.

### 4.2.2 Relational Model

The relational model represents data in the form of tables or relations. A relation is a two-dimensional table of data, one dimension consists of named columns and the other unnamed rows. Not all two-dimensional tables of data are relational. A relation must satisfy the following properties:

- **Atomic values**: every intersection row/column must hold atomic values or single-valued entries
- **Homogeneous column domains**: entries in columns are from the same domain/type
- **Row Identifiers**: each row is identified uniquely, duplicate rows are rejected

The relational model is strongly based on data integrity constraints. There are constraints that control the uniqueness of rows or attribute instances (uniqueness constraints), that check the validity of values associated to attributes (domain or check value constraints) and that are concerned with the relationships between entity types.
(referential integrity constraints). Referential integrity constraints are specified with the aid of foreign keys. A foreign key is an attribute that appears as a non-key attribute in one relation and as a primary key attribute or part of, in another relation.

The relational model is suited to represent a large number of knowledge domains. On the other hand, the relational model struggles to represent efficiently superclass/subclass relationships, inheritance and hierarchical structures which are naturally supported by object-oriented (OO) environments. Furthermore, those relations that are inter-related with a large number of other relations must have as many foreign keys as logically inter-connected relations. Taking into account the nature of the kind of information stored on the Web, we can adopt two types of strategies respect to integrity constraints:

- **A purist approach:** information on the Web may be represented ambiguously, concepts may be represented with the use of synonyms, some attributes are represented as numerals and as strings as well and so on. A purist approach respect to integrity constraints will reject automatically a majority of transactions. The database will contain a very small subset of accurate information.

- **A liberal approach:** instead of defining declarative integrity constraints we may define more permissive integrity constraints with the use of rules or triggers or more drastically reduce at a maximum the number of integrity constraints of a given relation. A liberal approach will accept a majority of transactions, trading in accuracy.

### 4.2.3 Object Oriented Model

Object oriented databases (OODBs) give a natural support for applications that use complex data structures (Maciaszek 1997). The object oriented data model (OODM)
has been formed from the merging of data modelling and object oriented programming languages (OOPL). The major concepts of the OODM include objects, classes, inheritance, encapsulation, polymorphism, object identifiers and domains. Objects are structures that encapsulate attributes and methods that operate on those objects. Objects are abstractions of real-world entities that exhibit state and behaviour. The state of an object is expressed in the values of the attributes of the object. The behaviour of an object is represented by a set of methods that operate on its attributes. Classes group logically related objects that have the same attributes and methods. The concept is similar to entity type or relation in the relational model with the added functionality of possessing behaviour (methods). A key feature present in the OODM is inheritance. Inheritance permits the organisation of classes into hierarchies. A subclass inherits the definitions of attributes and operations from its parent classes (superclasses), those higher up in the hierarchy.

A unique feature of the OODM is that attribute domains may include simple data types or abstract data types (ADTs). Further, the values of an attribute may be atomic, or multi-valued, violating one the principles involved in the definition of a relation in the relational model.

Compared to the relational model where rows are identified by the values stored in the attributes that are part of the primary key, objects are identified by object identifiers (OIDs). An OID is a system generated, globally unique logical identifier, completely independent of the object state or object address (Maciaszek 1997). This feature can be considered as the most significantly different from the relational model.
Additionally, one of the key features present in the OODM paradigm, is the natural representation of aggregation. Aggregation is a containment association stating that a superset class of objects contains one or more subset classes of objects. Aggregation is a natural way of modelling complex applications. These applications impose an articulated structure and semantics on object compositions. A subset object is normally dependent for its existence on its superset object.

An OODM is well suited for representing information available on the Web, to adapt to the kind of constructs present in such environment. Some relations are nested, some attributes are multi-valued, hierarchical structures and aggregation are almost present.

Again two different approaches can be outlined in this kind of paradigm, respect to the modelling of hierarchical structures:

- **A purist approach:** model all the entire set of classes and subclasses present in the logical domain being modelled. The model becomes rich and extensive but at the same time complex.

- **A liberal approach:** model only certain parts of the hierarchy, subclass up-to a certain degree, avoiding the formation of various levels in the hierarchy. The model becomes more compact, trading in detail level.

**4.2.4 Hybrid model: object-relational model**

A hybrid object oriented model is capable of integrating both worlds, the relational model and the OODM. One can come up with the conclusion that things become even more difficult in this kind of paradigm. Which are the kinds of features should we use
from the large set of available options? Definitely that is the issue, when to use OO constructs and when to use relational ones.

Object-relational models provide mainly support for (Stonebraker and Moore 1996)

- The creation of ADTs
- The creation of complex objects
- Inheritance
- A production rule system
- object identifiers (OID)
- records of objects
- sets of objects
- references to objects

An object-oriented model can be even more suited to represent the kind of information available on the Web. In order to obtain a rich but compact model we may adopt the liberal approach respect to subclassing in class hierarchies, proposed in the OODM subsection. In addition, if our goal is to collect a respectable portion of the logical domain being harvested we must follow the liberal approach respect to integrity constraints proposed in the relational model subsection.

4.2.5 A comparison of different modelling approaches

The ontology of the particular domain on the Web could be represented as a pure relational model but the complexity of objects in the domain and the hierarchical structure of the university domain suggests the need for object-oriented approaches. Hierarchical structures can be represented in a more natural way in an object-oriented paradigm. However, a pure object oriented model is not the most suitable one if we want to keep the model compact, parsimonious, and easy to understand, without having to depict all subclasses or subtypes or a given object class. A hybrid, third-generation, object-relational data model meets most of the requirements incorporating aspects of
relations (tables) and complex objects while also mapping directly to an object-relational database system such as ILLUSTRATM.

### 4.3 Object-Relational Model of UniGuide

The object-relational model presented in this section is entirely based on the information available on Web servers that pertain to the University logical domain. As remarked in *(Abiteboul 1997)* the notion of schema on the Web is often posterior to the existence of data. The data modelling process was constrained to a limited set of universities, including some of the most important Australian universities. Further, we can identify some kind of inductive behaviour based on the ontology associated to the University logical domain, justifying that this limited modelled subset represents coherently the whole set of Australian Universities. In fact, the information stored in all these Web servers, referred to courses, subjects, publications, faculties, departments, libraries, staff members and so on. In other words, the intellectual contents of Web pages were similar, referring to the same concepts. Although, we must emphasise that the textual and hypertextual organisation of these Web servers commonly differed. As a result, the wrapper approach adopted in *(Rajaraman 1998)* would not be feasible for this kind of environment, as we are dealing with hundreds or probably up-to thousands of different Web pages types taking into consideration the textual and hypertextual organisation. On the other hand, as stated before metadata is format neutral, and depends entirely on the intellectual contents of a given Web page. The design of metadata associated to Web pages representing entity instances or classes present in the ORDM of *UniGuide* is presented in the subsequent subsection **4.5.5 (Meta data design: UniGuide scheme meta-tags)**.
The modelling strategy adopted in the University domain data model is based on the strategy depicted in hybrid object-oriented data models. A minimum level of class hierarchies and a set of permissive data integrity constraints are included in the model. In addition, in order to transform the model into more object-oriented, we decided to represent a respectable number of relations as aggregate constructs.

The object-relational model for *UniGuide* is shown in figure 4.1. It shows the entities/objects that have been modelled currently but the model is extensible. The objects in the model represent the Web pages of corresponding entities in the university Web sites. There is a 1:M relation (R[1:M]) between these objects and URLs. Generally, a Web page may contain many entity-instances but an entity-instance can have one and only one URL. Due to the fact that our model is a hybrid, object-relational model, we can represent traditional relational constructs such as tables and rows as well as object-oriented constructs such as object identifiers and sets. The object-relational representation of the ontology model for Australian Universities is shown in figure 4.1.

![Figure 4.1 Synthesised representation of the Object Relational Model of *UniGuide*](image-url)
An explanation of the entities in the model is provided below:

- **university class** stores general information about particular universities and always points to the main home pages or starting Web pages of particular domain instances. A university may contain many university units.

- **university_unit** is an abstract class that serves as the superclass of all the following subclasses: club/association, administrative entity, library, residential college, campus and academic entities.

- The **library** class may have a set of catalogues available on the WWW, and a set of staff members with their respective home pages or information related to them.

- The **administrative_entity** class may represent any of the following types: Centre, Department, Division, Group, Institute, Office, etc. and may contain information related to it and a link to its home page. An administrative entity may be parent of other subordinate administrative entities (i.e. Division of the Registrar contains various offices). An administrative entity may contain a set of staff members and may store information related to them with links to their respective home pages.

- The **academic_entity** class comprises the following types: faculty, department, school, college, unit, program, etc. An academic entity may contain other delegated academic entities (eg. faculty contains various departments).

- The **research entity** class may represent any of the following types: research institute, research group, research centre, etc. Research entity is not considered as a subclass of the university_unit superclass because a research entity can be associated to many universities or can be a totally independent organisational entity.

- Both the **academic_entity** and the **research_entity** classes may have a set of publications, projects, courses, course units (subjects), staff members and students.
All these sets are represented as abstract classes that are instantiated as aggregate objects that form part of the container class (academic_entity, research_entity).

Finally we can also specify relationships between entities/classes such as:

- **academic_entity_research_entity**: an academic entity can have/collaborate with many research entities.
- **research_entity_university**: a research entity may belong-to/collaborate-with many universities.

A more detailed object-oriented model (OOM) under the Unified Modelling language (UML) notation is presented in figure 4.2. The model includes the encapsulated rules associated with each class with the exception of the classes that are abstract. The model contradicts a purist OO modelling approach, as it is stated that attributes should be only accessible via get/set methods, which means that attributes should be declared as private. In our model, all attributes are publicly accessible and rules are declared as private due to encapsulation. We can justify this extraneous behaviour reminding that although we are using a pure OOM notation, we are dealing with a hybrid ORM. Fortunately, a hybrid object-relational environment allows us to specify queries in a structured query language such as SQL3. Adopting a purist OOM, forces us to access attributes via get/set methods, which would make difficult the process of defining queries in SQL. In order to facilitate query construction, we allow the direct querying over attributes (public access).

In the UML model, we can distinguish the following types of relationships:

- **Inheritance**: superclass/subclass, IS-A relationship (i.e. university_unit - library)
- **Association**: pure relational logical interconnection between classes, USES-A relationship (i.e. university - research_entity)
- **Aggregation**: relationship between a high level object and its constituent lower level object components, IS-COMPOSED-OF (i.e. library - staff)

Abstract classes, represented with dotted lines, are instantiated as component objects of higher level objects, and not as truly autonomous and independent objects. A higher level object may have a set of component objects that are instantiated when constructing the set. Association may be feasibly represented with the use of pure relational constructs or object references. Aggregation may be coherently represented with the use of sets that are implicitly dependent on the constituent object. As mentioned previously, we advocate the use of aggregation with the aim of obtaining a more object-oriented model.

### 4.4 Transformation of the model into an object-relational database

Data modelling involves the design of a conceptual model, the conversion of the conceptual model into a logical one and the final conversion of the logical model into an internal or system dependent model (see fig. 4.3).

![Figure 4.3 Models obtained in a structured modelling process](image)
The target system at issue, is the ILLUSTRATE ORDBMS. ILLUSTRATE permits the
definition of relational and OO constructs via SQL3 data definition statements (DDL).
We shall present now a small introduction to SQL3 in order to explain the type of
constructs required in the transformation of the ORDM into an ORDB. SQL3 is
presented in detail in the following sub-section 4.6.2.2.

In ILLUSTRATE every relational table is based on the definition of an ADT. Types
represent classes of objects or abstract data types (ADTs). The general syntax of a
create type statement is:

```
CREATE TYPE type_name
{(attribute_name type_name [virtual] | LIKE table_name)}
[UNDER type_name [,type_name…] ]
```

Create type statements permit us to create abstract classes, aggregation (sets of an object
type), references to an object type and to define inheritance of types (UNDER
type_name clause). An example of a type definition representing an abstract class, is
presented below:

```
CREATE TYPE course_t
(course_url text, course_name text, course_spec text, course_type text, course_degree_type text, course_semesters int1, course_credits smallint, course_desc text, course_last_modified timestamp);
```

This abstract class represents courses that may be instantiated as sets of courses of a
given academic entity or research entity.

The statement that actually allows the saving of object instantiations is the create table
statement. A compact syntax of the create table statement is depicted here:
CREATE TABLE table_name
[OF NEW TYPE type_name] [OF TYPE type_name]
{column_name type_name [defaults] [check_clause] [references_spec]
[NOT NULL] [(UNIQUE | PRIMARY KEY)])
[UNDER table_name]

The defaults clause is used to associate default values to a given attribute. With the check_clause we may define declarative domain integrity constraints. The references_spec clause permits the definition of referential integrity constraints. The following example creates the academic_entity table:

CREATE TABLE academic_entity OF NEW TYPE academic_entity_t
(_type text not null,
parent ref(academic_entity_t),
parent_type text,
parent_name text,
parent_url text,
last_modified timestamp default current_timestamp not null,
sources setof(course_t),
scourse_units setof(course_unit_t),
spublications setof(publication_t),
sprojects setof(project_t),
staff setof(staff_t),
students setof(student_t),
PRIMARY KEY (uni_id, _type, name) UNDER university_unit;
objects are physically related via OIDs. A set instance may be related to a parent object or row instance storing the OID of the parent object. Another interesting feature is the inclusion of the UNDER table_name clause. The table university_unit was declared in SQL3 as:

```
CREATE TABLE university_unit OF NEW TYPE university_unit_t
(url text not null,
uni_id text not null,
uni_url text,
name text not null,
address text,
city text,
state text,
phone text,
fax text,
e_mail text);
```

More explicitly, the academic_entity table inherits all the attributes associated to the university_unit table.

The general strategy followed in the transformation of the ORM into object-relational constructs can be synthesised in the following list:

- **ADT definition**: declare abstract classes via create type statements and define types associated to tables, which allows references to objects of a given table type
- **Limited declarative constraints**: generally, we have only specified declaratively, not null constraints
- **Rules-based check/domain constraints**: use of rules-based check constraints (i.e. constraints associated to the domain of values permitted for a given attribute)
- **Rules-based referential integrity**: use of rules-based referential integrity, aiming to define more tolerant referential integrity such as case insensitive referential integrity
- **Aggregation constructs**: use of sets to represent aggregation
- **References to objects**: use of object references to simulate transitive closure
- **Inheritance**: the specification of minimum level hierarchies
As we mentioned before, a typical characteristic of the information stored on the Web is that the notion of schema may be determined a-posteriori. As a result, the discovery of the schema associated to WWW logical domains is inherently dynamic and evolutionary. SQL DDL statements are static, that is we can only alter a very limited set of features, forcing us to drop and redefine tables. In order to adapt to the Web environment, we definitely advocate the translation of declarative integrity constraints into rules-based integrity constraints. Rules-based integrity constraints may be dropped and redefined without affecting table definitions. This strategy is more suitable for the Web environment. Rules-based integrity constraints are described thoroughly in the *UniGuide Rules Subsystem subsection (4.5.2.4)*.

4.5 Architecture

4.5.1 High-level overview of components

A World Wide Web search engine is defined as a retrieval service, consisting basically of a harvester (*UniGuide indexing agent*), a database, search software and a user interface available via WWW (*Poulter 1997*). *UniGuide* has similar components with some subtle differences. The database is an object-relational hybrid, with the capability to handle sets, arrays, abstract data types (ADTs), object identifiers, references to objects, relations, user defined and dynamically loadable executable C user-defined functions, inheritance, rules, etc. (*Stonebraker and Moore 1996*). The deployment of the *UniGuide* system is based on the ILLUSTRA ORDBMS®, which is an extended relational database system with the following key features:

- ability to define abstract and constructed data types
- ability to define functions and operations on abstract and constructed data types
• an associative query language (SQL3)
• a Rules System that facilitates event-driven programming using a uniform syntax (Stonebraker and Moore 1996)

The search software is implemented in a client-server environment and is based on SQL3 queries which can call to dynamically loadable external user-defined C functions (ILLUSTRA API™) with the ability to run other SQL queries as well ("callback" feature), and rules (Enguix et al. 1998b). The system currently comprises a respectable number of encapsulated rules used intensively in order to support rules-based check constraints, rules-based case-insensitive referential integrity, uniqueness of sets, automatic actualisation of object references, timestamp values and hypertext links. For security reasons, SQL3 DML and SELECT statements are actually generated on the server side by the UniGuide SQL3 Generator. The generation of the interface and the input/output of data are done on the client-side. The interface is generated dynamically in HTML by ILLUSTRA Web Datablade® applications and Javascripts, enabling any Javascript® compatible browser to access UniGuide. The functionality provided by Javascripts consists mainly in verifying the validity and reprocessing of data before sending it to the server or in the reformatting and restructuring of data received from the server side into an appropriate format for the browser. Figure 4.4 outlines the main components of the UniGuide prototype
The breadth-first multi-threaded UniGuide indexing agent has been developed in the JAVA™ programming language. The UniGuide indexing agent scans all the Web...
pages in the Australian Universities logical domain, extracting the structured meta-tags stored within the contents of target Web pages. Figure 4.5 shows diagrammatically how metadata is extracted into the UniGuide database from different Web sites (such as University of Sydney, Monash University, etc.).

![Diagram of metadata extraction](image)

**Figure 4.5** Harvest of metadata (meta-tags) in UniGuide

From our viewpoint Web pages are containers of text that hold semi-structured data (HTML tags and text) and structured data (meta-tags). The core information of a given Web page from UniGuide perspective is not the Web page itself but the metadata stored within the contents of the Web page. The UniGuide indexing agent localises the required meta-tags and extracts their contents; it ignores the rest of the text and continues navigating in a breadth-first manner through links to other Web pages.

The complexity involved in the implementation of the UniGuide system derives from the number of different component subsystems involved and the requirement of a tight integration between these subsystems to form a database search engine. From this perspective it is possible to identify different types of subsystems:
• **Base subsystems**: these subsystems provide infrastructure for the whole system. Includes the *UniGuide Auxiliary Catalogue System*, the *UniGuide Rules Subsystem*, the ILLUSTRA ORDBMS, a more advanced structured query language such as SQL3, the ILLUSTRA API and the ILLUSTRA Web Datablade.

• **Interface subsystem**: provides the interface between the WWW and the base subsystems. Includes ILLUSTRA Web Datablade applications and Javascripts.

• **Logical subsystem**: the *UniGuide SQL3 Generator* which consists of a library of dynamically loadable external C functions that automatically validate and generate SQL3 DML and SELECT statements. Represents a critical part in the implementation of the search and data populating software required in the schema-based database search engine. The parser is used both by the interface and the data collector subsystem.

• **Data collector subsystem**: are specialised in gathering data from the Web. Includes the JAVA-based *UniGuide indexing agent*.

### 4.5.2 Server-side Components

In the client-server architecture, the application processing is divided between the client and the server. In this case the server-side is responsible for database recovery, security, and concurrent access management. The server-side accepts requests from client applications, check the syntax of these requests and responds with data or error messages to the client or alters the state of the database. The following list describes the main components of the server-side in the *UniGuide* system:

• **ILLUSTRA ORDBMS**: is the core component of the system as it coordinates with, integrates and interacts with the rest of subsystems. It is responsible for all database management processes.
• **ILLUSTRA SQL3:** an advanced structured query language with the capability of allowing the representation of data that do not fit well into the traditional tabular format of relations and procedural capabilities to make SQL computationally complete as any programming language (*Catell 1994*)

• **The UniGuide Auxiliary Catalogue System:** this component is critical in the development of reusable external C functions that automatically generate SQL DML and SELECT statements from data captured from forms-based manual input or from the automatic retrieval of *UniGuide* scheme meta tags (*UniGuide indexing agent*)

• **The UniGuide Rules Subsystem:** an event-driven generic rules subsystem capable of handling traditional integrity constraints, referential integrity, etc.

• **The UniGuide SQL3 Generator:** a library of dynamically loadable executable modules capable of generating SQL3 statements "on the fly" which are executed a posteriori.

• **ILLUSTRA Web-Datablade Applications:** the Web DataBlade module enables the creation of Web applications that retrieve data dynamically from ILLUSTRA databases. Implements the interface between Web browsers and the ILLUSTRA ORDBMS.

### 4.5.2.1 ILLUSTRA ORDBMS

The ILLUSTRA ORDBMS consists of a series of specialised programs that control the access to data, based-on a client-server architecture. End-users communicate with ILLUSTRA ORDBs by running a server process, therefore for each active client there is one server process (*ILLUSTRA 96b*). The following figure (fig. 4.6) shows the architecture components involved in a client-server request. Client programs connect via TCP/IP to the midaemon process running on the ILLUSTRA server machine (1).
The midaemon listens for connection requests from clients on TCP ports. When daemon detects a client connection request, an ILLUSTRA server process is generated to deal with the connection (2). At this point client and server establish connections with each other (3). The miserver process implements the ILLUSTRA database engine, which is the process that runs all client queries and commands and returns results to active client processes.

![Figure 4.6 ILLUSTRA Components involved in a client-server request](image)

The ORDBMS as mentioned previously allows the definition of custom base types, complex objects, inheritance of types and tables. In addition, an event-driven rules subsystem is tightly integrated with ORDBs. Respect to complex objects, the ORDMS allows the declaration of sets, records or references to objects. One can view the ORDBMS as an aggregation or association of different subsystems or components. Therefore, these are architectural issues, which are out of the scope of this thesis, but we shall mention that mainly we can identify the ORDBMS as a virtual artefact containing the following subsystems or components:
• Database engine (miserver process)
• Query optimiser
• Query parser
• Complex structured query language SQL3
• Application programming language (API)
• Rules subsystem
• Daemon that manages client requests (midaemon),
• System catalogue that stores metadata and so on

From our viewpoint the main issue is the type of facilities that provide this kind of ORDBMS and the type of information we are able to define and store. In the following subsections we indicate the generic type of information stored in any table or set instance. The two types of constructs being used in the construction of the database search engine are complex tables and set types. From a modelling perspective, this approach represents an interesting attempt in dealing with non-relational constructs.

From a performance viewpoint due to fact that the ILLUSTR A query optimizer ignores indexes defined on non-OID set tables attributes, the retrieval of set instances is definitely slower than the retrieval of row instances when requesting information on non-OID attributes.

### 4.5.2.1.1 Prototype table attributes

Every row instance in the ORDB has an associated OID. In addition, it may store references to objects and standard built-in data values. A prototype row in the model will contain the following attributes (Enguix et al. 1998b):

- **OID**: Object Identifier (physical identifier of an object/entity instance)
• **URL**: Hypertext Link to resource described, generally homepage

• **Components Primary Key**: logical identifiers of an entity instance

• **Other Functional Dependent Attributes**: other descriptive information

• **Last_Modified**: Timestamp attribute in order to guide the *UniGuide* indexing agent if a particular entity instance should be updated or not.

With this generic row instance type we are able to identify row instances "physically" by the use of OIDs and logically by using primary key values.

### 4.5.2.1.2 Prototype set attributes

Set instances are always associated to parent row instances with the use of OIDs. A set instance is related to a parent row instance when its parent OID attribute (poid) matches the OID of a parent row instance. In fact, a set table row instance stores two OID attributes, the OID of the set table row instance itself and the OID of the parent table that it is associated (poid). A prototype set in the model will contain the following attributes (*Enguix et al. 1998b*):

- **OID**: Object Identifier (physical identifier of a set instance)

- **URL**: Hypertext Link to resource described, generally homepage

- **Components Pseudo Primary Key**: logical identifiers of a set instance (uniqueness constraint controlled by rules). As we are dealing with unordered multi-sets that do not require uniqueness, we must provide uniqueness of sets with the aid of auxiliary tables and the use of chaining rules. Therefore attributes that must be unique in multi-sets are called as pseudo primary keys.

- **Other Functional Dependent Attributes**: other descriptive information
• **Last_Modified**: Timestamp attribute in order to guide the *UniGuide indexing agent* if a particular entity instance should be updated or not.

• **poid**: Object identifier of the parent row instance that holds the set (i.e. Row Faculty of Commerce may have a set of staff members)

To obtain more information about the construction and definition of pseudo primary keys please refer to the subsection 4.5.2.4.6 (*Insert/Update Valid set instance and Synchronised insert/update/delete Pseudo Primary keys Rules*)

### 4.5.2.2 ILLUстра SQL3

ILLUSTRA SQL can be thought of as a powerful extension of SQL92 and a SQL3 compliant structured query language. Some of the key features contributed by ILLU storia SQL are listed below:

• **Definition of ADTs**: abstract data types can be created and associated a-posteriori to tables, sets or references to objects

• **Inheritance of types and tables**: permits the reuse and sharing of types and tables definitions and the construction of hierarchical structures

• **Definition of object-oriented constructs**: OIDs, sets, arrays and references to objects

• **Definition of event-driven rules**: encapsulated rules fired by events under certain circumstances associated to tables

• **Incorporation of procedural extensions**: SQL statements with the ability to call to external C user-defined functions, augmenting the structured query language with procedural capabilities.
• **Dynamic data conversion**: data types can be converted on the fly with cast functions or operators

DDL and DML statements have been explained in previous subsections (4.4 Transformation of the model into an object-relational database; 4.5.2.3 The Auxiliary Catalogue System and 4.5.2.4 UniGuide Rules Subsystem). We shall focus on the rich syntax provided by SELECT statements in ILLUSTR.A.

4.5.2.2.1 DDL statements

Data definition language statements are needed to provide the required base infrastructure of databases. In our case the range of possible options is more complex than the ones supported by any SQL92 compliant version. In ILLUSTR.A there are a respectable number of different types of "constructor" statements, although we made intensive use of the ones that create types, tables, indexes, rules and functions.

4.5.2.2.1.1 CREATE TYPE statement

The create type statement is used to define new composite types composed of members that may be built-in (oid, integer, text, etc.), user-defined base (user-defined ADTs) or composite types. Types are reused in the definition of tables, sets and references to objects. In other words types can be considered as abstract classes or template type definitions that can be reused by other constructs.

The general syntax of a create type statement is:

```
CREATE TYPE type_name
{ (attribute_name type_name [virtual] | LIKE table_name)}
[ UNDER type_name [,type_name... ] ]
```
The UNDER clause is used to specify inheritance of types. ILLUSTR A supports multiple inheritance.

### 4.5.2.2.1.2 CREATE TABLE

Tables are the base constructs that store actual data, although evidently not all data being stored is pure tabular. With the use of create table statements, tables can inherit implicitly the members of a given type or can define new type definitions at the same time. Also, it is possible to inherit attributes and functions (i.e. rules) from ancestor tables (UNDER clause). The domain of a column member can correspond to any of the following domains:

- **Built-in base type**: oid, int1, smallint, integer, text, etc
- **User-defined base type**: new base types defined by C functions
- **Composite type**: defined by create type statements
- **Reference to an object type**: ref(type_name)
- **Array of base type**: arrayof(base_type)
- **Set of a type**: setof(type_name)

In addition, column members may be defined as virtual as data is not being actually stored. Column names correspond to functions that process and return data.

```sql
CREATE TABLE table_name
[OF NEW TYPE type_name] [OF TYPE type_name]
{column_name type_name [defaults] [check_clause] [references_spec] [NOT NULL] [(UNIQUE | PRIMARY KEY)]}
[UNDER table_name]
```

The defaults clause is used to associate default values to attributes. The check_clause defines declarative integrity constraints. The references_spec clause permits the definition of referential integrity constraints.
4.5.2.2.1.3 CREATE INDEX

The ILLUSTR A ORDBMS supports the creation of indexes with the use of create index statements. ILLUSTR A supports functional indexes and the specification of different access methods. The default access mode makes use of generic btrees, that is, ILLUSTR A can build b-trees on any data type (Stonebraker and Moore 1996). Additional access methods such as rtree for spatial data and dtree for document data are available, although up-to-date, we have defined only btree indexes in UniGuide.

```
CREATE INDEX index_name
ON table_name
USING (btree | rtree | dtree) {column_name[,]}
```

4.5.2.2.1.4 CREATE RULE

Rules are encapsulated to tables. From an object-oriented perspective we can view a rule in ILLUSTR A as an event-activated method encapsulated in a table object with private access. Rules are activated by select, insert, update or delete statements. The action to be executed by a rule can be propagated just before, at or just after the precise moment the rule is being handled.

The general syntax of the create rule statement is:

```
CREATE RULE rule_name [as]
ON (SELECT | UPDATE | DELETE | INSERT ) TO table_name
WHERE search_condition
DO [INSTEAD | BEFORE | AFTER]
(sql_statement | sql_statement_list | nothing)
```

4.5.2.2.1.4.5 CREATE FUNCTION

The create function statement registers new functions in the database. Functions may be written in SQL or C. SQL functions may be useful to automate some operations, to
group SQL statements or to define macros. On the other hand C functions that interact with the ILLUSTR A API and the ILLUSTR A ORDBMS are capable of dealing with much more complex situations. The automatic generation of DML and SELECT statements in *UniGuide*, depend entirely in the functionality provided by a set of dynamically loadable executable C user-defined functions.

The general syntax of the create function statement is:

```sql
CREATE FUNCTION function_name [[{parameter[,]}]]
RETURNS type_name
AS ({sql_statement[;]} | EXTERNAL NAME 'path_to_file [(function_name)]'
LANGUAGE C )
```

Functions may receive parameters and always return a given type, which can be a base built-in type, a composite type, a set of a type, etc. C functions precise the specification of the path to the external file that stores internally the function. Function name is needed only if the name of the file is different from the name of the function.

4.5.2.2.2 DML statements

The DML statements included in ILLUSTR A are insert, update and delete statements. Again the options available and variations of the syntax of each of these statements differs totally from the SQL92 syntax. We may include select statements that assign a reference to an object in an insert statement or define nested updates in an update of a set instance or delete a row instance and its associated set instances in a single delete operation.

4.5.2.2.1 INSERT INTO table statement

Insert statements populate the database with new row or set instances. The general form of an insert statement is:
In *UniGuide* we insert three different kinds of types: base built-in types, references to objects and set instances. In order to represent the insertion of any of the different types, let's modify the syntax expressed above to represent formally the insertion of a base type, an object reference and set instances:

```
INSERT INTO the_table (the_simple_column, the_ref_column, the_set_column)
USING (index = the_table_index)
VALUES (value1,
(SELECT unique_ref(the_table) FROM the_table WHERE search_condition),
((set1_value1, set1_value2, set1_value3), (set2_value1, set2_value2, set2_value3)))
```

In this example a base type value is assigned to the simple_column, a pointer to an object of the same type is assigned to the ref_column and two set instances are assigned to the set_column. As we have demonstrated, insert statements in ILLUSSRA SQL differ quite from SQL92.

### 4.5.2.2.2 UPDATE table statement

The purpose of the update statement is to modify the data stored in table or set members. The general syntax of an update statement is:

```
UPDATE table_name [correlation_name]
USING (index = index_name)
SET {column_name = expression [,]}
WHERE search_condition
```

As in the insert statement we may update any of the three different types used in *UniGuide*: base types, references to objects or set instances. The third case requires special attention as precis of the definition of nested DML statements. A summarised syntax for inserting, updating and deleting set instances follows next:
An update of a set instance associated to a parent table, requires the localisation of the
parent row instance that holds the set. In fact, parent row instances and subordinated
row instances are tightly related with the use of OIDs.

### 4.5.2.2.3 DELETE FROM table statement

The purpose of executing delete statements is to destroy row or set instances. The
general form of a delete statement is:

```
DELETE FROM [table_name ]
[USING(index = index_name)]
[WHERE search_condition]
```

The main feature we can identify in a delete statement is the automatic deletion of
subjugated set instances of a given row instance, although this behaviour can be
simulated in SQL92 with the definition of cascade deletes in triggers.

### 4.5.2.2.3 SELECT statements

In *UniGuide*, the search software in the server-side is entirely based on the use of select
statements, C functions, and btree indexes, opposed to keyword-based search engines
which are based on complex algorithms generally developed in C or C++ and inverted
indexes. In our case, the query taxonomy is much more complex than in keyword-based
search engines due to the range of available options. We may issue queries that involve
single tables (simple queries), that obtain statistics (summarised information), that relate
different tables or sets (joins) or that retrieve hierarchies of child objects from a parent object (simulation transitive closure).

The general syntax of a SELECT statement in ILLUSTRRA is:

```sql
SELECT [unique | distinct] {constant | column_name | function_expression | aggregate_expression}
FROM {table_name | view_name | select_statement}
[USING (index = index_name)]
[WHERE search_condition]
[GROUP BY {column_name [,]}]
[HAVING search_condition]
[ORDER BY {(column_name | column_number) [(asc | desc) [,]]}
[UNION [all] select_statement]
```

In the next subsections, we shall explain more thoroughly each of the different types of queries being used in UniGuide.

### 4.5.2.2.3.1 Simple Queries

The scope of simple queries affects only to single tables or sets (internal representation of the set). These kind of queries are generated automatically by C functions executed from SQL return statements (see return statements subsection) and executed immediately after. The general form of a simple query is:

```sql
SELECT {column_name [,]}
FROM table_name
USING(index = index_name)
[WHERE search_condition]
```

### 4.5.2.2.3.2 Summarised Information

These queries can be specified with the use of template queries that can be adapted to the type of query one is interested. Currently, there is no support to generate these types of queries automatically with the user of C functions, a feature that may be incorporated
in the future. This type of query is typically based on the use of built-in aggregate functions. A formal representation of the general form of summarised information queries:

```
SELECT {(avg | count | max | min | sum) (column_name) [,]}
FROM table_name
[WHERE search_condition]
[GROUP BY {column_name [,]}]
[HAVING search_condition]
[ORDER BY {(column_name | column_number) [(asc | desc) [,]]}]
```

4.5.2.2.3.3 Joins
Traditional relational environments support typical primary key-foreign key joins and joins on arbitrary columns. In an object-relational environment such as the one UniGuide is based on, things become more complex. We may perform join operations between tables, tables and set tables and between set tables and set tables. Tables are joined by primary-key-foreign key relations, whereas tables and set tables are joined via OIDs. Set tables may be joined with arbitrary values or with the use of OIDs that relate different set instances related to a same parent object.

The general form of a join query is:

```
SELECT {table_name.column_name [,]}
FROM {table_name [alias] [,]}
USING(index = index_name)
WHERE Join_Condition [AND Search_Condition]
```

4.5.2.2.3.4 Fetching sets: an alternative
As mentioned before, ILLUSTR A forces us to retrieve sets inside the scope of a transaction. In order to retrieve set instances associated to a given row instance we must execute the following statements:

1. **Begin a transaction**: BEGIN TRANSACTION
2. **Select the set column:** Execute a SELECT statement including the set column type
i.e. SELECT the_simple_col, the_set_col FROM the_table WHERE search_condition

3. **Retrieve cursor generated:** identify the dynamically generated cursor:

i.e. interactive msql session result:

<table>
<thead>
<tr>
<th>The_simple_col</th>
<th>The_set_col</th>
</tr>
</thead>
<tbody>
<tr>
<td>The_simple_value</td>
<td>The_set_col_2_1_0</td>
</tr>
</tbody>
</table>

4. **Fetch rows associated to cursor:** execute a fetch statement

i.e. FETCH FROM the_set_col_2_1_0 all

5. **terminate transaction:** END TRANSACTION

Fortunately, sets in ILLUSTR.A are internally represented as separate tables, linked through the OID of the row containing the set. The name of the set table is constructed with the name of the parent table and the name of the column, as:

'_<tablename>_<column_name>'

By default, the members of a set table are the members of the base or composite type which is based on the set and the poid attribute which stores the OID of the parent table which is related to. As a result, we can retrieve set instances associated to row instances issuing the following type of queries:

```
SELECT {[the_parent_pk_value]}, {[the_set_col]} FROM the_parent_table, the_set_table
WHERE the_parent_table.oid = the_set_table.poid AND Parent_Table_Search_Conditions AND Set_Table_Search_Conditions
```

Although not necessary, the components of the primary key of the parent table are included in this type of queries so end-users can determine the source of the object being retrieved. As a matter of fact, we have made use of set constructs to represent
aggregation and therefore these are dependent on their parent objects. In other words, the information stored in set instances may not be considered as semantically complete. The pieces of information that gives identity and meaning to a set instance are parent table primary key values. We may issue queries to find information about computer courses in NSW. If the results returned by this type of queries only include course information and not universities and departments or faculties involved, the information retrieved is practically useless.

4.5.2.2.3.5 Simulation of Transitive Closure

ILLUSTRA supports references or pointers to composite types. References point to table row instances objects. The deref operator support a bidirectional conversion between an object and its reference.

Let us suppose the following example, which issues a query to a table whose members are a base built-in data type column and a reference to a row instance of another table:

```
SELECT the_simple_col, deref(the_ref_col).*
FROM table_name
WHERE search_condition
```

As seen intuitively in this query, a deref operator is capable of retrieving column values from a table without explicitly defining join operations. This is especially useful in the case of unary relations, where our intention is to retrieve all subordinate objects of a parent object. Transitive closure can be implemented with C functions that execute SELECT statements. In our case, there are two entities that represent assembly structures: administrative_entity and academic_entity. Empirically we noticed that hierarchies are stratified at the most up-to 4 levels. This type of queries must be specified in manual SELECT statements aided by template queries. We decided for a
practical and easy solution to the problem with the use of the deref operator. A typical
query simulating transitive closure has the form described next:

```
SELECT  { deref(deref(ref_to_parent).ref_to_parent).column_name [,,]},
        { deref(ref_to_parent).column_name [,,]},
        {column_name [,,]}  -- Level 3
FROM table_name
WHERE (deref(deref(ref_to_parent).ref_to_parent).column_name = value [,,]) OR
       (deref(ref_to_parent).column_name = value [,]) OR
       (column_name = value [,])
```

This query retrieves only three levels in the hierarchy. The attributes of the first level
retrieve the parent of the parent of the current row being retrieved. The second level
retrieves the parent object of the current row and finally the third level retrieves the
current row. The deref operator returns a null value when a given row does not have an
ancestor in a level-n reference.

### 4.5.2.2.4 RETURN statements

The return statement is mainly used for retrieving results from functions. There are
cases where we need to define functions that are not encapsulated in any ADT or
executed from rules or table dependent, that is, independent or multipurpose or generic
functions. As a matter of fact, the RETURN statement gives us the possibility to call
and pass parameters to generic C functions and receive results. In UniGuide, DML and
SELECT statements are generated dynamically by executing return statements that call
to external C functions.

The general form of a RETURN statement is:

```
RETURN function_name((parameter[,,])
```

### 4.5.2.3 The Auxiliary Catalogue System
The auxiliary catalogue system is required to generate dynamically SQL DML, select statements and intelligent HTML forms with contextual help and data integrity rules. In a first glimpse, the model appears to be simple as it holds only two tables (see fig. 4.7), although the mechanism to populate both automatically is quite elaborated or complex.

![Relational Model of the Auxiliary Catalogue System](image)

**Figure 4.7 Relational Model of the Auxiliary Catalogue System**

### 4.5.2.3.1 Table _tables

The table _tables stores information about normal tables and tables that represent internally sets in the object-relational model of *UniGuide*. The table _tables has the following columns, depicted in table 4.1:

<table>
<thead>
<tr>
<th>Columns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_source</td>
<td>Original table name from the ILLUSTRATA System Catalog</td>
</tr>
<tr>
<td>table_name</td>
<td>Intelligible name of the table</td>
</tr>
<tr>
<td>parent_table</td>
<td>If the table represents internally a set, the parent table that holds the set</td>
</tr>
<tr>
<td>set_name</td>
<td>If the table represents internally a set, the name of the set column in parent table</td>
</tr>
<tr>
<td>table_help</td>
<td>Help text associated to the table</td>
</tr>
</tbody>
</table>

**Table 4.1. Column descriptions table _tables**

The internal representation of a set consists of a table named with the following convention: “_” + name parent table + “_” + name of set in parent table.
The following example will clarify the syntax mentioned above:

i.e. The academic_entity table has the attribute sstaff, which is a setof(staff_t). The internal representation of the set is the table named as: _academic_entity_sstaff. In ILLUSTR A we are forced to retrieve sets inside the scope of a transaction. The following steps are required to retrieve sets associated to a given table:

1. Begin a transaction
2. Execute a select statement
3. Identify the dynamically generated cursor associated to the set
4. Execute a fetch statement to retrieve the required set instances
5. Terminate Transaction

For an environment such as the Web, we must present results to the end-user as quick as possible, due to the fact that the HTTP protocol is a stateless protocol, and connections are opened only during the execution of a single operation. The solution is obtained querying directly to the internal representation of the set, which requires the following steps:

1. Execute a select statement on the internal representation of the set

As a result, obviously we have saved 4 steps required in the normal retrieval of sets. The attributes stored in the table _tables are required to generate automatically DML and select statements with the aid of dynamically loadable executable user-defined C functions called from ILLUSTR A SQL3 statements. In addition, it also presents contextual help associated to the table. In the case of DML statements, as we work with the internal representation of a set, we must provide a mapping between the internal
representation and the external notation. The attributes defined in the table `_tables, parent_table and set_name provide such mapping.

With the intention of clarifying the ideas expressed above, we shall introduce the following example:

Let us suppose that we are inserting data about a given faculty course in an HTML form. The entities involved are academic_entity and a set instance of scourses associated to the academic entity. The internal representation of the set in this case is: `_academic_entity_scourses. Having the internal representation of the set we may find in the table `_tables two critical values to generate the appropriate DML statement (see table 4.2):

- The parent table that internally holds the set (academic_entity)
- The name of set that is part of the parent table (scourses)

<table>
<thead>
<tr>
<th>Table_source</th>
<th>table_name</th>
<th>Parent_table</th>
<th>set_name</th>
<th>table_help</th>
</tr>
</thead>
<tbody>
<tr>
<td>_academic_entity_scourses</td>
<td>Academic Entity Course</td>
<td>Academic_entity</td>
<td>scourses</td>
<td>Academic Entity Course: course/degree of a given faculty, department, college, etc.</td>
</tr>
</tbody>
</table>

**Table 4.2:** An example of a given row instance from the table `_tables

The general syntax in ILLUSTRAX SQL3 for inserting set instances, which requires some kind of nested SQL statements, is the one denoted below:

```
UPDATE parent_table
SET(INSERT INTO
parent_table.set_name({set_attribute[,]})
VALUES({set_values[,]})
WHERE {PK_attribute_parent_table = PK_value [AND]})
```

Substituting the required values in the nested SQL statement described above we obtain:
UPDATE academic_entity
SET(INSERT INTO academic_entity.scourses(course_url, course_name,....)
VALUES('http://www.the_uni.edu.au/the_faculty/the_course.html','Bachelor in...
WHERE upper(uni_id) = upper('The University') AND
upper(_type) = upper('Faculty') AND upper(name) = upper('The_Faculty');

As we have demonstrated the table _tables from the auxiliary catalogue system is
critical to generate dynamically DML statements.

4.5.2.3.2 Table _columns

The attributes associated with the table _columns are required to generate automatically:

- **Intelligent HTML forms** with data integrity rules and contextual help
- **DML and SELECT statements** created by executing dynamically loadable
  executable user-defined C functions called from ILLUSTR A SQL3 statements (as in
  the table _tables)

The table _columns has the following attributes (table 4.3):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table_source</td>
<td>Name of the original table/set that the column belongs to</td>
</tr>
<tr>
<td>Column_source</td>
<td>Original name of the column in the ILLUSTR A System Catalog</td>
</tr>
<tr>
<td>Column_name</td>
<td>Intelligible name of the column</td>
</tr>
<tr>
<td>Column_type</td>
<td>Type of column:</td>
</tr>
<tr>
<td></td>
<td>• sB (search both Insert/Update &amp; Query Form),</td>
</tr>
<tr>
<td></td>
<td>• iB (insert both Insert/Update &amp; Query Form),</td>
</tr>
<tr>
<td></td>
<td>• iQ (insert Query Form)</td>
</tr>
<tr>
<td>Insert_column_number</td>
<td>Column number Insert/Update Form section</td>
</tr>
<tr>
<td>Sqquery_column_number</td>
<td>Column number simple Query Form section</td>
</tr>
<tr>
<td>Column_data_type</td>
<td>Original data type ILLUSTR A system catalogue: text, int1, oid, etc.</td>
</tr>
<tr>
<td>Insert_object_type</td>
<td>HTML form object type for Insert/Update Forms</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Squery_object_type</td>
<td>HTML form object type for simple Query Forms</td>
</tr>
<tr>
<td>Column_help</td>
<td>Help text associated to the column</td>
</tr>
<tr>
<td>Mandatory</td>
<td>Indicates if column is nullable or not</td>
</tr>
<tr>
<td>Column_pk</td>
<td>Indicates if column is component of a primary key or pseudo primary key.</td>
</tr>
</tbody>
</table>

**Table 4.3:** Column descriptions table _columns

Some concepts and notations must be introduced with the aim of explaining more coherently the purpose of some of the attributes described above. In the *UniGuide* interface to the WWW, we consider two types of HTML forms:

- **Insert/Update Forms:** forms used to insert/update row and set instances
- **Query Forms:** forms used to retrieve row and set instances

Additionally, we consider two different types of attributes related to the kind of construct it is associated:

- **Search columns:** columns that are component of the primary key of parent tables that hold sets and although not being part of the actual data being stored or retrieved, are necessary to insert/update or retrieve set instances that are part of row instances
- **Insert columns:** covers attributes associated to normal tables and sets

Again, an intuitive way of explaining the concept described above may be obtained recalling the example depicted in the _tables_ subsection (fig. 4.8):

```
UPDATE academic_entity
SET course_url = 'http://www.the_uni.edu.au/the_faculty/the_course.html',
    course_name = 'Bachelor in...',
WHERE upper(un_i_d) = upper('The University') AND
    upper(type) = upper('Faculty') AND upper(name) = upper('The_Faculty'),
```
Figure 4.8 Identification of column types in a DML statement involving a set instance

Expressed formally as:

```
UPDATE parent_table
SET(INSERT INTO
parent_table.set_name({set_attribute[,]})
VALUES({set_values[,]})
WHERE {PK_attribute_parent_table = PK_value [AND]})
```

In this case, set attributes (set_attribute columns) are considered as insert type columns, whereas the components of the primary key of the parent table that holds the set (PK_attribute_parent_table columns) are considered as search columns, necessary to insert the set instance. As a result, the columns associated to the internal representation of a set in the auxiliary catalogue system will include first the column components of the primary key of the parent table that holds the set, which must be considered as external attributes and the actual set attributes. In fact this strategy permits us to generate dynamically Insert/Update and Query Forms associated to row and set instances with simple SELECT statements.

Column types are obtained combining two different classifications:

- **By type of form**: a column may appear in an Insert/Update Form, in a Query Form or in both Insert/Update and Query Form
- **By type of construct**: columns are either Search or Insert columns

Table 4.4 summarises the possible combinations obtained by taking into consideration both classifications expressed above:
The majority of attributes are classified as iB or sB, which means that insert and search columns appear in both Insert/Update and Query forms. The attributes classified as iQ are attributes that are inserted/updated automatically by ILLUSTR A with the use of rules. An example of such attribute type are hypertext links to URLs associated to row instances referenced by foreign key values (i.e. uni_url attribute in academic_entity contains a URL referencing the homepage of the University that is part of). As a matter of fact, this classification provides an effective mechanism to filter columns on an HTML form depending of the type of form, to generate forms that can treat sets or normal tables indistinguishably, hiding the complexity to the end-user and to generate dynamically DML and SELECT statements.

The columns insert_column_number and squery_column_number are required to display the columns in a predefined order in Insert/Update and Query forms respectively. The attribute column_data_type is needed to reformat values into a valid SQL format (i.e. text attributes are enclosed in inverted commas: ‘value’). The columns insert_object_type and squery_object_type store HTML tags that generate form objects (i.e. textbox, textarea, etc.) in Insert/Update and Query forms respectively. Each object type contains a pair of functions associated to a given event in order to display contextual help in the status bar of the browser. Other modelled attributes include mandatory, which defines if the input of a given column is required. This column conditions if a given form is submitted or not when the user intends to save the entered data. And finally, the column column_pk indicates if a column is a component of a

<table>
<thead>
<tr>
<th>Insert Column</th>
<th>N.A.</th>
<th>iQ</th>
<th>iB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search Column</td>
<td>N.A.</td>
<td>N.A.</td>
<td>sB</td>
</tr>
</tbody>
</table>

Table 4.4 Column_type values. Note: N.A. stands for Not Applicable
primary key of a relational table or a component of the pseudo primary key in a set instance. The main purpose of modelling this column is to generate automatically WHERE clauses in DML and SELECT statements.

Table 4.5 shows an example of a row instance from the _columns table, storing metadata about the attribute course_url modelled in the object-relational model of UniGuide. The interpretation of the values is included.

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_source</td>
<td>_academic_entity_scourses</td>
<td>Table represent internally a set</td>
</tr>
<tr>
<td>column_source</td>
<td>Course_url</td>
<td>Column name ILLUSTRAT system catalogue</td>
</tr>
<tr>
<td>column_name</td>
<td>Course URL</td>
<td>Intelligible name to appear on an HTML form</td>
</tr>
<tr>
<td>column_type</td>
<td>IB</td>
<td>Insert type and appears in both Insert/Update and Query Forms</td>
</tr>
<tr>
<td>insert_column_number</td>
<td>4</td>
<td>Order to appear dynamically on a Insert/Update Form</td>
</tr>
<tr>
<td>squery_column_number</td>
<td>4</td>
<td>Order to appear dynamically on a Query Form</td>
</tr>
<tr>
<td>column_data_type</td>
<td>Text</td>
<td>ILLUSTRAT data type</td>
</tr>
<tr>
<td>insert_object_type</td>
<td>&lt;input type=text name=v_object size=47 maxlength=80 onblur=&quot;LF_clearstatus()&quot; onfocus=&quot;LF_showstatus(4)&quot; &gt;</td>
<td>HTML tag that generates dynamically a form input textbox object with contextual help in a Insert/Update Form</td>
</tr>
<tr>
<td>squery_object_type</td>
<td>&lt;input type=text name=v_object value=&quot;*&quot; size=47 maxlength=80 onblur=&quot;LF_clearstatus()&quot; onfocus=&quot;LF_showstatus(4)&quot; &gt;</td>
<td>HTML tag that generates dynamically a form input textbox object with contextual help in a Query Form</td>
</tr>
<tr>
<td>column_help</td>
<td>Course Url: include protocol: i.e. http:// OR ftp://. URL is case sensitive</td>
<td>Contextual help associated to the column</td>
</tr>
<tr>
<td>Mandatory</td>
<td>1</td>
<td>Value indicates is mandatory for an Insert/Update Form</td>
</tr>
<tr>
<td>column_pk</td>
<td>1</td>
<td>Value indicates is a component of the pseudo primary key</td>
</tr>
</tbody>
</table>

Table 4.5 An example of a given row of data from the _columns table

4.5.2.3.3 Conclusions Auxiliary Catalogue System
The *UniGuide Auxiliary Catalogue System* is critical in the sense that the *UniGuide SQL3 Generator* precises of this component in order to generate automatically DML and SELECT statements treating row and set instances transparently hiding the complexity to end-users. Additionally, it permit us to create dynamically intelligent HTML forms with contextual help controlled by events. This saves us substantially the maintenance of the interface to the search engine, as there is no need to create a given form for each particular entity defined in the model, therefore the model can expand smoothly and progressively without much problems. Almost 60% of the information stored in these tables is generated automatically by a batch of complex SQL queries, that extract information from the ILLUSTRAR System Catalogue. The rest of data (40%) must be inserted manually. In spite of the fact, that we mentioned DML and SELECT statements are generated by the *UniGuide SQL3 Generator* interactively from form-based interfaces, the metadata stored in the *UniGuide Auxiliary Catalogue System* is also critical for generating SQL3 DML statements from data gathered automatically by the *UniGuide indexing agent*.

### 4.5.2.4 UniGuide Rules Subsystem

The *UniGuide* rules subsystem is based entirely on ILLUSTRAR’s generic event-oriented rules subsystem. When rules are created, the rule handler informs the ORDBMS to perform an action either instead or in addition when a certain event occurs involving a certain object under certain conditions. The general form of a rule is:

```
ON EVENT
DO ACTION
```
Rules systems must include the capability of executing actions just before or just after the event is processed and typically default just after execution. (Stonebraker and Moore 1996). Rules are created using the create rule statement. The general syntax of the create rule statement is:

```
CREATE RULE rule_name [as]
ON (SELECT | UPDATE | DELETE | INSERT ) TO table_name
WHERE search_condition
DO [INSTEAD | BEFORE | AFTER]
(sql_statement | sql_statement_list | nothing)
```

The create rule statement interprets the keyword current and new used as correlation names depending on the type of action specified in the ON clause (table 4.6):

<table>
<thead>
<tr>
<th>Action</th>
<th>Current</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE</td>
<td>Refers to the row before the update</td>
<td>Refers to the row after update</td>
</tr>
<tr>
<td>SELECT/DELETE</td>
<td>Refers to the row being selected or deleted</td>
<td>N.A.</td>
</tr>
<tr>
<td>INSERT</td>
<td>N.A.</td>
<td>Refers to the row being inserted</td>
</tr>
</tbody>
</table>

**Table 4.6** Interpretation of the keywords current and new in rules

Having described the general syntax of a create rule statement and the scope of the keywords new and current we shall mention the generic rule types created in the *UniGuide* system. As mentioned in a previous subsection (4.4 Transformation of the model into an object-relational database) we must provide a framework adapted to the kind of information stored in the Web. The definition of rules-based integrity constraints and rules-based referential integrity is more suitable for a loosely typed and dynamic Web environment. Rules-based integrity constraints may be redefined at any moment in order to adapt to new situations, without affecting the static structure of a table. Probably the most original use of rules made in the *UniGuide* system is the virtual definition of primary keys in unordered multi-sets, a feature not provided originally by the ILLUSTRAT ORDBMS.
4.5.2.4.1 Rules-based Check Constraints

In SQL DDL statements, the normal procedure is to declare static check constraints associated to particular columns. As a result, when integrity constraints evolve, tables must be redefined, which implicates the dropping of the table/type and execution of a modified DDL statement. In order to overcome this obstacle, the majority of static declarative check constraints have been transformed into rules-based check constraints. Check constraints rules may be dropped and redefined continuously without affecting the table/type structure. Figure 4.9, shows an example of a rules-based check constraint associated to the table research_entity and the attribute state:

```
CREATE RULE on_insert_check_research_entity_constraints
ON INSERT TO research_entity
WHERE upper(state) NOT IN ('NSW','ACT','NT','SA','WA','QUEENSLAND','TASMANIA','VICTORIA')
DO INSTEAD
   Return ErrorRaise(2,'Invalid values (constraint violation)');
```

Figure 4.9 Example of a rules-based check constraint

An insert transaction is rejected if the state being inserted is not in the list. In this case state values are case insensitive.

4.5.2.4.2 Rules-based Referential Integrity

Data in SQL is by default case sensitive. With the aid of rules we are able to define rules-based case-insensitive referential integrity. This is in the line of our convictions, as we mentioned before a more liberal approach respect to integrity constraints is followed in order to accept a majority of transactions and represent a substantial portion of the logical domain being harvested. Although not currently implemented, we foresee as an interesting future option the implementation of "fuzzy referential integrity" with the aid
of similar pattern matching algorithms or functions such as agrep (Bowman et al. 1995).

An example of a rules-based case insensitive referential integrity constraint follows below:

```
CREATE RULE on_insert_check_research_entity_university_IRI
  ON INSERT TO research_entity_university
  WHERE NOT EXISTS
    (SELECT uni_id FROM university using(index=university_pk_idx) WHERE upper(university_pk) = upper(new.university_pk)) OR
  NOT EXISTS
    (SELECT name FROM research_entity using(index=research_entity_pk_idx) WHERE upper(name) = upper(new.name) AND url = new.re_url)
  DO INSTEAD
    RETURN ErrorRaise(2,'Referential Integrity Violation: Research Entity or University');
```

This rule pertains to a table representing a many to many relationship between research entities and universities. The rule tests case insensitive referential integrity for both related tables, research_entity and university. If any of these constraints are violated the transaction is rejected giving an error code and message.

4.5.2.4.3 Timestamp update Rules

Every table/type contains an attribute named as last_modified which stores timestamp values. Timestamp values are updated automatically by SQL3 DML statements generated by the UniGuide SQL3 Generator. This occurs for the majority of tables or set types, although we also provide support for tables that are updated manually. Currently the only table which cannot be populated by the UniGuide indexing agent is the university table. In order to update automatically the last_modified attribute we define timestamp update rules, which are fired after updating a university row instance. For insert transactions the default value for the last_modified attribute is the current timestamp value (current_timestamp).
The following rule updates the last_modified attribute with the current timestamp value after updating a row instance:

```sql
CREATE RULE after_update_university_update_timestamp
ON UPDATE TO university
DO AFTER
UPDATE university USING(index=university_pk_idx)
SET last_modified = current_timestamp
WHERE uni_id = new.uni_id;
```

4.5.2.4 URL Foreign key update rules

As we are dealing with object-relational constructs, our model is outside the scope of a traditional relational system. Therefore normalisation rules are not strict in this kind of environment and we apply some denormalization rules in order to give fast access to some attributes. With the aid of post-event rules, the data stored in these denormalized attributes remains consistent being automatically populated by the action fired by the rule. We introduce the notion of referential hypertext links, which are attributes that store the URL of the entity referenced by a foreign key value. In fact, for each foreign key in a given table there is referential hypertext link attribute. The approach outlined above, is ideal for tables representing a many to many relationship. The rule depicted below is associated to the table academic_entity_research_entity, which represents a many to many relationship between academic entities and research entities:

```sql
CREATE RULE after_insert_ac_entity_res_entity_update_urls
ON INSERT TO academic_entity_research_entity
DO AFTER
UPDATE academic_entity_research_entity
USING(index = academic_entity_research_entity_pk_idx)
SET ac_url = (SELECT unique(url) FROM academic_entity
USING(index = academic_entity_pk_idx)
WHERE upper(uni_id) = upper(new.uni_id)
AND upper(_type) = upper(new.ac_type)
AND upper(name) = upper(new.ac_name))
WHERE uni_id = new.uni_id AND ac_type = new.ac_type AND
```
In this case the table academic_entity_research_entity contains two referential hypertext links:

- **Ac_url**: URL homepage academic entity
- **Re_url**: URL homepage research entity. Not defined in the rule because it is part of the primary key of the relation and part of the foreign key that references to research entity. This attribute must remain consistent a-priori to the execution of the rule being described.

The rule updates automatically the referential hypertext link to academic_entity after inserting a new academic_entity_research_entity row instance.

### 4.5.2.4.5 Parent Object Reference update Rules

References to objects are created in ILLUSTRa executing specific SQL3 queries and assigning a pointer to the object retrieved in the query, to attributes. As a result, references in ILLUSTRa are static, they do not regenerate dynamically as data changes. Further, in some cases it is impossible to regenerate references as the attribute values involved in the specific queries that assign pointers to objects have not been previously stored. This example concerns the definition of the is_parent relation in the tables academic_entity and administrative_entity, where academic and administrative entities may contain/be parent of many subordinate entities. Storing the attribute values associated to queries that create object references, give us the possibility of regenerating object references either with a batch of SQL queries or either procedurally with C user-
defined functions. Two main reasons influenced in the modelling of the relation is_parent as an object reference:

- **Acceptance of transactions**: in an environment such as the Web, we cannot guarantee if parent objects are retrieved and stored in the database before inserting child related objects. An example of this kind of situation is that probably the *UniGuide indexing agent* or a manual form-based input stores data about a given department before inserting the faculty which is part of. A foreign key strictly rejects a transaction if referential integrity is violated. On the other hand, the object reference approach will accept the transaction pointing to a null object. Further, as we store the attribute values associated to the query that creates the object reference, when parent objects are inserted we are able to regenerate object references so child objects can point to them (tree structure).

- **Simulation of transitive closure**: in the relational model it is very difficult to represent assembly structures. Fortunately, with the aid of object references it is more natural and easy to represent this kind of structures. Object references gives us the possibility to issue queries such as: "Give me all the hierarchical structure of a given faculty: schools within the faculty, departments within schools and so on"

Having described the purpose of including object references in the model, we justify the use of rules that update object references to parent object. An example associated to the academic_entity table follows:
CREATE RULE after_insert_academic_entity_update_refs
ON INSERT TO academic_entity
DO AFTER
UPDATE academic_entity USING(index=academic_entity_pk_idx)
SET rparent = (SELECT unique ref(academic_entity) FROM academic_entity
USING(index = academic_entity_pk_idx) WHERE upper(un_id) =
upper(new.un_id) AND upper(_type) = upper(new.parent_type) AND
upper(name) = upper(new.parent_name))
WHERE uni_id = new.un_id AND _type = new._type AND name = new.name;

This rule assigns a pointer to a parent academic entity to the attribute rparent, after
being inserted a new academic_entity row instance.

4.5.2.4.6 Insert/Update Valid set instance and Synchronised insert/update/delete

Pseudo Primary keys Rules

An ILLUSTR A set is an unordered collection of elements of the same type. The
ILLUSTR A ORDBMS does not support uniqueness on sets, so technically an
ILLUSTR A set is a multi-set. In the UniGuide ORM, we transformed sets into virtual
ordered sets that support uniqueness by defining auxiliary tables and with the use of
chaining rules. The concept of pseudo primary key is introduced to define a new kind
of mechanism in order to control uniqueness of sets. Pseudo primary keys differentiate
from primary keys in:

- **Virtual nature**: cannot be defined in primary key clauses associated directly to the
set type. This is consequence of the steps involved in creating set types. Firstly
class/types are defined and secondly attributes declared as setof(type_name)
implement sets of a particular type. As ILLUSTR A does not support uniqueness on
sets, SQL3 DDL does not include any primary key clause associated to sets.
Furthermore, if we use the internal representation of the set, the query optimizer
only recognises the indexing of poid (OID to parent object) attributes. As a result,
indexes that implement unique constraints cannot be declared in tables representing internally sets. As a matter of fact, a new kind of virtual primary key associated to sets was defined with the use of rules encapsulated in tables that represent internally sets and auxiliary tables

- **Non-autonomous implementation**: precises of auxiliary tables, named as mirror tables, which store attribute values of the components of the pseudo primary keys. Data remains consistent thanks to the use of synchronised rules.

**Figure 4.10** shows diagramatically the rules activated by an insert event associated to a set instance and the possible actions being executed:

![Diagram](image)

**Figure 4.10** Chaining rules to control uniqueness of set instances
When a new set instance is being inserted, the insert_valid_set_instance rule is fired. The rule rejects the current transaction if the set instance exists in the mirror table that holds pseudo primary key values. If the transaction was accepted, the rule synchronised_insert_mirror_table is fired. The rule captures the attribute values associated to the component of the pseudo primary key and stores them in the mirror table. Therefore the mirror table always remains in a consistent state with updated values. The following rules associated to academic entity courses materialise the behaviour described above:

```
CREATE RULE insert_valid_academic_entity_scourses
ON INSERT TO _academic_entity_scourses
WHERE new.course_url='' OR new.course_name='' OR NullValue(new.course_url) OR NullValue(new.course_name) OR EXISTS(SELECT *
FROM m_academic_entity_scourses
USING(index=m_academic_entity_scourses_pk_idx)
WHERE poid=new.poid AND upper(course_name) = upper(new.course_name) AND course_url = new.course_url)
DO INSTEAD
Return ErrorRaise(2,'Invalid values (constraint violation) or Academic Entity Course Exists (unique constraint violation');
```

This rule rejects a course set instance associated to an academic entity if the set instance exists in the mirror table m_academic_entity_scourses on being inserted to set instance.

```
CREATE RULE sync_insert_m_academic_entity_scourses
ON INSERT TO _academic_entity_scourses
DO AFTER
INSERT INTO m_academic_entity_scourses
USING(index=m_academic_entity_scourses_pk_id)
VALUES(new.poid, new.course_name, new.course_url);
```

This rule inserts the attribute values associated to the components of the pseudo primary key (poid, course_name, course_url) in the mirror table m_academic_entity_scourses, after being inserted the new set instance.
4.5.2.5 UniGuide SQL3 Generator

The **UniGuide SQL3 Generator** consists of a library of dynamically loadable executable modules based on the ILLUSTRA application programming interface (API), capable of generating dynamically SQL3 DML/Select statements from data captured from **UniGuide** client-side components. The API is used for the development of standalone applications and dynamically loadable executable functions that access data stored in ILLUSTRA databases. The API provides its own set of data types that correspond to ILLUSTRA SQL3 types. The following table *(table 4.7)* shows the equivalence between some API types, standard C types and ILLUSTRA SQL3 base types:

<table>
<thead>
<tr>
<th>API type</th>
<th>Standard C type</th>
<th>SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>mi_char1</td>
<td>Char</td>
<td>Char1</td>
</tr>
<tr>
<td>mi_smallint</td>
<td>Short</td>
<td>Smallint</td>
</tr>
<tr>
<td>mi_integer</td>
<td>Int, long</td>
<td>Integer</td>
</tr>
<tr>
<td>mi_text</td>
<td>-</td>
<td>Text</td>
</tr>
</tbody>
</table>

*Table 4.7* Correspondence between ILLUSTRA API data types and standard C and SQL3 data types

In addition, the API supplies conversion routines from text to string:

- **mi_string_to_text**: converts null-terminated strings into text objects
- **mi_text_to_string** and **mi_text_to_buffer**: convert text objects into strings

It is difficult to explain the whole set of available functions in the API. Nevertheless, we shall explain briefly a simple generic program that can execute SQL statements. It is hoped that this example will help the reader understand parts of the library of C modules developed in **UniGuide** that form the Dynamic SQL3 Parser.
/* get_column_data_type.c */

/* Simple C function that interacts with an ILLUSTRATRA database */
/* Receives as arguments a table name and a column name and */
/* Execute a select statement that retrieves the data type */
/* of the column in the auxiliary catalogue table_columns */

#include "stdio.h"
#include "mi.h" /* includes all header files used in the API */

mi_text *get_column_data_type(mi_text *strtblarg, mi_text *strcolarg)
{
    MI_CONNECTION *conn; /* represents a connection to the server */
    MI_ROW *row; /* represents a handle to a row */
    mi_char1 *strtbl; /* name of table */
    mi_char1 *strcol; /* name of column */
    mi_char1 *strsql; /* SQL statement */
    mi_char1 *strvalue; /* value captured from SQL statement */
    mi_char1 *strdatatype;
    int intlen, interr, intresult;

    /* allocate memory dynamically */
    strdatatype = mi_alloc(sizeof(mi_char1) * 50);
    strtbl = mi_alloc(sizeof(mi_char1) * 50);
    strcol = mi_alloc(sizeof(mi_char1) * 50);
    strsql = mi_alloc(sizeof(mi_char1) * 100);

    /* (1) convert text types into strings */
    mi_text_to_buffer(strtblarg, strtbl);
    mi_text_to_buffer(strcolarg, strcol);

    /* (2) open a connection the default server */
    conn = mi_open(NULL, NULL, NULL);
    strcpy(strvalue, "");

    /* initialise strvalue */
    mi_exec(conn, strsql, 0);

    /* (3) generate SQL command that retrieves the column data type */
    strcat(strsql, "SELECT column_data_type FROM _columns WHERE table_source = ");
    strcat(strsql, strtbl);
    strcat(strsql, " AND column_source = ");
    strcat(strsql, strcol);

    /* (4) Send query to server to be executed */
    while((intresult = mi_get_result(conn)) != MI_NO_MORE_RESULTS) {
        switch(intresult) {
        case MI_ROWS:
            row = mi_next_row(conn, &interr);
            mi_value(row, 0, &strvalue, &intlen);
            break;
        }
    }

    /* (7) close the connection */
    mi_close(conn);

    /* deallocate memory */
    mi_free(strtbl);
    mi_free(strcol);
    mi_free(strsql);
    strcpy(strdatatype, strvalue);

    /* (8) returns data type and converts it into text for SQL */
    return mi_string_to_text(strdatatype);
}
The general steps involved in coding C functions that execute SELECT statements against ILLUSTRA databases and are called from SQL3 RETURN statements are:

1. Convert SQL3 text types into C pointers to char (strings)
2. Open a connection to the ILLUSTRA server
3. Generate an SQL statement and store it in a string variable
4. Send the query to the server in order to be executed
5. Loop over the retrieved rows
6. Loop over the columns associated to the row, retrieving column values
7. Close the connection to the server
8. Convert string values into SQL 3 text types and return the value to calling statement

The creation of dynamically loadable executable C functions requires the following steps:

1. Write the function in C compliant with the API. Do not include on the top a main function, only the name of the function and parameters
2. Compile the function
3. Link the function as a dynamically loadable executable
4. Register the function with the create function statement

As an example, let us suppose that we have already compiled the function and transformed it into a dynamically loadable executable module. The next step consists of registering the get_column_data_type function in the ORDBMS so it can be called by a RETURN statement:

```
CREATE FUNCTION get_column_data_type(text,text)
RETURNS text AS
EXTERNAL name 'the_root_dir/the_subdir/get_column_data_type.so'
LANGUAGE C
```
Finally, the function can be called from RETURN statements:

interactive mysql session:

```
RETURN get_column_data_type('the_table','the_column');
```

<table>
<thead>
<tr>
<th>Get_column_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
</tr>
</tbody>
</table>

One row returned

Our main focus is on SQL statements that generate DML and SELECT statements. The reason to adopt this strategy is that DML and SELECT statement are generated from the data captured from HTML forms or from data gathered by the *UniGuide indexing agent*. The actual execution of a DML of SELECT statement is performed in two phases, depicted by the following diagram (fig 4.11):

![Figure 4.11 Phases in the execution of a DML/SELECT statement in UniGuide](image-url)
In phase I, RETURN statements that invoke the UniGuide SQL3 Generator are executed from browsers or activated by the *UniGuide indexing agent*. Either the SQL3 statement is generated or an error code/error message is returned by the server. Error code/error messages are generated automatically by validating rules or returned by dynamically loadable executable modules. If an error code has been produced, the error is trapped and displayed to browser users or handled by code in the *UniGuide indexing agent*. If no errors were present, this means that a successful DML/SELECT statement has been generated. Phase II receives as a text parameter the SQL DML/SELECT statement and executes the statement immediately after.

There is only one case in which we violate in some way the principles mentioned above of a two-phase execution of DML/SELECT statements. The case in question, is the insertion of set instances. When we insert set instances whose parent object has not been inserted in the database previously, we are forced to insert a-priori the parent row instance in order to associate related set instances. As a result, we have a module that generates a DML statement for set instances and commits actually a transaction at the same time. In the next subsection we present a more thorough explanation describing the motive for such anomaly.

**4.5.2.5.1 DML Generation Statement Modules**

DML generation statement modules are called from Web Datablade applications or from the *UniGuide indexing agent* developed in JAVA™ via a JDBC driver for ILLUSTRA. Before executing a RETURN statement that generates automatically a DML statement, the application must deduce the kind of construct being inserted. There is a specific module for generating insert/update statements for row instances and a
specific module for set instances. Once the type of construct to be inserted or updated in
the database is known, the application calls the required module. Parameters are
captured automatically by Web Datablade applications or by the UniGuide indexing
agent. The list described below includes the required parameters for this type of
modules:

- Name of the table or set
- A list of attributes delimited by the '^' character and
- A list of correspondent attribute values delimited by the '^' character as well.

The delimiter '^' was chosen as an alternative to the typical ',', because the character '^' is
rarely used and definitely because a comma separator has the ability to corrupt easily
SQL statements.

We shall describe in an intuitive way by using simple structured English, a prototype
algorithm for generating automatically DML statements for row instances:

```
BEGIN
    Convert list of Attributes and Values into Arrays
    Generate an Array of Primary Key components needed to compose
    the WHERE clause of a SELECT statement that tests if the Row
    instance Exists in the Database
    Create the WHERE clause of a SELECT statement that tests if the
    Row instance Exists
    Test if Row instance Exists in the database
    IF Row instance Exists THEN
        Generate a DML Update statement
    ELSE
        Generate an Insert statement
    END
    Return SQL DML statement
END
```

A general notation is used to name functions. Function names are formed by
concatenating the string "GF_" plus a descriptive string that indicates the purpose of the
function. "GF" stands for general function. The C function named as
"GF_simple_insert" generates DML Insert/Update SQL statements. The correspondent
structure chart and source code can be found in Appendix A UniGuide SQL3 Generator Library (A.1.1 DML Statement Generation Modules: Row instances).

In order to demonstrate the different types of DML statements that are generated, we include an excerpt of an interactive msql session:

**Case 1: Row exists**

1. Let us suppose that the library table contains only one row instance. A select statement is issued to display the unique row:

   ```sql
   * SELECT url, uni_id, name FROM library;
   ```

<table>
<thead>
<tr>
<th>Url</th>
<th>uni_id (PK comp.)</th>
<th>name (PK comp.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www-library.uow.edu.au/">http://www-library.uow.edu.au/</a></td>
<td>University of Wollongong</td>
<td>University of Wollongong Library</td>
</tr>
</tbody>
</table>

   one row selected

2. The URL associated to the library has changed. A return statement calling the GF_simple_insert module is executed in order to reflect changes:

   ```sql
   * RETURN GF_simple_insert('library','url^uni_id^name',
   'http://www-newlib^University of Wollongong^University of Wollongong Library');
   ```

   ```sql
   GF_simple_insert
   UPDATE library
   SET url = 'http://www-newlib', uni_id = 'University of Wollongong', name = 'University of Wollongong Library',
   last_modified = current_timestamp
   WHERE upper(name)=upper('University of Wollongong Library')
   AND
   upper(uni_id)=upper('University of Wollongong');
   ```

   one row selected

   A valid DML Update statement for row instances has been generated.

**Case 2: Row does not exist**

```sql
* RETURN GF_simple_insert('library','url^uni_id^name',
'http://www.library.usyd.edu.au/^University of Sydney^University of Sydney Library');
```

```sql
GF_simple_insert
INSERT INTO library(url,uni_id,name,last_modified)
VALUES ('http://www.library.usyd.edu.au/','University of Sydney','University of Sydney Library',
current_timestamp);
```
The module detected the non-existence of the row and a valid DML Insert statement for row instances has been generated.

To date, the most complex C module developed in the UniGuide SQL3 Generator library is the one that generates DML Insert/Update statements for set instances. The UniGuide Auxiliary Catalogue System is critical as an aid in generating this kind of statements. A mapping between the internal representation of a set and the correspondent valid SQL notation is provided. As mentioned before, the main goal in UniGuide is to accept a maximum number of reasonably accurate transactions in order to cover as many logical Web domains. Our strategy advocates the pursuit of a liberal approach respect to integrity constraints. The most critical constraint involved in an aggregation construct, is the existence dependency between sub-components and parent objects. In other words, the insertion of set instances in the database is conditioned by the previous existence of associated parent row instances. In a non-deterministic and asynchronous environment such as the Web, there is no guarantee that transactions are executed synchronously in sequence. Some transactions may refer to "orphan" set instances before having been inserted in the database, parent row instances.

In the line of our approach, we accept this kind of transactions, inserting a-priori "dumb" parent row instances with primary key and default values, and finally associate the correspondent set instances. Further, parent row instances can be updated later on, by Web Datablade applications or by the UniGuide indexing agent.

A prototype algorithm for generating automatically DML statements for set instances has the following form:
BEGIN
Convert list of Attributes and Values into Arrays
Generate an Array of Pseudo Primary Key components needed to compose the
Interior WHERE clause of a SELECT statement that tests if the Set instance Exists
in the Database
Create Interior WHERE clause of a SELECT statement that tests if the Set instance
Exists
Create Exterior WHERE clause used both by a SELECT statement that tests if the
Parent Row that holds the Set and the DML statement that assigns the Set instance
to a Parent Row
Get the Name of the Parent Table that holds the Set
Test if the Parent Table Row instance Exists
IF Parent Row does not Exist THEN
BEGIN
Insert Parent Row with Primary Key component values and other default
values
Generate DML Insert statement for Set instance
END
ELSE
BEGIN
Test if Set instance Exists
IF Set instance Exists THEN
Generate DML Update statement for Set instance
ELSE
Generate DML Insert statement for Set instance
END
Return SQL DML statement for Set instance
END

The structure chart and source code associated can be found in Appendix A UniGuide
SQL3 Generator Library (A.1.2 DML Statement Generation Modules: Set
instances).

In the following excerpt of an interactive msql session, we can identify three possible
scenarios respect to the generation of DML statements on set instances:

1. Parent row and set instance exist: the query specified below, searches for the
catalogues associated to a particular library:

* SELECT uni_id, name, cat_url, cat_name, cat_type
  FROM library, _library_scatalogs
  WHERE uni_id = 'University of Wollongong' AND
  name='University of Wollongong Library' AND
  library.oid = _library_scatalogs.poid;
<table>
<thead>
<tr>
<th>Uni_id (PK comp.)</th>
<th>Name (PK comp.)</th>
<th>cat_url (Pseudo PK comp.)</th>
<th>cat_name (Pseudo PK comp.)</th>
<th>cat_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>University of Wollongong</td>
<td>University of Wollongong Library</td>
<td><a href="http://130.130.68.7/">http://130.130.68.7/</a></td>
<td>INNOPAC</td>
<td>ON-LINE Web Catalogue</td>
</tr>
</tbody>
</table>

one row selected

2. A return statement that calls the `GF_complex_insert` statement is executed. The catalogue type has been changed:

* RETURN

```
GF_complex_insert('_library_scatalogs','uni_id^name^cat_url^cat_name^cat_type', 'University of Wollongong^University of Wollongong Library^http://130.130.68.7^INNOPAC^ON-LINE Web Catalogue');
```

one row selected

A valid nested update statement on set instances has been generated

2. **Parent row exists but set instance does not exist:** a new library catalogue is associated to a particular library:

* RETURN

```
GF_complex_insert('_library_scatalogs','uni_id^name^cat_url^cat_name^cat_type', 'University of Wollongong^University of Wollongong Library^http://www-library.uow.edu.au/Webmag.html^WebMAG^ON-LINE Magazine Catalogue');
```

one row selected

A valid nested insert statement on set instances has been generated.
3. Both parent row and set instance do not exist: a new library catalogue is
associated to a previously non-existent library:

* RETURN
GF_complex_insert('_library_scatalogs','uni_id^name^cat_url^cat_name^cat_type',
'University of Technology Sydney^University of Technology Sydney Library^ http://orac.lib.uts.edu.au/^UTS Catalogue (WWW)^WWW Library Catalogue');

<table>
<thead>
<tr>
<th>Uni_id (PK comp.)</th>
<th>Name (PK comp.)</th>
<th>Cat_url (Pseudo PK comp.)</th>
<th>cat_name (Pseudo PK comp.)</th>
<th>cat_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>University of Wollongong</td>
<td>University of Wollongong Library</td>
<td><a href="http://130.130.68.7/">http://130.130.68.7/</a></td>
<td>INNOPAC</td>
<td>ON-LINE Web Catalogue</td>
</tr>
<tr>
<td>University of Technology Sydney</td>
<td>University of Technology Sydney Library</td>
<td><a href="http://orac.lib.uts.edu.au/">http://orac.lib.uts.edu.au/</a></td>
<td>UTS Catalogue (WWW)</td>
<td>WWW Library Catalogue</td>
</tr>
</tbody>
</table>

one row selected

A valid nested insert statement on set instances has been generated and a parent row
object has been inserted with primary key and other default values.

The select statement specified in case 1 is re-executed:

2 rows selected

4.5.2.5.2 Query Statement Generation Modules

Query statement generation modules can only be called from Web Datablade
applications by client Web browsers. The application detects a priori the type of
construct being queried. Specific constructs require specific modules as in DML
statement generation modules. In this case, an additional number of parameters are
required. The following parameters are indicated in a RETURN statement:
• Name of the table or set
• A list of attributes associated to the table or set
• A list of boolean values that indicates which attributes are included in the query
• A list of attribute search conditions
• A list of attribute values

All lists are delimited by the '^' character. Parameters are dynamically captured from Web browsers, when end-users submit data to the server. In the generation of SELECT instructions over tables, attribute search conditions are transformed into WHERE conditions, giving us the possibility of creating elaborated WHERE clauses. A prototype algorithm for automatic generation of SELECT statements for row instances is included below:

```
BEGIN
  Convert list of Attributes, Boolean values that indicates presence of Attributes in the query, Attribute search options and Attribute values into Arrays
  FOR each Attribute DO
    BEGIN
      Get Attribute Data Type
      Get Attribute Value
      Reformat Attribute Value into valid SQL Type
      IF Attribute is included in query THEN
        Add Attribute to SELECT clause
      IF Attribute has a Search Condition THEN
        Add Search Condition to WHERE clause
    END
  Generate SELECT statement
  Return SQL SELECT statement for row instances
END
```

The next interactive msq1 session executes a return statement that calls the GF_squery_simple module. A SELECT clause will be generated including all columns except state and fax. The third column, city has a search condition associated, which will generate a WHERE condition with the LIKE operator. An asterisk in the search condition indicates that all values are accepted, therefore no search condition is generated.
* RETURN GF_squery_simple('university', 'uni_id^name^city^state^phone^fax', '1^1^1^0^1^0','=^=^LIKE^=^=^=','*^*^Woll%^*^*^*');

GF_squery_simple

| SELECT uni_id,name,city,phone FROM university WHERE upper(city) LIKE upper('Woll%'); |

one row selected

Please refer to Appendix A UniGuide SQL Generator3 Library (A.2.1 Query Statement Generation Modules: Row instances), in order to find the corresponding structure chart and source code.

With respect to modules that generate SELECT statements for set instances, a more complex algorithm is required. In spite of the fact, that queries are generated from individual set types, a join between parent rows and child set instances is offered by default (parent-child objects are joined via OIDs). The reason for doing this, is that set instances may be considered as semantically incomplete. An individual set instance gives us information about a sub-component object, which requires of an external identifier provided by the parent row instance in order to give sense the information stored by set instances.

The prototype algorithm for automatic generation of SELECT statements for set instances has the following form:

```
BEGIN
  Convert list of Attributes, Boolean values that indicates presence of Attributes in the query, Attribute search options and Attribute values into Arrays
  FOR each Attribute DO
    BEGIN
      Get Column Type
      Get Attribute Data Type
      Get Attribute Value
      Reformat Attribute Value into valid SQL Type
      IF Attribute is included in query THEN
        Add Attribute to SELECT clause
      IF Attribute has a Search Condition THEN
        CASE Column Type
```
"search": Add Search Condition to Exterior WHERE clause (Parent Table)
"insert": Add Search Condition to Interior WHERE clause (Set)
END
END

Get Name of Parent Table that holds the Set
Generate SELECT statement Set
Return SELECT statement Set instances
END

Please refer to Appendix A UniGuide SQL Generator3 Library (A.2.2 Query Statement Generation Modules: Set instances), to find the corresponding structure chart and source code.

The following excerpt of an interactive mysql session executes a return statement that calls the GF_squery_complex module. In this case, a SELECT statement that retrieves all catalogues associated to a particular library is generated. The SELECT clause will include all columns:

* RETURN
GF_squery_complex('_library_scatalogs','uni_id^name^cat_url^cat_name^cat_type',
'1^1^1^1^1','=^=^=^=^=','University of Wollongong^*^*^*^*');

<table>
<thead>
<tr>
<th>GF_squery_complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT uni_id,name,cat_url,cat_name,cat_type</td>
</tr>
<tr>
<td>FROM _library_scatalogs.library</td>
</tr>
<tr>
<td>WHERE library.oid = _library_scatalogs.poid AND</td>
</tr>
<tr>
<td>poid IN (SELECT oid FROM library WHERE upper(uni_id) = upper('University of Wollongong')) ;</td>
</tr>
</tbody>
</table>

4.5.2.5.3 Query Filtering Modules

Query filtering modules are called from Web Datablade applications that offer the issuing of free manual SQL queries. The main purpose of these modules is to provide a security barrier against possible hostile SQL statements that can manipulate, damage, corrupt or destroy the database. ILLUSTR\_A SQL3 is so versatile that it allows the specification of queries that can retrieve and insert or update tuples at the same time. As a result, we must avoid that any ILLUSTR\_A expert has the opportunity to break the security barrier. Therefore only valid SQL SELECT statements are permitted.
A query filtering algorithm that parses SQL statements, tests if the statements refers to a valid SELECT operation and returns error codes/messages or a list of columns associated to the query is indicated below:

```
BEGIN
    Get pre-formatted SQL statement
    Convert pre-formatted statement into valid SQL statement
    Test if statement is a valid SQL SELECT instruction
    IF not errors in SQL statement THEN
        BEGIN
            Generate List of Columns in SELECT statement
            Return List of Columns in SELECT statement
        END
    ELSE
        Return Error Code/Error Message
    END
```

The ILLUSTRA SQL3 query engine permits the execution of the most unthinkable SQL statements. As a result, dangerous queries with nested inserts or deletes for example can be issued by any malicious end-user:

i.e. `SELECT (INSERT INTO the_table VALUES('The SQL3 hacker')) FROM the_table;`

i.e. `SELECT (DELETE FROM the_table) FROM the_table;`

A sophisticated Web form that allows end-users to specify complex queries is included in the *UniGuide* interface. Strict security features must be provided to avoid any potential damage to the ORDB. The GF_free_sql module rejects all non-SELECT SQL statements.

An interactive msql session that calls the GF_free_sql module follows next. A SELECT statement with the ability to delete row is rejected, returning an error state and an error message:

```
* RETURN GF_free_sql('select (delete from library) from library');
```
One row selected

In this msql session, a valid SELECT statement is issued and a list of columns in the SELECT clause is returned plus a message that indicates the absence of errors.

* `RETURN GF_free_sql('select * from _library_scatalogs');`

One row selected

The correspondent structure chart and source code can be found in Appendix A UniGuide SQL Generator3 Library (A.3 Query Filtering Modules).

**4.5.2.6 ILLUSTRATA Web-Datablade Applications**

Typical Web database applications are based on the execution of Common Gateway Interface (CGI) scripts written in Perl, C and so on. These CGI scripts interact with databases and generate dynamically HTML code. CGI scripts require a considerable effort in developing Web database applications. Fortunately, ILLUSTRATA supports an alternative to CGI scripts, the Web DataBlade, which enables a faster and easier development of Web database applications that interact with ILLUSTRATA ORDBs. HTML applications pages include Web DataBlade module tags and functions that dynamically execute SQL statements and format results.

The Web Datablade module is composed of the following components (*ILLUSTRA 1996*):
• **Webdriver**: the CGI or Netscape API (NSAPI) interface to the Web server.  
  Connects to Webdaemon processes and forwards HTML Web pages from the Webdaemon to the Web server.

• **Webdaemon**: is a client ILLUSTR A API application. Forwards requests from the Webdriver that demand application pages from the ORDBMS, executing the WebExplode function. It also returns the resulting HTML Web pages to Webdriver.

• **WebExplode** function: parses, builds and executes SQL statements embedded in application Web pages. Also, returns results to the Webdaemon.

• **Web Datablade module tags**: a set of built-in SGML compliant tags and attributes that enable the execution of SQL statements and the definition of user-defined tags within application Web pages.

• **Web Datablade module functions**: a library of functions that facilitate the design and maintenance of Web applications.

The architecture of the Web DataBlade module and the steps involved in executing Web Datablade applications is depicted in **fig. 4.12**:

![Figure 4.12 Web Datablade Module Architecture](image)

1. **A client Web browser makes a request** to the Web server to invoke Webdriver.

2. Based on configuration information (Web.conf file), **Webdriver composes the SQL statement** to request an application page from the Webdaemon.
3. **Webdaemon connects to the ILLUSTRAX ORDBMS** and executes the WebExplode function.

4. **WebExplode retrieves the application page** from the Web application table, executes SQL statements embedded in the application page by expanding Web Datablade module tags, and finally formats the results.

5. **WebExplode returns the resulting HTML** to the Web browser, passing through the Webdaemon, Webdriver, and the Web server.

Web Datablade module tags can be considered as some kind of macro programming language with limited procedural capability but with the power of issuing queries to ILLUSTRAX databases which enhances notably HTML. Error handling, conditional statements, conditional HTML blocks, variable processing functions, advanced query processing and formatting techniques are some of the best features found in Web Datablade module tags. The following table (table 4.8) lists the most important Web Datablade module tags:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;MISQL&gt;</code></td>
<td>Permits the execution of SQL statements</td>
</tr>
<tr>
<td><code>&lt;MIVAR&gt;</code></td>
<td>Creates, assigns, and displays variables</td>
</tr>
<tr>
<td><code>&lt;MIBLOCK&gt;</code></td>
<td>Delimits logical blocks of HTML</td>
</tr>
<tr>
<td><code>&lt;MIERROR&gt;</code></td>
<td>Permits error trapping</td>
</tr>
</tbody>
</table>

**Table 4.8** List of most important Web Datablade module tags

Another important add-on provided by the Web Datablade is a library of variable processing functions that enable the execution of arithmetic, string, and miscellaneous operations over variables passed into applications pages, generated within or returned from an ILLUSTRAX database. Variables are identified by a "$" followed by a variable name. Variable expressions start with a "$" followed by the expression within parenthesis.
We shall explain a simple generic Web Datablade application that permits the execution of SQL queries and display of formatted results:

1. An error handler is defined in order to trap errors generated and provide error information to end-users.

2. A form with a text area and a submit button to execute SQL statements is generated.

3. Once the form is submitted, values are assigned to the variables vquery and MIval. The MIval variable is a standard built-in variable used to indicate the next Web page to be retrieved and pass additional parameters (in this case, the call is recursive).

   The vquery variable activates the conditional HTML block, therefore it can be considered as a control variable.

4. The SQL statement is executed in a MISQL tag, which is linked to the error handler specified at the beginning with a MIERROR tag (1).

5. If the query is successful results are formatted into tabular form.

6. On the contrary (error has been generated), a virtual unnamed HTML Web page is dynamically generated displaying an error code/error message.

```html
<html>
<body>
<!-- (1) Define error handler before other tags -->
<?mierror err=ErrHandler1 tag=mivar>
<!-- (2) Form to issue SQL statements -->
<form name=frm0 method=post action=<?mivar>$WEB_HOME<?/mivar>>
<textarea name="vSQL" cols=40 rows=3"></textarea><br>
<input type=submit name=button value="Execute">
<!-- parameter: yes value is assigned when submitting the form -->
<input type=hidden name=vquery value=yes>
<!-- parameter: MIval stores name Application Page to be called -->
</form>
<!-- Virtual HTML Web page is generated with error info. -->
<html>
<head><title>Error Page</title></head>
<body>
<h2>Error Page:</h2>
<br>
<!-- Display ILLUSTRATA Error Code/Error Message generated -->
<strong>Error Code($MI_ERRORCODE):</strong>$MI_ERRORMSG<br>
<br>
</body>
</html>
</body>
</html>
```
The general method applied in UniGuide in generating Web Datablade applications that execute SQL statements against ILLUSTRATA databases consists of:

- **The definition of generic Error Handlers:** define with the MIERROR tag generic error handlers that perform error trapping and provide additional information about error sources. Must be declared at the top of the application so when errors are generated at some point, the error handler is already activated.

- **The definition of form variables:** these are required to pass attribute/value pairs to ILLUSTRATA servers. Two types of form variables have to be declared:
  
  1. **Interactive form object variables:** variables that represent textboxes, text areas, select lists, toggle boxes or radio buttons
  
  2. **Non-interactive form variables:** hidden variables that are passed as parameters when submitting forms. These types of variables serve as control variables that activate HTML blocks or invoke other applications. In our case, we advocate the use of recursive calls on the basis that the logic is centralised, the HTML source code is reused and from an end-user perspective the interface remains more stable ("common look and feel")
• **The definition of conditional HTML blocks:** conditional blocks are conceptually equivalent to procedural IF blocks. If a given condition is satisfied the HTML block is entered.

• **The execution of a two-phase SQL statement:** two-phase SQL statements are executed in conditional HTML blocks activated when submitting forms. As mentioned in the *UniGuide SQL3 Generator* section (Fig. 4.11 Phases in the execution of a DML/SELECT statement in the *UniGuide SQL3 Generator*), SQL statements are firstly dynamically generated by the *SQL3 Generator* with SQL RETURN statements that call to external user-defined C functions and secondly if successfully created, are executed immediately after. Therefore two MISQL tags are defined, one with a RETURN statement, and the other, executing the resulting DML or SELECT statement.

• **Dynamic generation of form objects:** form objects are generated from SQL SELECT statements, therefore the maintenance of the application is reduced drastically. As a result with one forms-based Web application we are capable of representing a whole set of tables or set types.

Some interesting methodologies for structured hypermedia design have been proposed by different authors (*Schwabe and Rossi 1998; Isakowitz et. al 1995*). Unfortunately none of them adequately fits our needs, therefore we were forced to invent a new graphical tool, in order to present an intuitive way of understanding how Web applications are developed and the modularity involved. We provide two graphical representations: top level navigational charts and detailed level WebApp charts. Both charts are similar, the difference between them is only on the level of detail expressed. Top level navigational charts (**TLNC**) show the hierarchies of Web pages and the
possible navigations between these in a Web Datablade application. A TLNC includes the following elements with the corresponding symbols depicted in table 4.9:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Notation</th>
<th>Symbol</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Web Page" /></td>
<td>Web Page</td>
<td><img src="image" alt="Load process" /></td>
<td>Load process</td>
</tr>
<tr>
<td><img src="image" alt="Frame" /></td>
<td>Frame</td>
<td><img src="image" alt="URL" /></td>
<td>URL Link</td>
</tr>
<tr>
<td><img src="image" alt="Form" /></td>
<td>Form</td>
<td><img src="image" alt="URL_Foreign_Key" /></td>
<td>URL Foreign Key Link</td>
</tr>
<tr>
<td><img src="image" alt="Navigational path" /></td>
<td>Navigational path</td>
<td><img src="image" alt="Plink" /></td>
<td>Parameterizable link</td>
</tr>
</tbody>
</table>

Table 4.9 List of TLNC elements/symbols

- **Web Page**: represents single Web pages
- **Frame**: represents by default a Web page containing a frameset loaded with two frames.
- **Form**: by default represents a single Web page with an embedded form
- **Navigational path**: possible navigation between Web Pages. Materialised when submitting a form or clicking on a link (Plink, URL, URL_Foreign_key)
- **Load process**: used specially in frames to indicate in which frame a Web page is loaded
- **Parameterizable link**: a link that includes parameters of the form of name/value pairs (i.e. URL&variable_name="variable_value")
- **URL Link**: link to the URL associated to row or set instance retrieved
- **URL Foreign Key Link**: link to the URL associated to the row instance referenced by a foreign key value

On the other hand, WebApp structure charts are similar in concept to procedural structure charts, in that the modularity of the application and parameter passing is
shown. In addition a navigational path can be derived from HTML form submissions or following links.

A page-based Web DataBlade application cannot be expressed in a pure OO notation, showing objects and message passing as there are no custom methods associated with particular Web pages, only limited sets of built-in methods (from a Javascript object hierarchy viewpoint). To make things even more difficult, we are dealing with server-side procedural HTML blocks and with a client-side event-driven object-based environment. WebApp structure charts are capable of representing in an intuitive way the events and conditions necessary to activate other parts of the application. In fact, we distinguish three types of WebApp structure charts:

- **Server-side**: the focus is on the server-side, characterised by being procedural. The majority of processes are executed on the server-side.

- **Client-side**: the focus is on the client-server, characterised by being event-oriented. The majority of modules are internal to a Web application and are activated on the Web browser (by the Javascript interpreter).

- **Client-Server interaction**: both client and server sides are included, showing only top-level client-based modules and the entire range of server-side components. The components that are formed by integrating or collaborating both sides are marked as C/S (Client/Server). Server-side activated components are marked with an S symbol and client-side activated components are marked with a C symbol. This type of chart is critical in the understanding the logic involved in Web applications due to the fact that some parts of the code are almost cryptic. In addition, these charts give us the possibility to understand intuitively the tight integration between client and server side components.
A WebApp structure chart includes the following elements and the corresponding symbols, depicted in table 4.10:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Notation</th>
<th>Symbol</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebApp/Web Page</td>
<td>![Symbol]</td>
<td>Load process</td>
<td>![Symbol]</td>
</tr>
<tr>
<td>WebApp/Frame</td>
<td>![Symbol]</td>
<td>Call/link process</td>
<td>![Symbol]</td>
</tr>
<tr>
<td>WebApp/Form</td>
<td>![Symbol]</td>
<td>Generate process</td>
<td>![Symbol]</td>
</tr>
<tr>
<td>Sub-component list</td>
<td>![Symbol]</td>
<td>![link]</td>
<td>Parameterizable link</td>
</tr>
<tr>
<td>Virtual Web Page</td>
<td>![Symbol]</td>
<td>Client-side</td>
<td>![Symbol]</td>
</tr>
<tr>
<td>Internal module</td>
<td>![Symbol]</td>
<td>Server-side</td>
<td>![Symbol]</td>
</tr>
<tr>
<td>External module</td>
<td>![Symbol]</td>
<td>![Symbol])</td>
<td>Client-server side</td>
</tr>
</tbody>
</table>

**Table 4.10** List of elements/symbols included in WebApp structure charts

- **WebApp/Web Page**: represents single application Web page component
- **WebApp/Frame**: represents by default a Web page including a two-element frameset
- **WebApp/Form**: single application Web page containing embedded forms
- **Sub-component list**: represents a list of row/set instances with optional or mandatory links (Plink, URL link or URL_Foreign_key link)
- **Virtual Web Page**: symbolizes an unnamed virtual Web page that is generated dynamically
- **Internal module**: internal Javascript functions embedded in WebApp components such as Web pages, frames or forms. These modules become active at the client side
• **External module**: external module that provides additional functionality to the Web Application. Generally activated at the server-side

• **Load process**: used specially in frames to indicate in which frame a Web page is loaded

• **Call/link process**: may include calls to external modules or links formed by submitting forms or following links

• **Generate process**: symbolizes components such as parameterised links, GUI objects, etc. that are dynamically generated

• **Parameterizable link**: represents links that includes parameters on the form of name/value pairs (i.e. URL?variable_name=variable_value)

• **Client-side**: functions/modules that become active at the client-side. In the construction of *UniGuide* generally includes Javascript functions

• **Server-side**: functions/modules that become active at the server-side, interacting with the ILLUSTRAR ORMDBMS

• **Client-server side**: Web application components that become active at both sides or are formed by interacting both client and server sides

Parameters are expressed as in traditional structure charts, differentiating between variables (normal variables or variable vectors) and control variables. Also, conditions specified by variable processing functions are included.

### 4.5.2.6.1 Insert/Update Web applications

HTML forms-based insert/update Web applications are characterised by allowing inserting or updating row/set instances. Data validation and data formatting is performed on the client-side whereas the Web application interacts and coordinates
between the client and server-side of the system. The parameters to be captured from an HTML form submission include the name of the table or set table and the columns and values associated. After submitting data to the server, the Web application receives the parameters and directs data to the proper **SQL3 Generator** module depending of the type of construct being inserted or updated. These external modules detect automatically if the object exists or not which determines the generation of insert or update SQL3 statements. Either an error or a successful DML statement is generated. In a successful case, the DML statement is captured and executed. On the contrary, if an error has occurred, this is trapped by an error handler defined by a MI_ERROR tag, which in turn generates a virtual Web page with error information. The main_ins Web application is based on a total of four physical Web pages. Thanks to the use of recursive calls the number of needed Web pages is reduced drastically.

The TLNC of the main_ins Web application has the following form (fig. 4.13):

![Figure 4.13 TLNC main_ins Web application](image)

The Server-side WebApp Structure Chart of the Web application main_ins is (fig. 4.14):
4.5.2.6.2 Query Generation Applications

Queries are generated "on the fly" from data captured from form submissions. The main_squery Web application has a total of four Web pages. The parameters to be captured from a form submission include:

- Name of the table/set table: v_table_source
- The columns and values associated v_fields and vobject respectively
- A list of boolean values (v_includes) indicating the inclusion of the column in a SELECT clause and
- A list of attribute search conditions (v_options)
When a HTML form is submitted (indicated by the control variable `v_genquery`), a recursive call that passes parameters to the `form_squery` tests the type of object being submitted and directs data to the proper **SQL3 Generator** module, in order to build an SQL3 SELECT statement. If the returned parameter is a valid SQL3 statement, row or set instances including links to external Web pages are retrieved. If an error has occurred, a virtual Web page containing error information is generated.

The TLNC of the Web application of `main_squery` Web application is indicated below (fig. 4.15):

![Figure 4.15 TLNC main_squery Web application](image)

The Server-side WebApp structure chart of the Web application `main_squery` has the following form (fig.4.16):
4.5.2.6.3 Query filtering Applications

The free_sql Web application is based entirely on one Web application page, although recursive calls are performed in order to process form submissions. The parameters passed from a form submission includes a pre-formatted string representing an SQL
select statement (v_sql) and a control variable (v_query) that indicates that a form submission has been executed. When a form is submitted a pre-formatted SQL string is passed to the adequate SQL3 Generator module that checks the validity of the SQL3 SELECT statement. The return parameter either specifies an error or a list of columns. Non-permitted SQL statements are rejected presenting a virtual Web page with error information. In a successful attempt a list of columns is captured to present row or set instances in a tabular format. Recursive calls are executed from form submissions or following parameterizable links. Parameterised links implement a sliding window in which a subset of rows is retrieved from a particular query. The parameters passed from a parameterised link are the starting point from which rows are retrieved (v_Begin), a control variable that indicates that the sliding window is activated (v_Walk), a label associated to the link (v_label) and the original SQL statement issued (v_sql).

The TLNC of the free_sql Web application is (fig. 4.17):

![Figure 4.17 TLNC free_sql Web application](image)

The Server-side WebApp Structure Chart of the free_sql Web application has the form depicted below (fig. 4.18):
4.5.3 Client-side: Javascripts

In a client-server paradigm, the execution of logical processes is distributed between the server and client. In our case, server-side processes are currently being executed on a Sun workstation under Solaris. On the client-side, UniGuide provides support to any Javascript enabled client browser. An alternative to this architecture is to develop JAVA applets embedded in Web pages. As a rule of thumb, search engines must provide an interface, which provides backward compatibility with the majority of browsers in order to become popular. Implementing JAVA enabled Web pages reduces significantly the number of users who can have access to the UniGuide database search engine. We may cite as an example that all Navigator browsers under Microsoft Windows 3.1 would not
be able to run JAVA applets as these browsers do not include the JAVA Virtual Machine (interpreter).

Web Datablade applications and client-based Javascripts are tightly integrated in order to create data error proof applications. Web Datablade applications interact with ILLUSTRAR ORDBs whereas client-side Javascript modules augment the efficacy of server-side applications, delegating some server functionality to client Web browsers. With the incorporation of Javascript functions, Web browsers are transformed from "dumb" to thin clients. An alternative option would have been the deployment of client applications that interact with the server-side in JAVA™ via a JDBC driver for ILLUSTRAR. We opted for the combination of Web DataBlade applications and Javascripts, which virtually offers the same functionality as JAVA applications with the additional advantage of offering the possibility to access UniGuide from any Javascript enabled browser. Javascript is an object-based event-driven scripting programming language, which provides a set of standard objects and facilities for manipulating them and for creating customised objects. Although Javascript does not allow the definition of classes, inheritance and polymorphism, still we have the opportunity to create custom object with constructor methods and define encapsulated functions associated to the object. The most powerful feature available in the Javascript language is an object model associated to HTML based on the W3C Document Object Model (DOM). A Web page is a container composed of a series of sub-components objects. The model presents a hierarchy of objects, that ranges from a window object on the top level, to links and built-in string objects. The following figure (fig. 4.19) shows a partial view of the Javascript object hierarchy:
The main purpose for using Javascript code, is the dynamic generation of form objects (i.e. textboxes, textareas, etc.), data validation, data re-formatting, interface construction and so on.

4.5.3.1 Insert/Update modules

Due to the tight integration between server-side Web Datablade applications and client Javascripts embedded in Web pages, it is very difficult to explain the functionality provided by Javascript modules as isolated subroutines, although in a next section we will explain in detail the sophisticated coupling between both. The major tasks executed...
by Javascript functions in insert/update modules (main_ins) is included in the following list:

- **Generation of a graphical tree structure table browser:** this module has been adapted from free public domain software. A tree structure is generated with the use of recursive calls. Each node in the tree structure represents a parameterised link that stores a table or set table value, which in turn generates forms dynamically depending on the value stored in the node.

- **Generation of an array of column objects:** column objects store attributes associated to columns retrieved from the Auxiliary Catalogue System. Contextual help, column name, column alias, boolean value that indicates mandatory value and an index value are stored in an array of column objects.

- **Contextual help:** each form object is created with contextual help by defining functions associated to event handlers.

- **Data re-formatting:** data is reformatted before submitting values to the server

- **Data validation:** data is validated before the form is being submitted

The Client-side WebApp structure chart of the Web application main_squery has the following form *(fig. 4.20):*
4.5.3.2 Query Generation modules

The major tasks executed by Javascript functions in query generation modules (main_squery) is included in the following list:

- **Generation of a graphical tree structure table browser**: a Windows file explorer tree structure like is generated with the use of recursive calls. Each node in the tree structure represents a parameterised link that stores a table or set table value, which in turn generates a form dynamically depending on the value stored in the node.

- **Generation of an array of column objects**: column objects store metadata associated to columns retrieved from the **Auxiliary Catalogue System**. Contextual
help, column name, column alias and an index value are stored in an array of column objects.

- **Contextual help:** each form object is created with contextual help by defining functions associated to event handlers.

- **Data re-formatting and validation:** form data is re-formatted and validated before submitting values to the server.

- **Transformation of URL columns into links:** URL columns retrieved from SQL queries are dynamically transformed into HTML links.

The Client-side WebApp structure chart of the Web application main_squery has the following form:

![Client-side WebApp structure chart main_squery Web application](image)

**Figure 4.21** Client-side WebApp structure chart main_squery Web application

4.5.3.3 Query filtering modules
The major tasks executed by Javascript functions in query generation modules (free_sql) is included in the following list:

- **Generation of an array of column objects**: column objects are dynamically created when loading the form. Each object stores a column source name and a table source name, therefore a select list that shows columns associated to a given table can be fed "on the fly".

- **SQL3 query re-formatting and validation**: the validity of queries is tested on the client-side. Valid queries are reformatted before submitting the query string to the server.

- **Transformation of URL columns into links**: URL columns retrieved from SQL queries are dynamically transformed into links.

The Client-side WebApp structure chart of the Web application free_sql has the following form (fig. 4.22):

![Client-side WebApp structure chart free_sql Web application](image)

**Figure 4.22** Client-side WebApp structure chart free_sql Web application

4.5.4 Integration and interaction between server-side and client-side
To really understand the behaviour of client-server Web applications we must describe the series of events and actions that occur when a form is loaded and when a form is submitted or when a parameterised link is followed. **Taking into account that the HTTP protocol is a stateless protocol where connections between client and server are open during a single operation and that HTML source code is interpreted sequentially from top to bottom,** we can distinguish the following events and actions:

- **Initial OnLoad form:** when a form is initially loaded called by a parameterised link, parameters are passed activating default HTML blocks, which in turn can call to server-side modules. May implicate the execution of server-side queries, whose results are captured by client modules in order to dynamically generate form objects or Javascript custom column objects.

- **OnSubmit form:** form data values are passed to the server in recursive calls. Data is validated and reformatted on the client-side previously to the form submission. Form submissions are cancelled when incorrect data input is detected.

- **OnReload form:** depending on the parameters received from the recursive call, specific HTML blocks are enabled. As data is being received from the server, this is pre-processed by client modules and finally displayed on the Web browser.

The complexity of developing client-server Web applications partly resides in demarcating and identifying when client and server-sides must be activated and how both sides interact and collaborate. We can distinguish two kinds of collaboration between client and server side in *UniGuide* Web applications:

- **From server to client side:** data is received from the server and processed a-posteriori by the client. May include generation of form objects, generation of table
lists of row/set instances. In this case, client modules are activated procedurally by conditions.

- **From client to server side:** data is pre-processed and validated on the client-side and sent to the server-side when correct. Includes form data validation and reformatting. In this case, client modules are typically activated by events.

### 4.5.4.1 Insert/Update modules

In the main_ins Web application a tree-like table browser is generated, which stores table/set table names. When the user clicks on a given node (PLink), the form_ins is called with a parameter that stores the table being referenced (v_table_source). An SQL statement that retrieves column data (auxiliary catalogue system) associated to the table/set table generates form objects (vobject) and an array of Javascript custom column objects (aobjCols_init). These column objects are needed to provide contextual help and for validation purposes. When the form_ins form is submitted, data is validated and reformatted (LF_tst_DataIsOk) taking into account information stored in column objects (aobjCols). Parameters are passed in a recursive call to the form. Depending on the type of table (row/set) an HTML block is activated executing a call to the required **SQL3 Generator** module, which in turn generates a DML statement (v_sqlinsert) that is executed immediately after if the DML statement is correct (Two Phase DML Statement). If the SQL3 statement is incorrect an error is generated and trapped by the error handler showing a virtual Web page with error information.

The Client-Server interaction WebApp Structure Chart of the main_ins Web application has the form depicted below *(fig. 4.23)*:
Figure 4.23 Client-Server interaction WebApp structure chart main_ins Web application

The state transition diagram (STD) and source code of the main_ins Web application can be found in the Appendix B. Web-Datablade Applications and Embedded JavaScripts (B.1 Insert/Update Web-Datablade Applications).

4.5.4.2 Query Generation modules

In the main_squery Web application a tree-like table browser is generated by combining client and server effort. When end-users click on a parameterised link (PLink) of the tree-like table browser, a parameter that stores the table/set table being referenced (v_table_source), is passed to the form_squery. An SQL statement that retrieves from
the **Auxiliary Catalogue System** column metadata associated to the table/set table will enable the generation of form objects (v_object) and an array of Javascript custom column objects that store the captured metadata (aobjCols_init). These column objects are needed to provide contextual help and to validate data input. When the form_squery form is submitted, data is validated and reformatted (LF_tst_Qry_Values) taking into account information stored in column objects (aobjCols). Parameters are passed in a recursive call to the form. Depending of the type of table (row/set) an HTML block is activated executing a call to the required **SQL3 Generator** module, which in turn generates a specific SELECT statement (v_sqlquery) that is executed immediately after (Two Phase Select Statement). If the SQL statement is correct, it retrieves row/set instances that contain URL links and other attributes. URL links and URL foreign key links attributes are transformed into HTML links by the client module LF_mark_hyperlink. If the SQL3 SELECT statement is incorrect an error is generated and trapped by the error handler showing a virtual Web page with error information.

The Client-Server interaction WebApp Structure Chart of the main_squery Web application has the form depicted below (**fig. 4.24**):
The state transition diagram (STD) and the source code of the main_squery Web application is included in the Appendix B. Web-Datablade Applications and Embedded JavaScripts (B.2 Query Web-Datablade Applications).
4.5.4.3 Query filtering modules

In the initial load of the free_sql Web application, an SQL statement that retrieves all column names (Auxiliary Catalogue System) associated to respective table/set tables, generates an array of Javascript custom column objects (LF_feed_array). The array of column objects (aobjCols) is needed to dynamically feed a select list that contains column names associated to a particular table. When the free_sql form is submitted, the specified SQL statement is validated and reformatted (LF_test_query). If the SELECT statement is valid, parameters are passed in a recursive call to the form. An HTML block is activated executing a call to an external SQL3 Generator query filtering module (GF_free_sql), which in turn returns an error message for non-allowed queries or returns the list of columns included in the SELECT clause of the query (v_columnlist). If the SQL statement is correct, it is executed immediately after, retrieving row/set instances that may contain URL links and other attributes. URL links attributes are transformed into HTML links by the client module LF_mark_hyperlink. If the SQL3 SELECT statement is incorrect an error is generated and trapped by the error handler showing a virtual Web page with error information. The Client-Server interaction WebApp Structure Chart of the free_sql Web application has the form depicted below (fig. 4.25):
Please refer to the Appendix B. Web-Datablade Applications and Embedded JavaScripts (A.3 Query Filtering Modules) to obtain the STD and source code of the Web application free_sql.

4.5.5 Meta data design: UniGuide scheme meta-tags

UniGuide scheme meta tags have been designed to be easily understood by information providers and to be easily parsed by the UniGuide indexing agent. These meta tags may be generated by customised meta tag generators. From a database viewpoint we can view UniGuide meta tags as structured tuples of metadata associated to and located
on the top of target Web pages. The general form for a UniGuide meta tag is described below (fig. 4.26):

```xml
<!-- mandatory columns marked with -->
<!-- Please Enter Values inside ' ' -->
<meta name="(table_name | set_table_name)" scheme="UniGuide"
content=
(~ {column_name }["["column_alias "]" =’’ ~) [,]} >>
```

**Figure 4.26** Formal syntax of a UniGuide meta tag

The scheme attribute is a standard compliant to the HTML 4.0 specification. In addition, considering that HTML is very tolerant, browsers that are compliant to older HTML versions may ignore the scheme attribute, not producing any errors associated. As a result, any target Web page can become a potential container of UniGuide meta tags.

Thanks to the scheme attribute the UniGuide indexing agent can identify immediately the required meta tags to be parsed. The name attribute stores the name of the table or set table to be populated with data. The content attribute stores information about the entire set of columns associated to the table/set table with the exception of the URL and %last_modified attributes. On the left hand-side of the equal character is indicated the name of the column and an intelligible column alias enclosed in brackets. If the column alias starts with an asterisk, this symbol indicates that the column is mandatory, therefore a value is required. On the right hand-side of the equal expression, the values associated to the respective column are included. Each (column_name, column_value) pair is enclosed between a tilde character. These are considered as delimiters with the combination of the parenthesis characters. We were forced to find a combination of characters, which are very infrequent to appear, avoiding the incorrect parsing of UniGuide meta tags. A more intuitive example follows next (fig. 4.27):
Figure 4.27 An example of a UniGuide meta tag

In this case, the meta tag represents a course/degree set instance associated to an academic entity. In other words the meta tag describes the relevant information about the Bachelor of Science (Computing Science speciality) issued by the Computer Science Department at the University of Technology in Sydney.

4.5.5.1 UniGuide Meta Tag Generator

The main_meta Web application presents to end-users the common tree-like table browser in order to permit the selection of a particular entity. When end-users click on a parameterised link (PLink) of the tree-like table browser, a parameter that stores the table/set table being referenced (v_table_source), is passed to the form form_meta. An SQL statement that retrieves column metadata (Auxiliary Catalogue System) associated to the table/set table enables the generation of form objects (v_object) and an array of Javascript custom column objects (aobjCols_init). These column objects store column information such as column name, alias, a boolean value that indicates if the column is mandatory and finally help text associated to the column. When end-users click on the "Generate" button a client-side module (LF_Generate) tests and reformats...
input data. If the input data is correct, the module automatically generates a compliant UniGuide Scheme meta tag, that can be cut and pasted in order to insert it into the header of a target Web page.

The Client-Server interaction WebApp Structure Chart of the main_meta Web application has the form depicted below (fig. 4.28):

![Client-Server interaction WebApp structure chart main_meta Web application](image)

**Figure 4.28** Client-Server interaction WebApp structure chart main_meta Web application

### 4.5.6 UniGuide Indexing Agent

The *UniGuide indexing agent* constitutes a special case of indexing robot, presenting its own specialised features, compared to the typical centralised indexing robots referred to in **subsection 2.4.3 (Robot-based centralised indexing)**. Firstly, full-text indexing is ignored, and secondly URLs are temporarily stored only to avoid Web pages that have been already scanned by the *UniGuide indexing agent* in the current indexing process.
The objective is to detect, parse and store in a breadth-first manner the information contained in *UniGuide* scheme meta tags. As a result the indexing process of a whole university domain instance is likely to be much faster than the one executed by a full-text indexing agent. Due to the fact, that the research prototype has been located on a relatively modest RISC processor based Sun workstation with 64Mb of RAM and 100 MB of swap space, we were forced to store temporarily the list of URLs visited and to visit on relational tables. Unfortunately both ILLUSTRA and its JDBC driver (*Kalvelagen 1997*) presents some considerable limitations. First of all the JDBC driver requires an additional daemon (bridged) that redirects client requests on a given port number to the midaemon (*see fig. 4.29*). And secondly, based on our experience we have found that ILLUSTRA under this type of JDBC implementation can handle at the most three simultaneous client connections. As a matter of fact we had to reduce the number of concurrent threads that can perform concurrently URL connections to a minimum of two, which added to client database requests performed by a main thread gives us exactly a total of three simultaneous concurrent client requests. As a proof of correct execution, the prototype has scanned the entire University domain instance located at the University of Wollongong, starting from the university homepage: [http://www.uow.edu.au](http://www.uow.edu.au). In addition a second prototype version has been implemented storing both the URLs visited and to visit in memory. This version required a much more powerful server with a 256 Mb of RAM and 200 Mb of swap space, and had to be executed over the weekend unloading a majority of residents programs. The throughput of URLs visited is dramatically faster than the hard drive version. This should be the ideal version, giving the possibility to the *UniGuide* indexing agent to index more frequently university domains. From a programming perspective both versions have been developed in JAVA™ and some parts in C. Logically they are very similar. The
difference is mainly in the type of objects used to store the list of URLs visited and remaining to be visited, therefore we have included only the hard drive version.

![Diagram of Double daemon architecture in the ILLUSTRATE JDBC Driver](image)

**Figure 4.29** Double daemon architecture in the ILLUSTRATE JDBC Driver

Considering that JAVA™ is a pure Object Oriented Programming Language (OOPL), the architecture of the UniGuide indexing agent is entirely based on the definition of classes and object instantiations. The following list includes the classes defined in the implementation of the prototype:

- **app_Index_Robot_HD**: main application class that coordinates and monitors the indexing process in a breadth-first manner
- **Robot_Stats**: class used to store general indexing agent statistics such as number of URLs visited, number of errors generated, number of UniGuide scheme meta tags processed and so on
- **URL_Scanner**: class represents individual threads that scan and extract URLs and UniGuide scheme meta tags referenced in the contents of a given Web page
- **JDBC_MetaData Saver**: class implements the JDBC interface to UniGuide scheme tables, which permits the storage of the captured meta tags
- **JDBC_URL_Saver**: class implements the JDBC interface to the tables URL_catalogue (stores URLs visited) and URL_to_visit (stores the next URLs to be visited)
• **ThreadLimit**: class is used to limit the number of concurrent URL_Scanner threads executing simultaneously

• **Stop_URL_Scanner**: class is defined to implement a daemon that stops URL_Scanner threads that have exceeded the maximum allowed time.

The following figure (fig. 4.30) shows the GUI of the *UniGuide Indexing Agent*. There are five well-defined sections:

• **URL text field**: end-users input the starting URL to begin the indexing process

• **URL information text area**: displays information of the URL currently being visited

• **Meta data text area**: displays meta data statistics, which includes the current number of meta tags being processed, the last entity instance type detected, and the status code returned after processing the meta tag (ignored, inserted, updated or error).

• **Robot Statistics text area**: displays general indexing agent statistics, indicating the current number of URLs visited, the number of URL in the URL catalogue table, the number of URLs to be visited, the number of errors generated, the number of *UniGuide* meta tags detected and the number of entity instances inserted or updated.

• **System statistics text area**: displays information about the current number of concurrent URL_Scanner threads, the number of stopped URL_Scanner threads, the time that was initiated the indexing process and the number of milliseconds that have passed since starting the indexing process.
Figure 4.30 UniGuide Indexing Agent: GUI

Figure 4.31 UniGuide Indexing Agent: UniGuide Scheme Meta tags detected
Figure 4.31 presents an example that indicates that the *UniGuide* indexing agent has detected some *UniGuide* scheme meta tags. A complete object oriented class diagram that shows in detail all the classes involved and the relationships among them is shown in figure 4.32 in UML notation. The JAVA™ and ANSI C source code can be found in Appendix C. (*UniGuide Indexing Agent*). The HTML documentation automatically generated by the JAVADOC utility can be found at:


The pre-requisite to generate this HTML documentation is to follow the syntax specified in (Sun 1998). A complete list of classes that show all attributes and methods associated is included, which will definitely help the reader to understand the logic of the program.

### 4.5.6.1 app_Index_Robot_HD

This class generates the GUI, which will allow end-users to specify a starting URL from which Web pages will be scanned and navigated in a breadth-first manner. When an end-user inputs the starting URL a main application thread is created and started. Due to the fact that the main thread runs independently, we are able to stop or exit the application when desired. When the main thread is started, the respective run method associated is called. The string URL is captured and the respective common domain name is extracted (i.e. An input of "http://www.monash.edu.au" results a domain name of "monash.edu.au") which restricts the indexing process to only servers associated to the university domain. A Robot_Stats object is instantiated in order to begin to store indexing agent statistics. An initial URL_Scanner thread is spanned and waited to its end of execution, so an initial list of URLs to be visited will be generated. A daemon
that stops delayed URL_Scanner threads is started (Stop_URL_daemon). In addition, a loop that asks for permission to extract the next URL to be visited and that spans a new URL_Scanner thread, is executed until the number of URL_Scanner threads is zero and there is no more URLs to visit. URL_Scanner objects are instantiated when the current number of concurrent URL_Scanner threads does not reach the maximum number allowed. The applet periodically displays system (no. of threads, no. of threads stopped, time elapsed and so on) and indexing agent statistics.

4.5.6.2 Robot_Stats

The Robot_Stats class is used to store the following indexing agent statistics:

- No. of URLs visited
- No. of URL in the URL_catalogue table
- No. of URLs to visit
- No. of http and JDBC erroneous requests generated. In the case of JDBC errors, URL row collisions are included.
- No. of UniGuide scheme meta tags detected
- No. of meta tags inserted or updated in the ILLUstra ORDB

4.5.6.3 URL_Scanner

The URL_Scanner class is used to generate object instantiations that implement autonomous independent threads that can run concurrently. The main goal of a URL_Scanner thread is to parse retrieved Web pages, detect, extracts and store UniGuide meta tags if present and fetch all links referenced in the Web page contents so other URL_Scanner threads may continue the navigation process. When a
URL_Scanner object is instantiated a respectable number of parameters are passed. The most critical ones are:

- The URL to be scanned
- The domain name string to restrict URL fetching to the university domain
- A ThreadLimit object (Gate) to give permission to other waiting threads when done and
- A Vector object (a dynamically resizable array object with the capability of storing any kind of object) that stores the list of current active URL_Scanner threads (tTable).

When a URL_Scanner is started executing the run method, the current URL_Scanner thread is added to the list (tTable) and a URL connection is opened. This connection allows the retrieval of both information from http headers and the contents of the object itself. The http header contains metadata associated to the http protocol that describes what kind of information is being retrieved, number of bytes to be transferred, date of creation and last modification of the object, and so on. URL_Scanner threads retrieve from the http header the date of last modification associated to the Web page and reformat the contents of this attribute into a valid ILLUSTRAR timestamp format. After capturing this information, Web page contents are transmitted line by line. The next step consists of detecting **UniGuide** scheme meta tags (Get_MetaData method). A successful attempt will span a JDBC_MetaData_Saver object instantiation that will interact with a **UniGuide** scheme table/set table. Finally the Web page contents is re-scanned again in order to find all types of URL being referenced (links, image maps, and frames). All relative URLs are converted into absolute URLs, filtering only those that point to Web pages under the http or ftp protocol and that pertain to the same university domain.
4.5.6.4 JDBC_MetaData Saver

This class is used to store UniGuide scheme meta tags. Implements the JDBC Interface to any table/set table of the UniGuide schema. When a JDBC_MetaData_Saver object is instantiated by a URLScanner thread, the JDBC interface is initialized with the following critical parameters needed to generate SQL statements that interact with a UniGuide schema tables/set tables:

- URL that contains the UniGuide meta tag
- Table/set table name involved
- List of Column Names
- List of Column Values
- Last Modified timestamp value, compliant to the standard ILLUASTRA timestamp format: \text{YYYY-MM-DD HH:MM:SS.9999}

The core method of the JDBC_MetaDataSaver is the Set_SQL_Statement method which invoke the proper SQL3 Generator module. This method generates SQL3 DML Statements that are executed in two phases as depicted in figure 4.11 (Phases in the execution of a DML/SELECT statement in UniGuide):

- **Phase I:** Determine the type of construct being processed (row/set instance), which in turn calls the required SQL3 Generator function by executing a RETURN statement that generates a DML Insert/Update SQL statement. For row instances the GF_robot_simple_insert module is called. For set instances the GF_robot_complex_insert is called.
• **Phase II**: the previously generated SQL DML statement is executed in the case of an Insert/Update statement. Otherwise the DML statement is ignored. Insert statements are generated when new entity instances are detected. Update statements are generated when a given existent entity instance has been modified, indicated by the date of last modification timestamp value, which necessarily must be newer than the one previously stored in the ILLUSTRÆ ORDB.

The next section describes in more detail the architecture of the SQL3 Generator functions called by URL_MetaData_Saver object instantiations.

### 4.5.6.4.1 UniGuide SQL3 Generator: DML Generation Statement Modules

**UniGuide Indexing Agent Version**

These DML Generation Statement functions are called by JDBC_MetaData_Saver object instances, which in turn are generated by URL_Scanner object instances. To obtain more detailed information about DML Generation Statement Modules please refer to the section 4.5.2.5.1 (*DML Generation Statement Modules*). As indicated in 4.5.2.5.1, before executing a RETURN statement that invokes the required SQL3 Generator module which generates automatically a DML statement, the activated JDBC_MetaData_Saver instance object must deduct the kind of construct being inserted. The GF_robot_simple_insert module is called for row instances, whereas the GF_robot_complex_insert module is called for set instances. Parameters are captured automatically by a URL_Scanner object instance and passed by instantiating a new JDBC_MetaData_Saver object. The list described below includes the parameters passed to the external C functions:

- Name of the table or set
• A list of attributes delimited by the '^' character and
• A list of correspondent attribute values delimited by the '^' character as well.
• Last Modified timestamp value associated to the Web page that contains UniGuide scheme meta tags

The following structured English describes the algorithm used for generating automatically DML statements for row instances (GF_robot_simple_insert):

```
BEGIN
  Convert list of Attributes and Values into Arrays
  Generate an Array of Primary Key components needed to compose the WHERE clause of a SELECT statement that tests if the Row instance Exists in the Database
  Create the WHERE clause of a SELECT statement that tests if the Row instance Exists
  Test if Row instance Exists in the database
  IF Row instance Does Not Exist THEN
    Generate a DML Insert statement
  ELSE
    BEGIN
      Get Stored Last Modified Value associated to row instance
      IF input Last Modified Value Is Older than Stored Last Modified Value
        Generate a DML Update statement
      ELSE
        Generate DML RETURN 0 statement
      END
    END
  END
  Return SQL DML statement
END
```

The **SQL3 Generator** "GF_robot_simple_insert" function generates DML Insert/Update SQL statements that are executed by a JDBC_MetaData_Saver instance object. The correspondent structure chart and source code can be found in **Appendix C. UniGuide Indexing Agent (C.1.1 DML Statement Generation Modules: Row instances).**
The structured English shown below, describes the algorithm used for generating automatically DML statements for set instances (GF_robot_complex_insert):

BEGIN
  Convert list of Attributes and Values into Arrays
  Generate an Array of Pseudo Primary Key components needed to compose the Interior WHERE clause of a SELECT statement that tests if the Set instance Exists in the Database
  Create Interior WHERE clause of a SELECT statement that tests if the Set instance Exists
  Create Exterior WHERE clause used both by a SELECT statement that tests if the Parent Row that holds the Set and the DML statement that assigns the Set instance to a Parent Row
  Get the Name of the Parent Table that holds the Set
  Test if the Parent Table Row instance Exists
  IF Parent Row does not Exist THEN
    BEGIN
      Insert Parent Row with Primary Key component values and other default values
      Generate a DML Insert statement for a Set instance
    END
  ELSE
    BEGIN
      Test if Set instance Exists
      IF Set instance Exists THEN
        BEGIN
          Get Last Modified Column associated to set table
          Get Stored Last Modified Value associated to set instance
          IF input Last Modified Value is Older Than Stored Last Modified Value
            Generate DML Update statement for Set instance
          ELSE
            Generate DML Return 0 statement
          END
        END
      ELSE
        Generate a DML Insert statement for Set instance
      END
    END
  END
RETURN SQL DML statement for Set instance
END
The structure chart and source code associated to the SQL3 Generator GF_robot_complex_insert module can be found in the Appendix C. UniGuide Indexing Agent (C.1.2 DML Statement Generation Modules: Set instances)

4.5.6.5 JDBC_URL_Saver

Class is used to save visited URLs and URLs to visit in the URL_catalogue and URL_to_visit tables respectively, therefore implements the JDBC interface to these tables. The app_Index_Robot_HD main thread instantiates by first time a JDBC_URL_Saver object which is initialized by executing the init method. This method which is executed only once per indexing session, detects and drops previous table contents and creates again the URL_catalogue and URL_to_visit tables by executing the required SQL DDL statements. The Save_URL method saves a given URL in both the URL_catalogue and the URL_to_visit tables. The Get_URL method retrieves the next URL to scan (oldest one) from the URL_to_visit table and deletes the row from the table at the same time. **What is remarkable is how these two last operations are executed in one single transaction, taking real advantage of the features offered by an ORDBMS such as ILLUSTR.A.** Firstly, we obtain the oldest URL to visit by simply selecting the row that has the smallest lexicographically OID. In fact OIDs in ILLUSTR.A are generated in sequential order and have the following format:

```
9999.HxHxHxHx:
```

The 9 digit represents a decimal number; the Hx string represents a hexadecimal number.
Secondly, **ILLUSTRA allows the amazing execution of hybrid SELECT/DML SQL statements** such as:

```
SELECT column, (DELETE FROM the_table WHERE cond1)
FROM the_table WHERE cond1
```

The use of this type SQL statement can be considered as an advantage for a multi-user, multi-transaction environment, where simultaneous transactions are being executed at the same time.

**4.5.6.6 ThreadLimit**

Class `ThreadLimit` is used to control the number of maximum concurrent active URL_Scanner threads. Two important methods are defined in the class, which both control the number of active URL_Scanner threads and it is entirely based on the classic operating system concept of semaphore. The `Get_Permit` method tests the current number of active URL_Scanner threads. When the maximum number of permitted URL_Scanner threads is not reached, the current active thread is allowed to continue proceeding, otherwise it will have to wait until notified by other URL_Scanner threads that are about to finish. The `Set_Permit` method is executed by terminating URL_Scanner threads that notify to waiting threads that it is about to finish, so others can continue executing.

**4.5.6.7 Stop_URL_Scanner**

Class `Stop_URL_Scanner` is used mainly to define a resident daemon thread that stops hanged/delayed URL_Scanner threads. Stop_URL_Scanner thread is activated every 30 seconds with maximum priority in order to ensure that it will have processor time more or less exactly when awaken (pre-emptive priorities scheduling). All currently active
URL_Scanner threads are tested, calculating the total number of milliseconds since they were started. If the calculated time exceeds the maximum permitted time, the URL_Scanner thread is stopped (and garbage collected) and the daemon will give permit to other URL_Scanner threads in order to have processor time.

4.6 End-user perspective: User Interface Sections

Currently the *UniGuide* database search engine system can be accessed on the Web at the following URL: [http://budhi.uow.edu.au/cgi-bin/Webdriver?MIval=main_menu](http://budhi.uow.edu.au/cgi-bin/Webdriver?MIval=main_menu). This URL is the starting point to access all the different *UniGuide* sections, and therefore constitutes the main menu from which end-users may choose a given option. The following figure (fig 4.33) shows the sections available publicly in *UniGuide*.

![Figure 4.33 UniGuide Database Search Engine: Main Menu](image)
• **Submit URL:** end-users may submit manually a given target Web page into the UniGuide ORDB

• **Simple Query:** permits the definition and execution of queries that involve a single table/set table

• **FreeSQL:** this section may be suited for advanced end-users that are familiar with SQL and are capable of defining more complex queries such as summarised information, joins and so on, with the aid of prototype query examples

• **Meta Tag Generator:** this section is definitely suited for information providers, such as Web masters, and individuals that publish target Web pages on the Web. The purpose of this section consists in the generation of UniGuide scheme meta tags that can be cut and pasted into target Web pages in order to permit the UniGuide indexing agent to extract periodically the information stored in these meta tags

We must mention again that our research prototype database search engine depends heavily on the existence of standard UniGuide meta tags on target Web pages. Although end-users may insert manually target Web pages, this is not the ideal scenario. Due to the fact, that the adoption of standards is a process that requires meetings between the parts involved and therefore involves bureaucracy, this is a requirement that is out of the scope of this thesis. **Our purpose is to demonstrate that under ideal conditions our research prototype becomes a fully articulated production system.** This current situation force us to manually input data or to define a limited set of example target Web pages containing UniGuide meta tags in order to provide test data.

### 4.6.1 Submit URL Section
Entities are grouped by domain/sub-domain, graphically represented as a tree structure (table browser). Three types of icons are included in the table browser (table 4.11):

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>📚</td>
<td>UniGuide icon</td>
</tr>
<tr>
<td>🗃️</td>
<td>Domain Folder</td>
</tr>
<tr>
<td>📤</td>
<td>Form icon</td>
</tr>
</tbody>
</table>

**Table 4.11 UniGuide Table Browser icons**

- **UniGuide icon**: represents the root of the table/set table browser
- **Domain Folder**: represents a domain/sub-domain that holds child entity form icons
- **Form icon**: represents an entity type that may refer internally to a table or set table

When end-users click on a folder icon, the tree structure hides or expands. When end-users click on a form icon, a dynamically generated HTML form will appear on the right-frame.

End-users can input data on intelligent forms, that provide contextual help available on the status-bar and with the ability to validate data input at the client-side. When the form is submitted, the length of data, the syntax of expressions (i.e. URL definition), the existence of data in not-nullable columns will be tested on the client-side. If data input is correct, this is sent to the server, stored in the UniGuide ORDB and finally an alert box appears on the screen indicating the status of the transaction. We can distinguish three cases:
- A successful Insert transaction
• A successful Update transaction

• An error message indicating the error code and error message. This case rarely occurs as data input is thoroughly tested on the client-side previous to URL submissions.

The following figure (fig 4.34) shows the GUI associated to the URL submit section:

![Figure 4.34 UniGuide Submit URL Section](image)

The table browser may allow the submission of any entity instance type, except university entity instances. As a matter of fact, we input manually university entity instances as they reference to the main University homepages and therefore represent the starting point of any logical university domain instance.
4.6.2 Simple Queries Section

A table browser that is exactly the same as the one described in the previous subsections is included in order to permit end-users to choose the required entity type to be queried. End-users can specify the list of column to be included in the output of a given query, the required search options and search values. Search options are contextual depending of the type of data in the column (i.e. LIKE option is activated only for columns of type text). Search values may be specified with the aid of select lists. An asterisk indicates that all values are accepted. In addition the "%" char is precised for search values that have a LIKE search option. The SELECT statement generator will parse the query and generate automatically a WHERE clause that will have the form of either "LIKE '%string_pattern' or "LIKE 'string_pattern%' or 'LIKE '%string_pattern%''.

The output of a query is displayed in a tabular form. Other information displayed includes the SQL query generated. A maximum of 25 rows is displayed, giving the possibility to end-users to browse query results back and forth. This kind of user interface can provide even to casual users a rapid overview of the UniGuide schema to which she/he can get accustomed easily.

The figure depicted below (fig. 4.35), shows an example of simple query formulation. In this case an end-user intends to retrieve all the Web pages that refer to sports clubs or associations at the University of Technology, Sydney:
Figure 4.35 *UniGuide* Simple Query Section: Query Formulation

Figure 4.36 *UniGuide* Simple Query Section: Query Results
The SQL SELECT statement generated and results displayed in tabular format are returned to end-users (see fig 4.36).

4.6.3 Free-SQL Section

The FreeSQL form is shown in Figure 4.37. End-users can configure and customise the required query. A more complex interface is provided for advanced end-users, that allows the definition of more complex SQL queries with the aid of predefined queries, functions, operators, and a list of tables and columns available in the schema. End-users may specify standard SQL92 queries or non-standard SQL3 queries. For those that are not familiar with SQL3, prototype query examples can be selected from the Sample Queries select list. This select list includes a set of examples stored in the query_samples table. Currently the query_samples table stores examples of case insensitive queries, aggregate query (summarised information), simulation of transitive closure (i.e. retrieval of all child academic entities within a given faculty), joins between tables (i.e. research entities associated to a given university), joins between parent table and dependent set table (i.e. retrieve courses associated to a given academic entity). In addition, end-users can click on the Replace button in order to replace every occurrence of a given word, so queries can be customised more easily.

This section is well-suited for individuals or university departments that try to extract simple statistics from the University logical domain, specially Web masters. The following figure (fig. 4.37) shows an end-user formulating a simple query that retrieves information about universities:
Figure 4.37 UniGuide FreeSQL Section: Query Formulation

Figure 4.38 UniGuide FreeSQL Section: Query Results
Figure 4.38 shows the results retrieved from executing the query defined above (fig. 4.37). Results are displayed in tabular format in blocks of 25. End-users can retrieve more row/set instances by clicking on a dynamically generated parameterised link (Next, Prev).

4.6.4 Meta Tag Generator Section

The UniGuide schema table browser described in the previous sub-sections allows end-users to choose the required entity type to generate the associated UniGuide scheme compliant meta tag. When end-users click on the Generate button, data input is tested. This process requires at least the definition of mandatory columns, normally components of the primary key or pseudo primary key (set instances). On a successful attempt, a complete UniGuide scheme meta tag is generated and displayed on the Meta tag textarea, which can be cut and pasted into the contents of target Web pages. Figure 4.39 shows an end-user defining a meta tag that represents a club_association entity.

Figure 4.39 UniGuide Meta Tag Generator Section: Data Input
The figure depicted below (fig. 4.40) demonstrates the successful execution of the meta tag generation operation.

![Figure 4.40 UniGuide Meta Tag Generator Section: Data Output](image)

**Figure 4.40** *UniGuide* Meta Tag Generator Section: Data Output

### 4.6.5 Query Taxonomy: examples of possible queries

We distinguish between intra-site queries (queries which involve a particular university) and inter-site queries (queries that span multiple universities, e.g. all universities in the state of New South Wales) (*Enguix et. al 1998b*). Also, we can classify queries on a continuum from simple to complex. We have identified the following query types (see table 4.12): simple queries (one entity/set involved), summarised information (aggregate functions: average, sum, count, etc.), joins (more than one entity/set involved) and simulation of transitive closure (over unary relations). Examples of queries in each cell are presented below.
Table 4.12 Query Taxonomy

Examples of simple queries:

- **Intra-site**
  
  Q1: Find all the home pages of university residential colleges of a particular university that offer full board and are coed.

  **Classes/tables:** residential_college

- **Inter-site**
  
  Q2: Find all the relevant information of Information Systems departments located in the state of NSW.

  **Classes/tables:** academic_entity

Examples summarised information:

- **Intra-site**
  
  Q3: Find the number of research entities associated with a particular University

  **Classes/tables:** university, research_entity

- **Inter-site**
  
  Q4: Find the number of publications printed in 1998 and related to databases by university for all the universities.

  **Classes/tables:** university, academic_entity, research_entity, _academic_entity_spublications, _research_entity_spublications
Examples joins:

- **Intra-site**

  Q5: Find the e-mail addresses of all staff members in the Computer Science department of a particular university who are interested in Web-related research.

    Classes/tables: `academic_entity, _academic_entity_staff`

- **Inter-site**

  Q6: Find the URLs of academic entities and research entities involved in a particular project funded by a particular agency and started in the current academic year.

    Classes/tables: `academic_entity, research_entity, _academic_entity_sprojects, _research_entity_sprojects`

Example simulation of transitive closure:

- **Intra-site**

  Q7: Find and retrieve all the relevant information of all subordinate academic entities of the Faculty of Commerce of a particular university.

    Classes/tables: `academic_entity`

- **Inter-site**

    Not applicable in this case.

4.7 Summary

The use of a hybrid object-relational model is justified in some reasonable way in this chapter. An object-relational model permits us to choose the best features available in both technologies. We indicated the type of object-relational constructs we can define in
ILLUSTRA. We described thoroughly the main components of the *UniGuide* architecture, which is based on the integration of server and client activated components. In order to give a general idea of what information is being stored in the *UniGuide* ORDB, we outlined the type of attributes being stored. We also discussed the importance of the *Auxiliary Catalogue System* which combined with the *UniGuide SQL3 Generator*, gives us the possibility of generating automatically complex SQL3 statements. In *UniGuide*, we make intensive use of the rules subsystem in order to control data integrity constraints, user-defined referential integrity, to define pseudo primary keys in set tables and other miscellaneous tasks.

A new graphical tool which is capable of presenting in an intuitive way the complexity of integrating Web-based components in tightly integrated operational systems has been presented in this chapter. Additionally we explained the type of meta data that should be attached to target Web pages in order to permit the *UniGuide indexing agent* capture this information.

Finally, we concluded with an overview of the various applications available in *UniGuide* from an end-user perspective. Section 4.6.5 (Query Taxonomy: examples of possible queries) can give a general idea to the reader the type of queries one can formulate, and obviously this type of queries are definitely out of the scope of keyword-based search engines.
Chapter 5.
Generalised Methodological Guidelines

5.1 Introduction
5.2 Methodology
5.3 Summary
Chapter 5. Generalised Methodological Guidelines

5.1 Introduction

One of the criteria proposed by Nunamaker et al. (1991) research methodology is the generalisation of the experience and design expertise gained from building the research prototype, for an innovative system development work to conform to a valid research method.

Following an inductive reasoning, if we present the following assertions:

1. In our research we have identified a single well-demarcated logical domain such as the university domain

2. Logical domains are characterised by high regularity in their content and organisation, normally integrating information providers that belong to the same activity sector, which enables the development of ontologies (i.e. Software companies, health care sector, etc.)

3. The WWW is expanding exponentially and has been converted in the Information Universe, where all information providers desire to be present

Then we can justify in a consistent form that:

There are likely to be other logical domains with similar characteristics to be discovered on the WWW.

The most problematic issue in the implementation of database search engines is the enforcement of meta data standards. For logical domains that involve independent
information providers, this phase requires a consensus between the parties involved. On the other hand, for large corporate intranets owned by single companies, or groups, the enforcement of meta data standards may constitute a feasible option.

5.2 Methodology

Taking into account the experience gained from the development of the current prototype, we list a generalised sequence of activities involved in the creation of domain-specific database search engines:

1. **Identification of logical domains on the WWW**: as mentioned above, we can identify and isolate logical domains associated to information providers that belong to the same activity sector. Additionally large intranets that present highly structured Web sites, can be considered as potential targets.

2. **Schema construction**: we must identify the core entities represented in the contents of Web pages and the logical relationships between them. This process is guided by the definition of ontologies associated to the domain and requires a considerable effort. Web developers must analyse thoroughly the contents of different Web pages on different Web sites. The modelling process must be evolutionary, beginning with a mock-up model that is successively refined. A modelling process must be taken into account as what it is, an abstraction of what is present in the real world. Unfortunately the real world presents more complexity than the desired. In other words, one should realise that it is not feasible to represent any possible single entity, or to represent all possible variations of a given entity. Our goal consists of creating a compact model but rich model that represents the entities captured in the analysis of Web page contents.
3. **Model mapping**: map the model into a relational, object-oriented or hybrid object-relational database. Any of the technologies mentioned above can be used, nevertheless we advocate the use of object-relational technology as ideal for this type of environment.

4. **Meta data design**: design customised meta-tags that represent entity-instances for attaching these meta-tags to target web pages. An ideal format for designing meta tags, is the typical list of attribute/value pairs.

5. **Deployment of a Web interface**: as in any search engine, although ideally one should avoid basing the interface to the database on cgi scripts, which require a respectable effort. For JAVATM enabled browsers, the optimal solution would be the deployment of applets that interact with the database via a JDBC driver. Considering that JAVATM is platform independent, this solution constitutes the most generic one.

6. **Deployment of a Meta tag Generator**: develop a customised meta-tag generator that generates meta-tags compliant with the standard designed in step 5 and that is publicly available on the WWW.

7. **Indexing Agent customisation**: develop from scratch or adapt a public domain indexing agent that can parse and extract the required meta-tags from target Web pages. The main goal of the indexing agent is to detect and store meta data associated to target Web pages, therefore discarding all the rest of the contents. Nevertheless, the indexing agent must fetch all the links referenced by a given Web page in order to navigate them subsequently. The ideal solution is to execute the indexing agent on a powerful workstation with at least 128 MB of RAM, allowing us to store URLs on memory, so only databases are accessed when meta data is
detected. Following this strategy, the indexing process can be accelerated drastically and so the indexing cycle (revisit of a Web site).

5.3 Summary

In this chapter we have presented a methodology that can be used in the development of database search engines to query Web data. It is our expectation that this can lead to the conversion of this proposed methodology into a feasible production system standard, suited especially for large intranets where meta data standards can be enforced and potentially for sector related information providers such the one proposed in this research, the logical domain of Australian Universities.
Chapter 6.
Contributions, Implications, Limitations and Future Research

6.1 Introduction
6.2 Research contributions
6.3 Implications
6.4 Limitations
6.5 Future Research
6.6 Summary
Chapter 6. Contributions, Implications, Limitations and Future Research

6.1 Introduction

We begin the chapter by justifying in some way the knowledge contributed in this thesis to the research domain. Further, we indicate the communities that may benefit from the output of the research, which may include Web developers, academic/research communities, university departments and so on. In the next section we differentiate two types of limitations associated in the development of this research prototype, conceptual and technical limitations. What follows next includes all the desired features we would like to incorporate to the database search engine in the near future. Finally, we conclude with a summary of the chapter.

6.2 Research Contributions

We have advocated in this research a strategy of "divide & conquer" to the Information Universe, the WWW. As the Web expands exponentially, more challenging becomes the resource discovery problem. Generic keyword-based search engines, could present at certain point, a limited capacity of retrieving useful information, unless the technology is improved or backed up with other components. If Internet connections were improved, and browsers could include temporary databases, probably a good strategy for keyword-based search engines would be to send big "chunks" of query results to the client browser that are reorganised and restructured in local temporary tables, providing a more intelligent search over query results. On the other hand, database search engines permit to end-users to define focused and structured queries.
We believe that the construction of the database search engine prototype constitutes an innovative way of restructuring and organising Web information. End-users are enabled to define intelligent queries capable of retrieving useful information. Furthermore, our contribution is expanded with a list of methodological guidelines required to build this kind of database search engines, which indicates that the methodology can be applied to different domains, and that is suited specially for intranets. In addition, the implementation of this research prototype is largely based on the integration of diverse tools, technologies and languages such as the ILLUSTRATA ORDBMS, external C functions, Web DataBlade applications and so on. The graphical representation of the structure of the design and implementation of the prototype posed a challenge because of the non-availability of suitable tools or methods to guide our work. We have developed a new graphical model such as WebApp Structure Charts described in chapter 4 which should be of interest to Web application developers.

6.3 Implications

The research proposed in this thesis might be of interest and benefit to a wide range of communities, such as academic or researchers, developers and other end-users.

6.3.1 Academic/Research Community

Firstly we will focus on the use of the prototype itself and the possible implications over the academic/research community. Database search engines will foster the optimal sharing of Web information between the research community. We might include as an example, *UniGuide* converted in a fully articulated production system that permit
researchers to be able to find technical reports or articles or information about research projects with great recall and precision.

Secondly, we will focus on the use of the methodology to discover new logical domains. In section 5.1 (Methodological Guidelines: Introduction), we justified in a consistent way the existence of more logical domains on the WWW to be discovered. Now it is the turn of specific academic or research communities to analyse their Web servers in order to deduce the existence of high regularity in content and organisation. In fact, if one analyses for example Computer Science department Web servers, the majority present a very similar organisation, including information about projects, staff members, publications, technical reports and so on. Basing ourselves on the concept of ontologies, there is a specific common vocabulary for specific academic and research communities. Homonyms and synonyms must be detected in order to provide a common standard vocabulary to describe entity instances associated to the underlying ontology. We may cite as an example the development of the Chemical Markup Language (CML), which has been specifically developed for the Chemical sector. This kind of initiatives may influence as well in the development of specific ontologies, meta data and database search engines adapted to specific communities, such as the Chemical sector.

6.3.2 Developers

Web developers, especially those dedicated in the deployment of large Intranets might benefit from the methodological guidelines proposed in chapter 5 (Methodological Guidelines). Considering that meta data standards can be enforced in this type of environment, we presume that this should be the target environment to produce a fully operational system.
Another aspect that we must investigate, is the graphical model proposed in this research used to model the integration of Web-based components (WebApp Structure Chart) sufficiently generic to be applied to any kind of Web development project.

6.3.3 Other End-users

Web Masters, university departments, government departments such as the Department of Employment, Education, Training, and Youth affairs (DEETYA), might benefit considerably:

- **Web masters**: a database search engine such as *UniGuide* permits the generation of simple statistics such as summarised information, feature not available in any keyword-based search engine. Further, Web masters may proceed with maintenance tasks by using the last modified timestamp value associated to any entity instance. For example, a Web master might be able to find out in a large Intranet that involves various Web servers, the URLs of Web pages that have not been updated since a given date.

- **University Departments**: might rely more on getting directly information on the Web about their own departments, publish even more information on the Web due to the fact that end-users can retrieve information with great precision and potentially provide statistics for internal use.

- **Government departments**: such as the DEETYA might promote this type of initiatives in their own benefit, allowing them to obtain statistics about the tertiary sector in a much faster way. The more information is pushed by information providers, the more accurate these statistics can be. As an example, if all the
departments of a given University have their respective Web servers or sub-Web hierarchy, and all the home pages of those departments contain internally the required meta data formats, then we can say that a query that must retrieve the number of departments of a given University will provide the correct answer.

- **End-users:** in this case we must constrain ourselves to the use of *UniGuide* under the end-user perspective. An intuitive GUI such as the one provided in the different *UniGuide* sections, is suitable even for casual users (except the FreeSQL section). End users can define intelligent queries with great precision. As a result, both end-users and information providers are benefited by facilitating a partial solution to the resource discovery problem. Partial in the sense, that a database search engine cannot cover all the information stored on a given domain.

- **General information providers:** the fact that a great precision can be obtained in this type of environment, it may encourage information providers to push even more information on the Web following the required meta data standards.

### 6.4 Limitations

As in any research, there are conceptual limitations. The fact that our model is schema-based, and that a model is an abstraction of the real world, we only represent the core entities on a given domain. End-users may use alternative technologies such as keyword-based search engines to find more rare or uncommon information. Another limitation is that the success of our research prototype depends entirely on the promotion of the *UniGuide* meta data format in the University domain. The enforcement of meta data standards in such domain requires a respectable effort, considering the hypothesis that the parties involved agree to adopt such standard. Obviously this fact is out of the scope of this thesis. *An ideal alternative solution*
would be the development of intelligent indexing agents that are capable of extracting automatically Web information guided by pre-built ontologies. These guided intelligent indexing agents pose a serious technological challenge due to the fact that we are dealing with complex ontologies.

Additionally, we shall list some of the technical limitations identified in the development of the research prototype. We can mention for instance that database accesses are much slower on set instances than row instances, as ILLUWikTRA does not allow us to define B-TREE indexes on set table attributes (except on OID attributes). This is the dark side of using sets in ILLUWikTRA. Another limitation is the double daemon architecture required for interacting with ILLUWikTRA databases via a JDBC driver. The system accepts at the most three simultaneous database transactions, which limits drastically the number of permitted simultaneous threads executed by the UniGuide Indexing Agent (hard drive version).

6.5 Future Research

The ultimate goal of a system developer is to build more robust systems that are capable of reacting well in any kind of situation. Experience can teach us that it is almost impossible to develop prototype systems that are perfect at earlier stages. Perfection can only be obtained via the typical life cycle of an evolutionary project. In other words, systems may be continuously improved and refined in order to obtain a more acceptable and usable system. The UniGuide database search engine prototype requires the existence of meta data in the University logical domain to become a mature system. Nevertheless, if we immerse ourselves in the ideal world where the UniGuide system becomes a production system, still there would be a series of additional steps to reach
the required perfection. Such tasks include: survey of acceptance of *UniGuide* meta data standard, evaluation of the system, fuzzy referential integrity, follow the robot exclusion standard and so on. We describe below some of the future improvements we would like to incorporate in the *UniGuide* architecture:

- **Survey of acceptance of the UniGuide meta data standard:** a form that can be made available on the WWW, will give us the chance to send e-mails to all Web masters of an initial limited set of universities. Furthermore, we must contact with high-level university personnel in order to encourage them to promote such initiative. Data will be collected in relational tables when end-users submit information to our Web server, and analysed afterwards.

- **Evaluation of the system:** an initial evaluation must be performed with current manual data in order to deduct the useability of the system. All the required procedures are explained in detail in section 3.5 (Evaluation Methodology) and include the analysis of Web log files, use of audit rules, cookies, etc.

- **Fuzzy referential integrity:** currently the referential integrity built in *UniGuide* is rules-based and permits the definition of case insensitive referential integrity. As mentioned frequently in other section the information stored on the Web is characterised to be ambiguous or non-exact. An example could be when and end-user specifies manually the identifier of a University such as "Sydney University" and the identifier has been previously stored as "University of Sydney", our current implementation will reject such transaction because it violates the constraint. If we implement an algorithm similar to the agrep tool provided in the Harvest system *(Bowman et al. 1995)*, which defines approximate pattern matching, and associate this algorithm to a rules-based referential integrity constraint, our system becomes
more tolerant to non-exact or ambiguous values. Such an algorithm does not constitute any serious challenge and the most simple implementation may consist on tokenizing both strings (the stored one and the captured one), removing all stop words from the string (i.e. remove "of" from the "University of Sydney"), sort the arrays of tokens in alphabetical order and compare the resulting arrays in order to find if they semantically refer to the same concept.

- **Follow the robot exclusion standard**: ideally, before beginning the indexing process, the *UniGuide* indexing robot should read the robots.txt file normally situated in the root directory of a given server, transform the contents of the text into a table of sub-Webs that the indexing robot is not allowed to navigate. In the case that *UniGuide* becomes fully operational, it is our responsibility to follow such standard.

- **Tune the indexing robot**: we have been observing the behaviour of the memory version, which can be considered as the ideal solution. Unfortunately, the most stable version is the hard drive version, which relies on more database accesses. We must study more thoroughly the cause of instability in the memory version. We have tested this version up to a total of 100.000 URLs processed, whereas in other indexing processes at the most 1.000 URLs were processed.

- **Transfer our methodology to the real world**: contact a corporate intranet in order to serve as an experimental base for applying the same methodology imposed in this research.

### 6.6 Summary
Some of the future research directions we intend to pursue what communities are the most potential users of this type of technology and methodology. The ideal target community should include Intranet and Web developers requiring the enforcement of metadata standards. The deployment of schema-based database search engines enables the definition of a new kind of resource discovery over Web data: the transformation of well-demarcated sub-Webs on the WWW into highly structured databases.
### 7. References

<table>
<thead>
<tr>
<th>Reference</th>
<th>Title and Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Callery 1997</td>
<td></td>
</tr>
</tbody>
</table>
http://www.library.ucsb.edu/untangle/callery.html

Cerf 1998
Cerf, V “A Brief History of the Internet and Related Networks”, Internet Society (ISOC, May 1998
http://www.isoc.org/internet-history/cerf.html

Cho et al. 1998

De Bra and Post 1994
http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching.html

Dreilinge and Howe 1997

Enguix et al. 1998a

Enguix et al. 1998b
http://budhi.uow.edu.au/postgrad/carlos/tr98_5_101.htm

Francis et al. 1995
http://www.w3.org/Conferences/WWW4/Papers/300/

Han et al. 1995

**Hartmanis 1995**

**Heery 1996**

**Ianella and Waugh 1997**

**ILLUSTRA 1996**

**Isakowitz et al. 1995**

**Kahle 1996**

**Kalvelagen 1997**

**Koch 1996**

**Koster 94a**

**Koster 94b**
Leiner et al. 1998
http://www.isoc.org/internet-history/brief.html

Maciaszek 1997
Maciaszek, A. L., “Relational versus Object Databases Contention or Coexistence?”, Computing Department School of Mathematics, Physics, Computing and Electronics Macquarie University - Sydney, Australia.

Manber et al. 1997

Miller and Iannella 1998
Miller, E. and Iannella, R., “Dublin Core Examples in RDF”, DSTC Resource Discovery Unit, 1998

Nunamaker et al. 1991

Pinkerton 1994
http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/pinkerton/WebCrawler.html

Poulter 1997

Rajaraman 1998
Rajaraman, A., "Transforming the Internet into a Database", Workshop on Reuse of Web information hold in conjunction with the 7th International World Wide Web Conference (WWW7), Brisbane, April 14,1998

Schwabe and Rossi 1998
### References


**Schwartz et al. 1992**  

**Stonebraker and Moore 1996**  

**Sun 1998**  
Sun Microsystems, "JAVADOC Homepage", 1998-08-04.  

**W3C 1998**  
http://www.w3.org/TR/PR-rdf-syntax/

**WebSQL Homepage 1997**  
http://www.cs.toronto.edu/~websql/lang-ref.html

**Yuwono and Lee 1996**  
Appendix A. UniGuide SQL3 Generator Library: Source Code

A.1 DML Statement Generation Modules
A.1.1 DML Statement Generation Modules: Row instances
A.1.2 DML Statement Generation Modules: Set instances

A.2 Query Statement Generation Modules
A.2.1 Query Statement Generation Modules: Row instances
A.2.2 Query Statement Generation Modules: Set instances

A.3 Query Filtering Modules

A.1 DML Statement Generation Modules:
A.1.1 DML Statement Generation Modules: Row instances
Structure Chart GF_simple_insert.c:

![Structure Chart GF_simple_insert.c](image)

Figure A1. Structure Chart GF_simple_insert.c
Source Code GF_simple_insert.c:

/* GF_simple_insert.c */

#include "stdio.h"
#include "string.h"
#include "mi.h"

/* prototypes */

mi_integer get_pk_cols(MI_CONNECTION *conn, /* connection to default database */
                      mi_char1 *strtbl,  /* name of table */
                      mi_char1 *astrpkcols[15]);  /* array of cols. components PK */

mi_integer test_row_exists(MI_CONNECTION *conn, /* connection to db */
                           mi_char1 *strtbl, /* name of table */
                           mi_char1 *astrwhere);  /* where SQL statement */

mi_char1 *get_where_clause(mi_char1 *strtbl,  /* name of table */
                           mi_char1 *astrflds[15], /* array of columns */
                           mi_char1 *astrvalues[15], /* array of values */
                           mi_char1 *astrpkcols[15]);  /* array of comp. of pk */

mi_char1 *get_update_statement(mi_char1 *strtbl, /* name of table */
                                mi_char1 *astrflds[15], /* array of columns */
                                mi_char1 *astrvalues[15], /* array of values */
                                mi_char1 *strwhere);  /* where SQL statement */

mi_char1 *get_insert_statement(mi_char1 *strtbl, /* name of table */
                                mi_char1 *astrflds[15], /* array of columns */
                                mi_char1 *astrvalues[15]);  /* array of values */

mi_char1 *change_to_type(mi_char1 *strvalue);  /* value to change */

void get_words(mi_char1 *straux, /* string delimited by "^" */
               mi_char1 *astraux[15]);  /* array of strings */

/* end of prototypes */

mi_text *GF_simple_insert(mi_text *strtblarg,mi_text *strfldsarg,mi_text *strvaluesarg)
{
    MI_CONNECTION *conn;  /* conn. to default database */
    mi_char1 *strsql;  /* SQL statement returned */
    mi_char1 *astrwhere;  /* WHERE clause */
    mi_char1 *strtbl;  /* name of table */
    mi_char1 *strflds;  /* list of columns */
    mi_char1 *astrvalues;  /* list of values */
    mi_char1 *astrfields[15];  /* array of columns */
    mi_char1 *astrvalues[15];  /* array of values */
    mi_char1 *astrpkcols[15];  /* array of cols. comp. PK */
    mi_integer inti, int_row_exists, interror;

    /* allocate memory dynamically */
    strtbl    = mi_alloc(sizeof(mi_char1) * 50);
    strflds   = mi_alloc(sizeof(mi_char1) * 300);
    strvalues = mi_alloc(sizeof(mi_char1) * 1000);
    strsql    = mi_alloc(sizeof(mi_char1) * 1500);
    strwhere  = mi_alloc(sizeof(mi_char1) * 500);

    /* allocate memory dynamically to array of chars */
    for(inti=0;inti<15;inti++)
    {
        astrfields[inti]    = mi_alloc(sizeof(mi_char1) * 50);
        astrvalues[inti]    = mi_alloc(sizeof(mi_char1) * 300);
        astrpkcols[inti]    = mi_alloc(sizeof(mi_char1) * 50);
    }

    /* convert text types into strings */
    mi_text_to_buffer(strtblarg,strtbl);
    mi_text_to_buffer(strfldsarg,strflds);
    mi_text_to_buffer(strvaluesarg,strvalues);

    /* opens current opened database */
conn=mi_open(NULL,NULL,NULL);

/* generate array of columns to insert values*/
get_words(strflds,astrflds);

/* generate array of values associated to each column */
get_words(strvalues,astrvalues);

/* generate array of cols that are components of the key */
interror = get_pk_cols(conn,strtbl,astrpkcols);
if(interror == 0) {
    strcpy(strsql,"Return 0;"),
    return mi_string_to_text(strsql);
}
else
    /* generate WHERE clause of SQL statement */
    strcpy(strwhere,get_where_clause(strtbl,astrflds,astrvalues,astrpkcols));

/* generate SQL query to test if row exists in the table */
int_row_exists = test_row_exists(conn,strtbl,strwhere);
if(int_row_exists == 1) /* row exists, get update SQL statement */
    strsql = get_update_statement(strtbl,astrflds,astrvalues,strwhere);
else
    /* Row does not exist, it will insert new row with current timestamp */
    strsql = get_insert_statement(strtbl,astrflds,astrvalues);

/* close the connection to the default database */
mi_close(conn);

/* deallocate memory */
mi_free(strtbl);
mi_free(strflds);
mi_free(strvalues);
mi_free(strwhere);

/* deallocate memory to array of chars */
for(inti=0;inti<15;inti++) {
    mi_free(astrflds[inti]);
    mi_free(astrvalues[inti]);
    mi_free(astrpkcols[inti]);
}

/* convert into text type again */
return mi_string_to_text(strsql);

mi_integer get_pk_cols(MI_CONNECTION *conn, /* connection to default database */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *astrpkcols[15]) /* array of cols. components PK */
{
    /* Returns an array of columns that are component of the Primary key */
    MI_ROW *row;
    mi_char1 *strcolvalue, *strsql;
    mi_integer intlen,interr,intresult,
    inti;
    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 500);
    /* generate SQL command */
    /* get the columns that are components of the primary key */
    strcpy(strsql,"SELECT c.column_source FROM _columns c"),
    strcat(strsql," WHERE c.table_source='"),
    strcat(strsql,strtbl),
    strcat(strsql,"' AND c.column_pk = 1;"),
    /*execute query */
    mi_exec(conn,strsql,0);
    inti = 0;
    while((intresult = mi_get_result(conn))!=MI_NO_MORE_RESULTS) {
        switch(intresult)
        {
        case MI_ERROR:
return 0;
case MI_ROWS:
  /* for each row, get only one column */
  while ((row=mi_next_row(conn,&interr)) != NULL)
  {
    /* get the value of the single column */
    switch (mi_value(row,0,&strcolvalue,&intlen))
    {
      case MI_ERROR:
        return 0;
      case MI_NULL_VALUE:
        return 0;
      case MI_NORMAL_VALUE:
        strcpy(astrpkcols[inti],strcolvalue);
        inti++;
        break;
      default: return 0;
    }
  }
  break;
}
/* deallocate memory */
mi_free(strsql);
return 1;

mi_integer test_row_exists(MI_CONNECTION *conn, /* connection to db */
                         mi_char1 *strtbl, /* name of table */
                         mi_char1 *strwhere) /* WHERE clause */
{
  /* Checks if a given row instance exists receiving as input a WHERE */
  /* clause that involves components of the primary key */
  mi_integer    inti, intj;
  mi_char1      *strnumrows, *strsql;

  /* allocate memory dynamically */
  strsql  = mi_alloc(sizeof(mi_char1) * 500);
  strcpy(strsql,"SELECT count(*) FROM ");
  strcat(strsql,strtbl);
  strcat(strsql," WHERE ");
  strcat(strsql,strwhere);
  strcat(strsql,";");

  /*execute query */
  strnumrows = get_column_value(conn,strsql);
  /* deallocate memory */
  mi_free(strsql);

  /* return number of rows retrieved, should be 0 or 1 */
  if(!strcmp(strnumrows,"0"))
    return 0;
  else
    return 1;
}

mi_char1 *get_where_clause(mi_char1 *strtbl, /* name of table */
                           mi_char1 *astrfids[15], /* array of columns */
                           mi_char1 *astrvalues[15], /* array of values */
                           mi_char1 *astrpkcols[15]) /* array of comp. of pk */
{
  /* Generates WHERE clause with components of the primary key */
  /* NOTE: all components of the PK are to date of type text that is why we can */
  /* convert into uppercase, if it is a number then we must cast to text ::text */
  mi_integer    inti, intj;
  mi_char1      *strvaluetype, *strwhere, *straux;

  /* allocate memory dynamically */
  strwhere = mi_alloc(sizeof(mi_char1) * 500);
strvaluetype = mi_alloc(sizeof(mi_char1) * 350);
strcpy(strwhere,"");
for(inti=0;inti<15;inti++) {
    if (!astrpkcols[inti][0])
        break;
    if(inti > 0)
        strcat(strwhere," AND ");
    straux=strstr(astrpkcols[inti],"url");
    if (straux!=NULL)
        strcat(strwhere,astrpkcols[inti]);
    else {
        strcat(strwhere,"upper(");
        strcat(strwhere,astrpkcols[inti]);
        strcat(strwhere,")");
    }
    strcat(strwhere,"=");
    for(intj=0;intj<15;intj++) {
        if(!astrflds[intj][0])
            break;
        if(!strcmp(astrflds[intj],astrpkcols[inti])) {
            strvaluetype = change_to_type(astrvalues[intj]);
            if (straux!=NULL)
                strcat(strwhere,strvaluetype);
            else {
                strcat(strwhere,"upper(");
                strcat(strwhere,strvaluetype);
                strcat(strwhere,")");
            }
            break;
        }
    }
}
/* deallocate memory */
mi_free(strvaluetype);
/* return WHERE clause of SQL statement */
return strwhere;

mi_char1 *get_update_statement(mi_char1 *strtbl,        /* name of table        */
                                mi_char1 *astrflds[15],         /* array of columns       */
                                mi_char1 *astrvalues[15],       /* array of values         */
                                mi_char1 *strwhere)             /* WHERE clause         */{
    mi_char1 *strset, *strvaluetype, *strsql;
    mi_integer inti;
    /* allocate memory dynamically */
    strsql  = mi_alloc(sizeof(mi_char1) * 1500);
    strset  = mi_alloc(sizeof(mi_char1) * 700);
    /* generate SET clause SQL statement */
    strcpy(strset," SET ");
    for(inti=0;inti<15;inti++) {
        if(!astrvalues[inti][0])
            break;
        if(inti>=1)
            strcat(strset," ,");
        strcat(strset,astrflds[inti]);
        strcat(strset," = ");
        if(!strcmp(astrvalues[inti],"null"))
            strcat(strset, "null");
        else {
            strvaluetype=change_to_type(astrvalues[inti]);
            strcat(strset,strvaluetype);
        }
    }

    /* generate WHERE clause of SQL statement */
    strcat(strsql,"WHERE ");
    strcat(strsql,strset);
    strcat(strsql,strtbl);
strcat(strsql,strset);
strcat(strsql,"",last_modified=current_timestamp");
strcat(strsql," WHERE ");
strcat(strsql,strwhere);
strcat(strsql,";");

/* deallocate memory */
mi_free(strset);

/* return UPDATE SQL statement */
return strsql;
}

mi_char1 *get_insert_statement ( mi_char1 *strtbl,       /* name of table        */
                                      mi_char1 *astrflds[15],         /* array of columns     */
                                      mi_char1 *astrvalues[15])       /* array of values      */
{
    /* Generates an SQL DML Insert statement for a row instance */
    mi_char1      *strvaluetype, *strsql, *strvalues, *strflds;
    mi_integer    inti;
    /* allocate memory dynamically */
    strsql  = mi_alloc(sizeof(mi_char1) * 1500);
    strvalues = mi_alloc(sizeof(mi_char1) * 1000);
    strflds = mi_alloc(sizeof(mi_char1) * 400);
    strcpy(strsql,"");  
    strcpy(strvalues,"");  
    strcpy(strflds,"");  
    for(inti=0;inti<15;inti++) {
        if(!astrvalues[inti][0]) break;
        if(inti>0){
            strcat(strvalues",");
            strcat(strflds",");
        }
        strcat(strflds,astrflds[inti]);
        if(!strcmp(astrvalues[inti],"null"))
            strcat(strvalues,"null");
        else {
            strvaluetype=change_to_type(astrvalues[inti]);
            strcat(strvalues,strvaluetype);
        }
    }
    strcat(strsql,"INSERT INTO ");
    strcat(strsql,strtbl);
    strcat(strsql,"(");
    strcat(strsql,strflds);
    strcat(strsql,"last_modified")");
    strcat(strsql," VALUES (");
    strcat(strsql,strvalues);
    strcat(strsql,"current_timestamp);");
    /* deallocate memory */
    mi_free(strvalues);
    mi_free(strflds);
    /* return INSERT SQL statement */
    return strsql;
}

mi_char1 *change_to_type(mi_char1 *strvalue)    /* value to change */
{
    /* Reformats value: adds to all values: '' */
    MI_ROW *row;
    mi_char1 *strexpr;
    /* allocate memory dynamically */
    strexp = mi_alloc(sizeof(mi_char1) * 350);
    if(!strcmp(strvalue,"null"))
```c
strcpy(strexpr,"null");
else {
    strcpy(strexpr,"'\n");
    strcat(strexpr,strvalue);
    strcat(strexpr,"'\n");
}
return strexpr;

mi_char1 *get_column_value(MI_CONNECTION *conn,    /* connect. to def. database */
                                 mi_char1 *strsql)
    /* SQL statement to execute */
{
    /* Executes an SQL statement that returns a single column value */
    MI_ROW *row;
    int intlen,interr,intresult;
    mi_char1 *strvalue;
    /*execute query */
    mi_exec(conn,strsql,0);
    while((intresult = mi_get_result(conn))!=MI_NO_MORE_RESULTS) {
    switch(intresult) {
        case MI_ROWS:
            /* retrieve the single required row */
            row=mi_next_row(conn,&interr);
            /*get the value of the single column */
            mi_value(row,0,&strvalue,&intlen);
        }
    }
    return strvalue;
}

void get_words(mi_char1 *straux,    /* string delimited by "^" */
                                 mi_char1 *astraux[15])    /* array of strings */
{
    /* Generates an array of strings from a ^ delimited list of strings */
    mi_integer inti;
    mi_char1 *pstr,*pstrword,*strword;
    /* the max. length that can have a values is 300 */
    strword = mi_alloc(300 * sizeof(mi_char1));
    pstr=straux;
    inti=0;
    while (*pstr) {
        pstrword=strword;
        while(*pstr!='^' && *pstr) {
            *pstrword=*pstr;
            pstrword++;
            pstr++;
        }
        if(*pstr) pstr++;
        *pstrword=\0;
        /* allocate dynamically memory for string depending on len of */
        /* length of the string */
        astraux[inti] = mi_alloc(strlen(strword)+1) * sizeof(mi_char1));
        strcpy(astraux[inti],strword);
        inti++;
    }
    /* deallocate memory */
    mi_free(strword);
}
```
A.1.2 DML Statement Generation Modules: Set instances
Structure Chart: GF_complex_insert

Figure A2. Structure Chart GF_complex_insert.c
Structure Chart `get_complex_insert_statement` module (GF_complex_insert):

![Diagram for `get_complex_insert_statement` module](image)

Figure A3. Structure Chart `get_complex_insert_statement` module (GF_complex_insert)

Structure chart `get_complex_update_statement` module:

![Diagram for `get_complex_update_statement` module](image)

Figure A4. Structure Chart `get_complex_update_statement` module (GF_complex_insert)
Source Code: GF_complex_insert.c

/* GF_complex_insert.c */

#include "stdio.h"
#include "string.h"
#include "mi.h"

/* prototypes */

mi_char1 *get_last_modified_column(MI_CONNECTION *conn, /* conn to def. db */
    mi_char1 *strtbl);

mi_char1 *get_set_name(MI_CONNECTION *conn, /* conn to def. db */
    mi_char1 *strtbl);

mi_integer insert_parent_tuple(MI_CONNECTION *conn, /* conn to def. db. */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *strparenttbl, /* name of parent table */
    mi_char1 *astrflds[15], /* array of columns */
    mi_char1 *astrvalues[15]); /* array of values */

mi_char1 *get_complex_update_statement(MI_CONNECTION *conn, /* conn. to def db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *strparenttbl, /* parent table */
    mi_char1 *astrflds[15], /* array of columns */
    mi_char1 *astrvalues[15], /* array of values */
    mi_char1 *strwhere, /* ext. where */
    mi_char1 *strsetwhere); /* int. where set */

mi_char1 *get_complex_insert_statement(MI_CONNECTION *conn, /* conn. to def db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *strparenttbl, /* parent table */
    mi_char1 *astrflds[15], /* array of columns */
    mi_char1 *astrvalues[15], /* array of values */
    mi_char1 *strwhere); /* ext. WHERE */

mi_char1 get_column_type(MI_CONNECTION *conn, /* conn. to def db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *strcol); /* name of column */

mi_char1 *get_ext_where_clause(MI_CONNECTION *conn, /* conn to def. db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *astrflds[15], /* array of columns */
    mi_char1 *astrvalues[15]); /* array of values */

mi_char1 *get_parent_table(MI_CONNECTION *conn, /* conn. to def db */
    mi_char1 *strtbl); /* name of table */

mi_integer test_parent_row_exists(MI_CONNECTION *conn, /* connection to db */
    mi_char1 *strparenttbl, /* parent table */
    mi_char1 *strwhere); /* array of comp. of pk */

mi_integer test_setrow_exists(MI_CONNECTION *conn, /* connection to db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *strparenttbl, /* parent table */
    mi_char1 *strwhere, /* array of comp. of pk */
    mi_char1 *strsetwhere); /* WHERE inside set */
mi_integer get_pk_cols(MI_CONNECTION *conn, /* conn. to def. db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *astrpkcols[15]); /* array cols.comp.PK */

mi_char1 *get_where_clause(mi_char1 *strtbl, /* name of table */
    mi_char1 *astrflds[15], /* array of columns */
    mi_char1 *astrvalues[15], /* array of values */
    mi_char1 *astrpkcols[15]); /* array of comp. of pk */

mi_char1 *change_to_type(mi_char1 *strvalue); /* value to change */

mi_char1 *get_column_value(MI_CONNECTION *conn, /* connect. to def. db */
    mi_char1 *strsql); /* SQL stat. to execute */

void get_words(mi_char1 *straux, /* string delim. by "^" */
    mi_char1 *astraux[15]); /* array of strings */

/* end of prototypes */

mi_text *GF_complex_insert(mi_text *strtblarg,mi_text *strfldsarg,mi_text *strvaluesarg)
{
    MI_CONNECTION *conn; /* connection to open database */
    mi_char1 *strsql; /* complex SQL statement returned */
    mi_char1 *strwhere; /* exterior where clause SQL statement */
    mi_char1 *strsetwhere; /* interior where clause(set) SQL statement */
    mi_char1 *strparenttbl; /* name of parent table that holds the set */
    mi_char1 *strtbl; /* name of table */
    mi_char1 *strflds; /* list of columns */
    mi_char1 *strvalues; /* list of values */
    mi_char1 *astrflds[15]; /* array of columns */
    mi_char1 *astrvalues[15]; /* array of values */
    mi_char1 *astrpkcols[15]; /* array of cols. that are components PK */

    mi_integer inti, intcounter, int_row_exists, introwsaffected, interror;

    /* allocate memory dynamically */
    strtbl = mi_alloc(sizeof(mi_char1) * 50);
    strflds = mi_alloc(sizeof(mi_char1) * 300);
    strvalues = mi_alloc(sizeof(mi_char1) * 700);
    strsql = mi_alloc(sizeof(mi_char1) * 2000);
    strparenttbl = mi_alloc(sizeof(mi_char1) * 50);
    strsetwhere = mi_alloc(sizeof(mi_char1) * 300);
    strwhere = mi_alloc(sizeof(mi_char1) * 300);

    /* allocate memory dynamically to array of chars */
    for(inti=0;inti<15;inti++) {
        astrflds[inti] = mi_alloc(sizeof(mi_char1) * 30);
        astrvalues[inti] = mi_alloc(sizeof(mi_char1) * 300);
        astrpkcols[inti] = mi_alloc(sizeof(mi_char1) * 30);
    }

    /* convert text types into strings */
    mi_text_to_buffer(strtblarg,strtbl);
    mi_text_to_buffer(strfldsarg,strflds);
    mi_text_to_buffer(strvaluesarg,strvalues);
/ opens current opened database */
conn=mi_open(NULL,NULL,NULL);

/* generate array of columns to insert values*/
get_words(strflds,astrflds);

/* generate array of values associated to each column */
get_words(strvalues,astrvalues);

/* generate array of cols that are components of the pseudo keys */
interror = get_pk_cols(conn,strtbl,astrpkcols);
if(interror == 0) {
    /* deallocate memory */
    mi_free(strtbl);
    mi_free(strflds);
    mi_free(strvalues);
    mi_free(strparenttbl);
    mi_free(strsetwhere);
    mi_free(strwhere);
    /* deallocate memory to array of chars */
    for(inti=0;inti<15;inti++) {
        if(strlen(astrpkcols[inti])>0) mi_free(astrpkcols[inti]);
        if(strlen(strvalues[inti])>0) mi_free(astrvalues[inti]);
        if(strlen(strflds[inti])>0) mi_free(astrflds[inti]);
    }
    strcpy(strsql,"Return 0;");
    /* convert into text type again */
    return mi_string_to_text(strsql);
}

/* generate interior WHERE clause of the set on pseudo PKs set */
strcpy(strsetwhere,get_where_clause(strtbl,astrflds,astrvalues,astrpkcols));

/* generate exterior WHERE clause of the parent table that holds the set */
strcpy(strwhere,get_ext_where_clause(conn,strtbl,astrflds,astrvalues));

/* get name of parent table */
strcpy(strparenttbl,get_parent_table(conn,strtbl));

/* test parent row exists to insert tuple in parent table if it does not exist */
int_row_exists = test_parent_row_exists(conn,strparenttbl,strwhere);
if(int_row_exists == 0) {
    introwsaffected =
    insert_parent_tuple(conn,strtbl,strparenttbl,astrflds,astrvalues);
    if(introwsaffected == 0) {
        /* deallocate memory */
        mi_free(strtbl);
        mi_free(strflds);
        mi_free(strvalues);
        mi_free(strparenttbl);
        mi_free(strsetwhere);
        mi_free(strwhere);
        /* deallocate memory to array of chars */
        for(inti=0;inti<15;inti++) {
            if(strlen(astrpkcols[inti])>0) mi_free(astrpkcols[inti]);
            if(strlen(strvalues[inti])>0) mi_free(astrvalues[inti]);
            if(strlen(strflds[inti])>0) mi_free(astrflds[inti]);
        }
    }
strcpy(strsql,"Return 0;"); /* convert into text type again */
return mi_string_to_text(strsql);
}
/* Parent row does not exist, therefore child set row does not exist as well */
strsql=
get_complex_insert_statement(conn,strtbl,strparenttbl,astrflds,astrvalues,strwhere);
}
else {
/* generate SQL query to test if set row exists in the table */
int_row_exists = test_setrow_exists(conn,strtbl,strparenttbl,strwhere,strsetwhere);
if(int_row_exists == 1) {
strsql =
get_complex_update_statement(conn,strtbl,strparenttbl,astrflds,astrvalues,
strwhere,strsetwhere);
}
else {
/* Row does not exist, it will insert new row with current timestamp */
strsql=
get_complex_insert_statement(conn,strtbl,strparenttbl,astrflds,astrvalues,
strwhere);
}
}
/* close the connection to the default database */
mi_close(conn);
/* deallocate memory */
mi_free(strtbl);
mi_free(strflds);
mi_free(strvalues);
mi_free(strparenttbl);
mi_free(strsetwhere);
mi_free(strwhere);
/* deallocate memory to array of chars */
for(inti=0;inti<15;inti++) {
    if(strlen(astrpkcols[inti])>0) mi_free(astrpkcols[inti]);
    if(strlen(astrvalues[inti])>0) mi_free(astrvalues[inti]);
    if(strlen(astrflds[inti])>0) mi_free(astrflds[inti]);
}
/* convert into text type again */
return mi_string_to_text(strsql);
}
mi_integer insert_parent_tuple(MI_CONNECTION *conn, /* conn to def. db. */
mi_char1 *strtbl, /* name of table */
mi_char1 *strparenttbl, /* name of parent table */
mi_char1 *astrflds[15], /* array of columns */
mi_char1 *astrvalues[15]) /* array of values */
{
/* Inserts tuple in parent table that holds the set */
mi_char1 strcoltype, *strvalue_type, *astrvalues, *astrflds, *strsql;
mi_integer inti, insearchcols, intresult, introwsaffected;
```c
/* allocate memory dynamically */
strsql = mi_alloc(sizeof(mi_char1) * 700);
strflds = mi_alloc(sizeof(mi_char1) * 200);
strvalues = mi_alloc(sizeof(mi_char1) * 300);

strcpy(strvalues, "\0");
strcpy(strflds, "\0");
strcpy(strsql, "\0");

int intsearchcols = 0;
for (int i = 0; i < 15; i++) {
    if (!astrvalues[i][0]) break;
    strcoltype = get_column_type(conn, strtbl, astrflds[i]);
    switch (strcoltype) {
    case 'i':
        continue;
    case 's':
        intsearchcols++;
        if (intsearchcols > 1) {
            strcat(strflds, "," );
            strcat(strvalues, "," );
        }
        strcat(strflds, astrflds[i]);
        strvalue_type = change_to_type(astrvalues[i]);
        strcat(strvalues, strvalue_type);
        break;
    }
}

/* generate SQL command */
/* insert tuple parent table */
strcpy(strsql, "INSERT INTO ");
strcat(strsql, strparenttbl);
strcat(strsql, "(");
strcat(strsql, strflds);
strcat(strsql, ",url) VALUES(");
strcat(strsql, strvalues);
strcat(strsql, ",'URL to be updated');

/* execute query */
while ((intresult = mi_get_result(conn)) != MI_NO_MORE_RESULTS) {
    switch (intresult) {
    case MI_ERROR:
        introwsaffected = 0;
        break;
    case MI_DML:
        introwsaffected = mi_result_row_count(conn);
        if (introwsaffected == MI_ERROR)
            introwsaffected = 0;
        break;
    default:
        introwsaffected = 0;
        break;
    }
```
mi_free(strflds);
mi_free(strvalues);
mi_free(strsql);

return introwsaffected;

mi_char1 *get_ext_where_clause(MI_CONNECTION *conn, /* conn to def. db */
mi_char1 *strtbl,             /* name of table        */
mi_char1 *astrflds[15],       /* array of columns */
mi_char1 *astrvalues[15])     /* array of values      */
{
    /* Generates exterior Where clause parent table that holds the set */

    mi_char1 strcoltype, *strvalue_type, *strwhere;
    mi_integer inti, intsearchcols;

    /* allocate memory dynamically */
    strwhere = mi_alloc(sizeof(mi_char1) * 500);
    strcpy(strwhere,"");
    intsearchcols=0;
    for(inti=0;inti<15;inti++) {
        if(!astrvalues[inti][0]) break;
        strcoltype=get_column_type(conn,strtbl,astrflds[inti]);
        switch(strcoltype) {
            case 'i':
                continue;
            case 's':
                intsearchcols++;
                if(intsearchcols>1)
                    strcat(strwhere," AND ");
                if(!strcmp(astrflds[inti],"url"))
                    strcat(strwhere,astrflds[inti]);
                else {
                    strcat(strwhere,"upper(");
                    strcat(strwhere,astrflds[inti]);
                    strcat(strwhere,")");
                }
                strcat(strwhere,"=");
                strvalue_type=change_to_type(astrvalues[inti]);
                if(!strcmp(astrflds[inti],"url"))
                    strcat(strwhere,strvalue_type);
                else {
                    strcat(strwhere,"upper(");
                    strcat(strwhere,strvalue_type);
                    strcat(strwhere,")");
                }
                break;
            }
    }
    return strwhere;


```c
mi_integer get_pk_cols(MI_CONNECTION *conn, /* conn. to def. db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *astrpkcols[15]) /* array of cols.comp.PK */
{
    /* Returns array of columns that are component of the key */

    MI_ROW *row;
    mi_char1 *strcolvalue, *strsql;
    mi_integer intlen,interr,intresult,inti;

    /* allocate memory dynamically */
    strsql  = mi_alloc(sizeof(mi_char1) * 300);

    /* generate SQL command */
    /* get the columns that are components of the primary key */
    strcpy(strsql,"SELECT c.column_source FROM _columns c"");
    strcat(strsql," WHERE c.table_source='");
    strcat(strsql,strtbl);
    strcat(strsql,"' AND c.column_pk = 1;");

    /*execute query */
    mi_exec(conn,strsql,0);

    inti = 0;
    while((intresult = mi_get_result(conn))!=MI_NO_MORE_RESULTS) {
        switch(intresult) {
            case MI_ERROR:
                mi_free(strsql);
                return 0;
            case MI_ROWS:
                /* for each row, get only one column */
                while ((row=mi_next_row(conn,&interr)) != NULL)
                {
                    /* get the value of the single column */
                    switch (mi_value(row,0,&strcolvalue,&intlen))
                    {
                        case MI_ERROR:
                            mi_free(strsql);
                            return 0;
                        case MI_NULL_VALUE:
                            mi_free(strsql);
                            return 0;
                        case MI_NORMAL_VALUE:
                            astrpkcols[inti] = strcolvalue;
                            inti++;
                            break;
                        default:
                            mi_free(strsql);
                            return 0;
                    }
                }
        }
        break;
    }
}
```
mi_integer test_parent_row_exists(MI_CONNECTION *conn, /* connection to db */
    mi_char1 *strparenttbl, /* parent table */
    mi_char1 *strwhere) /* Ext. Where clause */
{
    /* Test if parent row table that holds the set exists */

    mi_integer inti, intj;
    mi_char1 *strnumrows, *strsql;

    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 500);

    strcpy(strsql,"SELECT count(*) FROM ");
    strcat(strsql,strparenttbl);
    strcat(strsql," WHERE ");
    strcat(strsql,strwhere);
    strcat(strsql,";");

    /*execute query */
    strnumrows = get_column_value(conn,strsql);

    /* deallocate memory */
    mi_free(strsql);

    /* return number of rows retrieved, should be 0 or 1 */
    if(!strcmp(strnumrows,"0")) return 0;
    else return 1;
}

mi_integer test_setrow_exists(MI_CONNECTION *conn, /* connection to db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *strparenttbl, /* parent table */
    mi_char1 *strwhere, /* Exterior Where clause */
    mi_char1 *strsetwhere) /* Interior Where clause */
{
    /* Tests set instance exists */

    mi_integer inti, intj;
    mi_char1 *strnumrows, *strsql;

    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 500);

    strcpy(strsql,"SELECT count(*) FROM ");
    strcat(strsql,"m");
    strcat(strsql,strtbl);
    strcat(strsql," WHERE poid = (SELECT oid FROM ");
    strcat(strsql,strparenttbl);
    strcat(strsql," WHERE ");
strcat(strsql,strwhere);
strcat(strsql,"") AND ");
strcat(strsql,strsetwhere);
strcat(strsql,";");

/>*/execute query */
strnumrows = get_column_value(conn,strsql);

/>*/deallocation memory */
mi_free(strsql);

/>*/return number of rows retrieved, should be 0 or 1 */
if(strncmp(strnumrows,"0",)) return 0;
elsereturn 1;

}

mi_char1 *get_where_clause(mi_char1 *strtbl,
/* name of table */
mi_char1 *astrflds[15],    /* array of columns */
mi_char1 *astrvalues[15],  /* array of values */
mi_char1 *astrpkcols[15])  /* array of comp. of pk */
{

/>*/Generates exterior Where clause */

mi_integer inti, intj;

mi_char1 *strvaluetype, *strwhere, *straux;

/>*/ allocate memory dynamically */
strwhere = mi_alloc(sizeof(mi_char1) * 300);
strvaluetype = mi_alloc(sizeof(mi_char1) * 350);

strcpy(strwhere,""");

for(inti=0;inti<15;inti++) {
if (!astrpkcols[inti][0])
break;
if(inti > 0)
    strcat(strwhere," AND ");
straux=strstr(astrpkcols[inti],"url");
if (straux!=NULL)
    strcat(strwhere,astrpkcols[inti]);
else{
    strcat(strwhere,"upper(");
    strcat(strwhere,astrpkcols[inti]);
    strcat(strwhere,")");
}
strcat(strwhere,"=");
for(intj=0;intj<15;intj++) {
if (!astrflds[intj][0])
break;
if(!strncmp(astrflds[intj],astrpkcols[intj])) {
    strvaluetype = change_to_type(astrvalues[intj]);
    if (straux!=NULL)
        strcat(strwhere,strvaluetype);
    else{
        strcat(strwhere,"upper(");
        strcat(strwhere,strvaluetype);
        strcat(strwhere,")");
}
}
mi_char1 *get_complex_update_statement(MI_CONNECTION *conn, /* conn. to def db */
             mi_char1 *strtbl, /* name of table */
             mi_char1 *strparenttbl, /* parent table */
             mi_char1 *astrflds[15], /* array of columns */
             mi_char1 *astrvalues[15], /* array of values */
             mi_char1 *strwhere, /* ext. where */
             mi_char1 *strsetwhere) /* int. where set */
{
    /* Generates complex SQL set instance update */

    mi_char1 *str_last_modified;
    mi_integer inti, intinsertcols;

    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 1500);
    strsetupdate = mi_alloc(sizeof(mi_char1) * 700);
    strsetname = mi_alloc(sizeof(mi_char1) * 50);
    str_last_modified = mi_alloc(sizeof(mi_char1) * 50);
    
    strcpy(strsetupdate,"");
    strcpy(strsql,"");
    intinsertcols=0;
    for(inti=0;inti<15;inti++) {
        if(!astrvalues[inti][0]) break;
        strcoltype=get_column_type(conn,strtbl,astrflds[inti]);
        switch(strcoltype) {
        case 'i':
            intinsertcols++;
            if(intinsertcols>1)
                strcat(strsetupdata,",");
            strcat(strsetupdata,astrflds[inti]);
            strcat(strsetupdata,"");
            strvaluetype=change_to_type(astrvalues[inti]);
            strcat(strsetupdata,strvaluetype);
            break;
        case 's':
            continue;
        }
    }

    /* get name of set */
    strcpy(strsetname,get_set_name(conn,strtbl));
/* get name of last_modified column of the set */
strcpy(str_last_modified, get_last_modified_column(conn,strtbl));

strcpy(strsql,"UPDATE ");
strcat(strsql,strparenttbl);
strcat(strsql," SET(update ");
strcat(strsql,strparenttbl);
strcat(strsql,".");
strcat(strsql," SET ");
strcat(strsql,strsetname);
strcat(strsql," WHERE ");
strcat(strsql,strsetwhere);
strcat(strsql," SET ");
strcat(strsql,str_last_modified);
strcat(strsql," WHERE ");
strcat(strsql,strwhere);
strcat(strsql,"; ");

/* deallocate memory */
mi_free(strsetupdate);
mi_free(str_last_modified);
mi_free(strsetname);

/* return UPDATE SQL statement */
return strsql;
}

mi_char1 *get_complex_insert_statement(MI_CONNECTION *conn, /* conn. to def db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *strparenttbl, /* parent table */
    mi_char1 *astrflds[15], /* array of columns */
    mi_char1 *astrvalues[15], /* array of values */
    mi_char1 *strwhere) /* ext. WHERE */
{
  /* Generates complex SQL statement insert set instance */

  mi_char1 *strvalue_type, *strsql, *strvalues, *strflds, *str_last_modified;
  mi_char1 *strsetname, strcoltype;
  mi_integer inti, intinsertcols;

  /* allocate memory dynamically */
  strsql = mi_alloc(sizeof(mi_char1) * 2000);
  strvalues = mi_alloc(sizeof(mi_char1) * 1000);
  strflds = mi_alloc(sizeof(mi_char1) * 400);
  strsetname = mi_alloc(sizeof(mi_char1) * 50);
  str_last_modified = mi_alloc(sizeof(mi_char1) * 50);

  strcpy(strsql,"");
  strcpy(strvalues,"");
  strcpy(strflds,"");
  
  intinsertcols=0;
  for(inti=0;inti<15;inti++) {
    if(!astrvalues[inti][0]) break;
    strcoltype=get_column_type(conn,strtbl,astrflds[inti]);
    switch(strcoltype) {
case 'i':
    int insertcols++;
    if(insertcols>1) {
        strcat(strflds,"","");
        strcat(strvalues,"","");
    }
    strcat(strflds, astrflds[inti]);
    strvalue_type=change_to_type(astrvalues[inti]);
    strcat(strvalues, strvalue_type);
    break;
    case 's':
        continue;
    }

/* get name of set */
strcpy(strsetname, get_set_name(conn, strtbl));

/* get name of last_modified column of the set */
strcpy(str_last_modified, get_last_modified_column(conn, strtbl));

strcpy(strsql,"UPDATE ");
strcat(strsql, strparenttbl);
strcat(strsql," SET(INSERT INTO ");
strcat(strsql, strparenttbl);
strcat(strsql,".");
strcat(strsql, strsetname);
strcat(strsql,"( ");
strcat(strsql, strflds);
strcat(strsql," ");
strcat(strsql, str_last_modified);
strcat(strsql," VALUES(");
strcat(strsql, strvalues);
strcat(strsql," current_timestamp)) WHERE ");
strcat(strsql, strwhere);
strcat(strsql,";");

/* deallocate memory */
mi_free(strvalues);
mi_free(strflds);
mi_free(str_last_modified);
mi_free(strsetname);

/* return complex INSERT SQL statement */
return strsql;
}

mi_char1 get_column_type(MI_CONNECTION *conn, /* conn. to def db */
   mi_char1 *strtbl, /* name of table */
   mi_char1 *strcol) /* name of column */
{
    /* Retrieves the column type: 's' search or 'i' insert */
    mi_char1 *strvalue;
    mi_char1 *strsql;

    /* allocate dynamically memory */
    strsql= mi_alloc(sizeof(mi_char1) * 200);
/* generate SQL command */
strcpy(strsql,"SELECT c.column_type FROM _columns c");
strcat(strsql," WHERE c.table_source=");
strcat(strsql,strtbl);
strcat(strsql,"' AND c.column_source='");
strcat(strsql,strcol);
strcat(strsql,"';");

/* execute query */
strvalue = get_column_value(conn,strsql);

/* deallocate memory */
mi_free(strsql);

/* return only 1st letter */
return strvalue[0];
}

mi_char1 *get_parent_table(MI_CONNECTION *conn, /* conn. to def db */
                           mi_char1 *strtbl) /* name of table */
{
/* Retrieves the parent table name that holds the set of a given set table */

mi_char1 *strvalue;
mi_char1 *strsql;

/* allocate memory dynamically */
strsql = mi_alloc(sizeof(mi_char1) * 300);

/* generate SQL command */
strcpy(strsql,"SELECT t.parent_table FROM _tables t");
strcat(strsql," WHERE t.table_source='");
strcat(strsql,strtbl);
strcat(strsql,"';");

/* execute query */
strvalue = get_column_value(conn,strsql);

/* deallocate memory */
mi_free(strsql);

return strvalue;
}

mi_char1 *get_set_name(MI_CONNECTION *conn, /* conn to def. db */
                        mi_char1 *strtbl) /* name of table */
{
/* Retrieves the set name of a given table (set name for sets or null for tables) */

mi_char1 *strvalue;
mi_char1 *strsql;

/* allocate memory dynamically */
strsql=mi_alloc(sizeof(mi_char1) * 200);

/* generate SQL command */
strcpy(strsql,"SELECT t.set_name FROM _tables t");
strcat(strsql," WHERE t.table_source='");
strcat(strsql,strtbl);
strcat(strsql,"';");

/*execute query */
strvalue = get_column_value(conn,strsql);

/* free memory */
mi_free(strsql);

return strvalue;

}

mi_char1 *get_last_modified_column(MI_CONNECTION *conn, /* conn to def. db */
                                   mi_char1 *strtbl) /* name of table */
{
    /* Retrieves the last_modified column of the table */
    mi_char1 *strvalue;
    mi_char1 *strsql;

    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 200);

    /* generate SQL command */
    strcpy(strsql,"SELECT unique(c1.column_name) FROM columns c1, tables t1");
    strcat(strsql," WHERE c1.column_member = t1.table_type AND t1.table_name='");  
    strcat(strsql,strtbl); 
    strcat(strsql," AND c1.column_name like '%_last_modified';");

    /*execute query */
    strvalue = get_column_value(conn,strsql);

    /* free memory */
    mi_free(strsql);

    return strvalue;

}

mi_char1 *change_to_type(mi_char1 *strvalue) /* value to change */
{
    /* Reformats value: adds to all values: '' */
    MI_ROW *row;
    mi_char1 *strexpr;

    /* allocate memory dynamically */
    strexpr = mi_alloc(sizeof(mi_char1) * 350);

    if(!strcmp(strvalue,"null"))
        strcpy(strexpr,"null");
    else {
        strcpy(strexpr,"'");  
        strcat(strexpr,strvalue);  
        strcat(strexpr,");
    
```
return strexpr;
}

mi_char1 *get_column_value(MI_CONNECTION *conn,  /* connect. to def. db */
                           mi_char1 *strsql)  /* SQL stat. to execute */
{
    /* Executes SQL statement that returns a single column value */
    MI_ROW *row;
    int intlen,interr,intresult;
    mi_char1 *strvalue;

    /*execute query */
    mi_exec(conn,strsql,0);

    while((intresult = mi_get_result(conn))!=MI_NO_MORE_RESULTS) {
        switch(intresult) {
        case MI_ROWS:  
            /* retrieve the single required row */
            row=mi_next_row(conn,&interr);

            /*get the value of the single column */
            mi_value(row,0,&strvalue,&intlen);
        }
    }
    return strvalue;
}
A.2 Query Statement Generation Modules:
A.2.1 Query Statement Generation Modules: Row instances

Structure Chart: GF_squery_simple

Figure A5. Structure Chart: GF_squery_simple

Source Code: GF_squery_simple.c

/* GF_squery_simple.c */

#include "stdio.h"
#include "string.h"
#include "mi.h"

/* prototypes */
mi_char1 *get_column_value(MI_CONNECTION *conn, /* connect. to def. db */
               mi_char1 *strsql); /* SQL stat. to execute */

char *get_data_type(MI_CONNECTION *conn, /* conn. to def. db */
               char *strtbl, /* name of table */
               char *strcol); /* name of column */

void get_words(mi_char1 *straux, /* string delim. by "^" */
               mi_char1 *astraux[20]); /* array of strings */

mi_text *GF_squery_simple(mi_text *strtblarg,mi_text *strfldsarg,mi_text *strincludesarg,
                        mi_text *stroptionsarg,mi_text *strvaluesarg)
{
    MI_CONNECTION *conn; /* connection to open database */
    mi_char1 *strsql; /* complex SQL statement returned */
mi_char1 *strwhere;          /* where clause SQL statement */
mi_char1 *strexpr;           /* prebuilt where clause */
mi_char1 *strparenttbl;      /* name of parent table that holds the set */
mi_char1 *strtbl;            /* name of table */
mi_char1 *strflds;           /* list of columns */
mi_char1 *strincludes;       /* list of included columns */
mi_char1 *strvalues;         /* list of values */
mi_char1 *stroptions;        /* list of search options */
mi_char1 *strdatatype;       /* data type */
mi_char1 *astrflds[20];      /* array of columns */
mi_char1 *astrincludes[20];  /* array of columns included in the query */
mi_char1 *astrvalues[20];    /* array of values */
mi_char1 *astroptions[20];   /* array of search options */

int inti, intcounter;

inti = 0; inti++;
/* allocate memory dynamically */
strtbl = mi_alloc(sizeof(mi_char1) * 50);
strflds = mi_alloc(sizeof(mi_char1) * 300);
strincludes = mi_alloc(sizeof(mi_char1) * 40);
strvalues = mi_alloc(sizeof(mi_char1) * 700);
stroptions = mi_alloc(sizeof(mi_char1) * 150);
strsql = mi_alloc(sizeof(mi_char1) * 2100);
strwhere = mi_alloc(sizeof(mi_char1) * 1550);
stexpr = mi_alloc(sizeof(mi_char1) * 1500);

/* allocate memory dynamically to array of chars */
for(inti=0;inti<20;inti++) {
    astrflds[inti] = mi_alloc(sizeof(mi_char1) * 30);
    astrincludes[inti] = mi_alloc(sizeof(mi_char1) * 2);
    astrvalues[inti] = mi_alloc(sizeof(mi_char1) * 300);
    astroptions[inti] = mi_alloc(sizeof(mi_char1) * 10);
}

/* convert text types into strings */
mi_text_to_buffer(strtblarg,strtbl);
mi_text_to_buffer(strfldsarg,strflds);
mi_text_to_buffer(strexprarg,stexpr);
mi_text_to_buffer(strincludesarg,strincludes);
mi_text_to_buffer(stroptionsarg,stroptions);
mi_text_to_buffer(strvaluesarg,strvalues);
strcpy(strwhere,"");
strcpy(strexpr,"");

/* opens current opened database */
conn = mi_open(NULL,NULL,NULL);

/* generate array of columns */
get_words(strflds,astrflds);

/* generate array of columns */
get_words(strincludes,astrincludes);

/* generate array of search options */
get_words(stroptions,astroptions);

/* generate array of values associated to each column */
get_words(strvalues,astrvalues);
/* regenerate strflds delimited by "^" to be delim. by "," */
strcpy(strflds,""");
intcounter=0;
for(inti=0;astrflds[inti][0];inti++) {
    if(strlen(strflds)>0 && astrincludes[inti][0]=='1')
        strcat(strflds,",");
    if(astrincludes[inti][0]=='1')
        strcat(strflds,astrflds[inti]);
    if(!strcmp(astrvalues[inti],"**")) continue;
    else {
        intcounter++;
        if(intcounter>1) strcat(strexpr," AND ");
        strdatatype = get_data_type(conn,strtbl,astrflds[inti]);
        if(!strcmp(strdatatype,"text")) {
            strcat(strexpr,"upper(");
            strcat(strexpr,astrflds[inti]);
            strcat(strexpr," )");
            strcat(strexpr," upper('";
            strcat(strexpr,astroptions[inti]);
            strcat(strexpr," upper('";
            strcat(strexpr,astrvales[inti]);
            strcat(strexpr,""')");
        } else {
            if(!strcmp(strdatatype,"timestamp")) {
                strcat(strexpr,astrflds[inti]);
                strcat(strexpr,":text ");
                strcat(strexpr,astroptions[inti]);
                strcat(strexpr," "");
                strcat(strexpr,astrvales[inti]);
                strcat(strexpr,""');
            } else {
                /* type is integer, int1 or smallint */
                strcat(strexpr,astrflds[inti]);
                strcat(strexpr,astroptions[inti]);
                strcat(strexpr,astrvales[inti]);
            }
        }
    }
}
if((int) strlen(strexpr)>0) {
    strcat(strwhere," WHERE ");
    strcat(strwhere,strexpr);
}

/* generate SQL SELECT statement */
strcpy(strsql,"SELECT ");
strcat(strsql,strflds);
strcat(strsql," FROM ");
strcat(strsql,strtbl);
strcat(strsql,strwhere);
strcat(strsql,";");

/* close the connection to the default database */
mclose(conn);

/* deallocate memory */
mi_free(strtbl);
mi_free(strflds);
mi_free(strvalues);
mi_free(stroptions);
mi_free(strexpr);
mi_free(strwhere);

/* deallocate memory to array of chars */
for(inti=0;inti<20;inti++)
    if(strlen(astroptions[inti])>0) mi_free(astroptions[inti]);
    if(strlen(astrvalues[inti])>0) mi_free(astrvalues[inti]);
    if(strlen(astrflds[inti])>0) mi_free(astrflds[inti]);

/* convert into text type again */
return mi_string_to_text(strsql);

char *get_data_type(MI_CONNECTION *conn,
    char *strtbl,                     /* conn. to def. db */
    char *strcol)                     /* name of table */
{
    /* Retrieves ILLUSTRA column data type */
    char *strdatatype,*strsql;
    strsql=mi_alloc(sizeof(mi_char1) * 200);

    /* generate SQL command */
    strcpy(strsql,"SELECT c.column_data_type FROM _columns c");
    strcat(strsql," WHERE c.column_source='");
    strcat(strsql,strcol);
    strcat(strsql," AND c.table_source='");
    strcat(strsql,strtbl);
    strcat(strsql,"';");

    /*execute query */
    strdatatype = get_column_value(conn,strsql);
    mi_free(strsql);
    return strdatatype;
}

mi_char1 *get_column_value(MI_CONNECTION *conn,
    mi_char1 *strsql)                     /* connect. to def. db */
{
    /* SQL stat. to execute */

    /* Executes an SQL statement that returns a single column value */

    MI_ROW *row;
    int intlen,interr,intresult;
    mi_char1 *strvalue;

    /*execute query */
    mi_exec(conn,strsql,0);
while((intresult = mi_get_result(conn))!=MI_NO_MORE_RESULTS) {
    switch(intresult) {
    case MI_ROWS:
        /* retrieve the single required row */
        row=mi_next_row(conn,&interr);
        /* get the value of the single column */
        mi_value(row,0,&strvalue,&intlen);
    }
}
return strvalue;

A.2.2 Query Statement Generation Modules: Set instances

Structure Chart: GF_squery_complex

Figure A6. Structure Chart: GF_squery_complex

Source Code: GF_squery_complex.c

/* GF_squery_complex.c */
#include "stdio.h"
#include "string.h"
#include "mi.h"

/* prototypes */
mi_char1 get_column_type(MI_CONNECTION *conn, /* conn. to def db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *strcol); /* name of column */

mi_char1 *get_parent_table(MI_CONNECTION *conn, /* conn. to def db */
    mi_char1 *strtbl); /* name of table */

mi_char1 *get_column_value(MI_CONNECTION *conn, /* connect. to def. db */
    mi_char1 *strsql); /* SQL stat. to execute */

char *get_data_type(MI_CONNECTION *conn, /* conn. to def db */
    char *strtbl, /* name of table */
    char *strcol); /* name of column */

void get_words(mi_char1 *straux, /* string delim. by "^" */
    mi_char1 *astraux[15]); /* array of strings */

mi_text *GF_squery_complex(mi_text *strtblarg, mi_text *strfldsarg, mi_text *strincludesarg, mi_text *stroptionsarg, mi_text *strvaluesarg)
{
    MI_CONNECTION *conn; /* connection to open database */
    mi_char1 strcoltype; /* column type inser or search type */
    mi_char1 *strsql; /* complex SQL statement returned */
    mi_char1 *strwhere; /* interior where clause SQL statement */
    mi_char1 *strextwhere; /* exterior where clause SQL statement */
    mi_char1 *strparenttbl; /* name of parent table that holds the set */
    mi_char1 *strtbl; /* name of table */
    mi_char1 *strflds; /* list of columns */
    mi_char1 *strextflds; /* list of ext. cols parent table set */
    mi_char1 *strintflds; /* list of int. cols set */
    mi_char1 *strincludes; /* list of included columns */
    mi_char1 *strvalues; /* list of values */
    mi_char1 *stroptions; /* list of search options */
    mi_char1 *strdatatype; /* data type */
    mi_char1 *astrflds[20]; /* array of columns */
    mi_char1 *astrincludes[20]; /* array of included columns */
    mi_char1 *astrvalues[20]; /* array of values */
    mi_char1 *astroptions[20]; /* array of search options */

    int inti, intinsertcols, intsearchcols;

    /* allocate memory dynamically */
    strtbl = mi_alloc(sizeof(mi_char1) * 50);
    strparenttbl = mi_alloc(sizeof(mi_char1) * 50);
    strflds = mi_alloc(sizeof(mi_char1) * 300);
    strextflds = mi_alloc(sizeof(mi_char1) * 300);
    strintflds = mi_alloc(sizeof(mi_char1) * 300);
    strincludes = mi_alloc(sizeof(mi_char1) * 40);
    strvalues = mi_alloc(sizeof(mi_char1) * 700);
    stroptions = mi_alloc(sizeof(mi_char1) * 150);
    strsql = mi_alloc(sizeof(mi_char1) * 2500);
    strwhere = mi_alloc(sizeof(mi_char1) * 1500);
strextwhere = mi_alloc(sizeof(mi_char1) * 500);

/* allocate memory dynamically to array of chars */
for(inti=0;inti<15;inti++) {
    astrflds[inti] = mi_alloc(sizeof(mi_char1) * 30);
    astrincludes[inti] = mi_alloc(sizeof(mi_char1) * 2);
    astrvalues[inti] = mi_alloc(sizeof(mi_char1) * 300);
    astroptions[inti] = mi_alloc(sizeof(mi_char1) * 10);
}

/* convert text types into strings */
mi_text_to_buffer(strtblarg,strtbl);
mi_text_to_buffer(strfldsarg,strflds);
mi_text_to_buffer(strincludesarg,strincludes);
mi_text_to_buffer(stroptionsarg,stroptions);
mi_text_to_buffer(strvaluesarg,strvalues);

/* opens current opened database */
conn=mi_open(NULL,NULL,NULL);

/* generate array of columns */
get_words(strflds,astrflds);

/* generate array of columns */
get_words(strincludes,astrincludes);

/* generate array of search options */
get_words(stroptions,astroptions);

/* generate array of values associated to each column */
get_words(strvalues,astrvalues);

/* regenerate strflds delimited by "^" to be delim. by "," */
strcat(strintflds,"");
strcat(strextflds,"");
strcat(strwhere,"");
strcat(strextwhere,"");

intinsertcols=0;
intsearchcols=0;

for(inti=0;astrflds[inti][0];inti++) {
    strcoltype=get_column_type(conn,strtbl,astrflds[inti]);
    if(!strcmp(astrvalues[inti],"*")) continue;
    if(astrincludes[inti][0]=='1') {
        switch(strcoltype) {
            case 'i':
                if(strlen(strintflds)>0 && astrincludes[inti][0]=='1')
                    strcat(strintflds,"," );
                strcat(strintflds,astrflds[inti]);
                break;
            case 's':
                if(strlen(strextflds)>0 && astrincludes[inti][0]=='1')
                    strcat(strextflds,"," );
                strcat(strextflds,astrflds[inti]);
                break;
        }
    }
}
}
strdatatype = get_data_type(conn, strtbl, astrflds[inti]);
switch(strcoltype) {
    case 'i':
        intinsertcols++;
        if(intinsertcols>1) strcat(strwhere," AND ");
        if(!strcmp(strdatatype,"text")) {
            strcat(strwhere,"upper(");
            strcat(strwhere,astrflds[inti]);
            strcat(strwhere,")");
            strcat(strwhere,astroptions[inti]);
            strcat(strwhere," upper('");
            strcat(strwhere,astrvalues[inti]);
            strcat(strwhere,")'"));
        } else {
            if(!strcmp(strdatatype,"timestamp")) {
                strcat(strwhere,astrflds[inti]);
                strcat(strwhere,"::text ");
                strcat(strwhere,astroptions[inti]);
                strcat(strwhere,"");
                strcat(strwhere,astrvalues[inti]);
            } else {
                /* type is integer, int1 or smallint */
                strcat(strwhere,astrflds[inti]);
                strcat(strwhere,astroptions[inti]);
                strcat(strwhere,astrvalues[inti]);
            }
        }
        break;
    case 's':
        intsearchcols++;
        if(intsearchcols>1) strcat(strextwhere," AND ");
        /* all search cols. are of type text */
        strcat(strextwhere,"upper(");
        strcat(strextwhere,astrflds[inti]);
        strcat(strextwhere,astroptions[inti]);
        strcat(strextwhere," upper('");
        strcat(strextwhere,astrvalues[inti]);
        strcat(strextwhere,")'"));
        break;
}
/* get name of parent table */
strcpy(strparenttbl,get_parent_table(conn,strtbl));

/* generate SQL SELECT statement */
strcpy(strsql,"SELECT ");
if(strlen(strextflds) > 0)
    strcat(strsql, strextflds);
if(strlen(strextflds) > 0 && strlen(strintflds)>0)
    strcat(strsql,"",");
if(strlen(strintflds) > 0)
strcat(strsql,strintflds);
strcat(strsql," FROM ");
if(strlen(strextflds)==0)
  strcat(strsql,strtbl);
else {
  /* both tables by default */
  strcat(strsql,strtbl);
  strcat(strsql,".");
  strcat(strsql,strparenttbl);
}
if(strlen(strextflds) > 0 || strlen(strextwhere)>0 || strlen(strwhere) > 0) {
  strcat(strsql," WHERE ");
  if(strlen(strextflds) >0) {
    strcat(strsql,strparenttbl);
    strcat(strsql," .oid = ");
    strcat(strsql,strtbl);
    strcat(strsql," .oid");
  }
  if(strlen(strextflds) > 0 && strlen(strextwhere) > 0)
    strcat(strsql," AND ");
  if(strlen(strextwhere) > 0) {
    strcat(strsql," poid IN (SELECT ");
    strcat(strsql,strparenttbl);
    strcat(strsql," WHERE ");
    strcat(strsql,strextwhere);
    strcat(strsql,")");
  }
  if((strlen(strwhere) > 0 && strlen(strextflds) > 0) ||
    (strlen(strwhere) >0 && strlen(strextwhere) > 0))
    strcat(strsql," AND ");
  strcat(strsql,strwhere);
}
strcat(strsql,";");
/* close the connection to the default database */
mi_close(conn);

/* deallocate memory */
mi_free(strtbl);
mi_free(strparenttbl);
mi_free(strflds);
mi_free(strintflds);
mi_free(strextflds);
mi_free(strincludes);
mi_free(strvalues);
mi_free(stroptions);
mi_free(strextwhere);
mi_free(strwhere);

/* deallocate memory to array of chars */
for(inti=0;inti<15;inti++) {
  if(strlen(astroptions[inti])>0)  mi_free(astroptions[inti]);
  if(strlen(astrvalues[inti])>0)   mi_free(astrvalues[inti]);
  if(strlen(astrflds[inti])>0)     mi_free(astrflds[inti]);
  if(strlen(astrincludes[inti])>0) mi_free(astrincludes[inti]);
}
/* convert into text type again */
return mi_string_to_text(strsql);
}

mi_char1 *get_parent_table(MI_CONNECTION *conn, /* conn. to def db */
                          mi_char1 *strtbl) /* name of table */
{
    /* Retrieves the parent table that holds the set name of a given set table */

    mi_char1 *strvalue;
    mi_char1 *strsql;

    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 300);

    /* generate SQL command */
    strcpy(strsql,"SELECT t.parent_table FROM _tables t");
    strcat(strsql," WHERE t.table_source='");
    strcat(strsql,strtbl);
    strcat(strsql,");");

    /* execute query */
    strvalue = get_column_value(conn,strsql);

    /* deallocate memory */
    mi_free(strsql);

    return strvalue;
}

mi_char1 get_column_type(MI_CONNECTION *conn, /* conn. to def db */
                         mi_char1 *strtbl, /* name of table */
                         mi_char1 *strcol) /* name of column */
{
    /* Retrieves the column type: 's' search or 'i' insert */

    mi_char1 *strvalue;
    mi_char1 *strsql;

    /* allocate dynamically memory */
    strsql= mi_alloc(sizeof(mi_char1) * 200);

    /* generate SQL command */
    strcpy(strsql,"SELECT c.column_type FROM _columns c");
    strcat(strsql," WHERE c.table_source='");
    strcat(strsql,strtbl);
    strcat(strsql,",' AND c.column_source='");
    strcat(strsql,strcol);
    strcat(strsql,");");

    /* execute query */
    strvalue = get_column_value(conn,strsql);

    /* deallocate memory */
    mi_free(strsql);

    /* return only 1st letter */
    return strvalue[0];
char *get_data_type(MI_CONNECTION *conn, char *strtbl, char *strcol) {
    char *strdatatype,*strsql;
    strsql=mi_alloc(sizeof(mi_char1) * 200);
    /* generate SQL command */
    strcpy(strsql,"SELECT c.column_data_type FROM _columns c"");
    strcat(strsql," WHERE c.column_source='");
    strcat(strsql,strcol);
    strcat(strsql," AND c.table_source='");
    strcat(strsql,strtbl);
    strcat(strsql,"';");
    /*execute query */
    strdatatype = get_column_value(conn,strsql);
    mi_free(strsql);
    return strdatatype;
}

mi_char1 *get_column_value(MI_CONNECTION *conn, mi_char1 *strsql) {
    MI_ROW *row;
    int intlen,interr,intresult;
    mi_char1 *strvalue;
    /*execute query */
    mi_exec(conn,strsql,0);
    while((intresult = mi_get_result(conn))!=MI_NO_MORE_RESULTS) {
        switch(intresult) {
        case MI_ROWS:
            /* retrieve the single required row */
            row=mi_next_row(conn,&interr);
            
            /*get the value of the single column */
            mi_value(row,0,&strvalue,&intlen);
            }
    }
    return strvalue;
}
A.3 Query Filtering Modules

Structure Chart: GF_free_sql

Source Code: GF_free_sql

```c
/* GF_free_sql */

#include "stdio.h"
#include "string.h"
#include "ctype.h"
#include "mi.h"

/* prototypes */
void test_sql_statement(mi_char1 *strsql, /* name of table */
                        mi_char1 *strerr); /* error message */
void get_column_list(MI_CONNECTION *conn, /* conn. to def. db */
                    mi_char1 *strsql, /* sql statement */
                    mi_char1 *strcolumns, /* column list */
                    mi_char1 *strerror); /* error message */
mi_char1 *str_reformat(mi_char1 *straux); /* string delimited by "^^" */

mi_text *GF_free_sql(mi_text *strsqlarg) {
    MI_CONNECTION *conn;
    mi_char1 *strsql; /* sql statement */
    mi_char1 *strsqlaux; /* sql statement */
    mi_char1 *strcolumnlist; /* name of table */
    mi_char1 *strerror; /* error message */
    mi_integer inti, intcounter;

    /* allocate memory dynamically */
    strcolumnlist= mi_alloc(sizeof(mi_char1) * 1000);
    strsql = mi_alloc(sizeof(mi_char1) * 2500);
    strsqlaux = mi_alloc(sizeof(mi_char1) * 2500);
    strerror = mi_alloc(sizeof(mi_char1) * 100);

    /* convert text types into strings */
    mi_text_to_buffer(strsqlarg,strsql);
    /* opens current opened database */
    conn=mi_open(NULL,NULL,NUL);

    /* convert sql statement into lower case and replace ^ for ' */
```
strcpy(strsqlaux,str_reformat(strsql));

/* detect if it contains a not allowed operation */
test_sql_statement(strsqlaux,strerror);
if(strlen(strerror) >0)
    strcpy(strcolumnlist,"Error");
else
    /* generate array of cols that are components of the key */
    get_column_list(conn,strsqlaux,strcolumnlist,strerror);

/* close the connection to the default database */
m_close(conn);

/* free memory */
m_free(strsql);
m_free(strsqlaux);

/* concatenate column list and error message */
strcat(strcolumnlist,"^^");
strcat(strcolumnlist,strerror);

/* convert into text type again */
return mi_string_to_text(strcolumnlist);
}

void test_sql_statement(mi_char1 *strsql, /* name of table */
                        mi_char1 *strerr) /* error message */
{
    /* Checks if SQL statement is a valid statement */

    mi_integer inti, intj;
    mi_char1 *strnumrows;

    /* test if SQL statement contains not allowed statement or ",;" */
    if(strstr(strsql,"insert") !=NULL || strstr(strsql,"update") !=NULL ||
       strstr(strsql,"drop") !=NULL || strstr(strsql,"alter") !=NULL ||
       strstr(strsql,"alter") !=NULL || strstr(strsql,"create") !=NULL ||
       strstr(strsql,"delete") !=NULL || strstr(strsql,"copy") !=NULL ||
       strstr(strsql,"declare") !=NULL || strstr(strsql,"dump") !=NULL ||
       strstr(strsql,"grant") !=NULL || strstr(strsql,"declare cursor") !=NULL ||
       strstr(strsql,"revoke") !=NULL || strstr(strsql,"drop") !=NULL ||
       strstr(strsql,"set schema") !=NULL || strstr(strsql,"set time zone") !=NULL ||
       strstr(strsql,"set transaction") !=NULL ||
       strstr(strsql,"vacuum") !=NULL || strstr(strsql,"return") !=NULL) {
        strcpy(strerr,"SQL DDL, DML and other miscellaneous operations are not allowed!");
        return;
    }

    if(strstr(strsql,"select") ==NULL) {
        strcpy(strerr,"SQL statement must include SELECT clause!");
        return;
    }

    if(strstr(strsql,";")) !=NULL {
        strcpy(strerr,"Do not include `;` in SQL statement!");
        return;
    }

    /* statement is OK */
    strcpy(strerr,""");
}

void get_column_list(MI_CONNECTION *conn, /* conn. to def. db */
                     mi_char1 *strsql, /* sql statement */
                     mi_char1 *strcolumns, /* column list */
                     mi_char1 *strerr) /* error message */
{
    /* Returns an array of columns involved in the query */

    MI_ROW_DESC *rowdesc;
    mi_char1 *strsqlaux, *strcolname;
    mi_integer inti, intnumcols, intresult;

    /* allocate memory dynamically */
strsqlaux = mi_alloc(sizeof(mi_char1) * 2500);
strcpy(strsqlaux,strsql);
strcat(strsqlaux,";");
/*execute query*/
mi_exec(conn,strsqlaux,0);
strcpy(strcolumns,"");
if((intresult = mi_get_result(conn))!=MI_NO_MORE_RESULTS) {
  switch(intresult)
  {
    case MI_ERROR:
      strcpy(strcolumns,"Error");
      strcpy(strerr,"Malformed SQL statement");
      break;
    case MI_DDL:
      strcpy(strcolumns,"Error");
      strcpy(strerr,"SQL DDL statement is not allowed");
      break;
    case MI_DML:
      strcpy(strcolumns,"Error");
      strcpy(strerr,"SQL DML statement is not allowed");
      break;
    case MI_ROWS:
      /* get row descriptor */
      rowdesc=mi_get_row_desc_without_row(conn);
      /* get number of columns in the query or row */
      intnumcols= mi_column_count(rowdesc);
      /* generate column list */
      for(inti=0;inti<intnumcols;inti++) {
        strcolname=mi_column_name(rowdesc,inti);
        if (inti>0)
          strcat(strcolumns,"^");
        strcat(strcolumns,strcolname);
      }
      strcpy(strerr,"No Errors");
      break;
    default:
      strcpy(strcolumns,"Error");
      strcpy(strerr,"Unknown Error");
      break;
  }
  /* free mem. */
  mi_free(strsqlaux);
}
mi_char1 *str_reformat(mi_char1 *straux)         /* string delimited by "^" */
{  /* SQL statements are pre-formatted and must be reformatted into a valid SQL */
  /* statement. Changes every occurrence of the char ^ by the char '           */
  /* For example: SELECT uni_id from university WHERE state=^NSW^ is converted */
  /* into SELECT uni_id from university WHERE state='NSW'                      */
  /* We use this strategy because the SQL string is passed in a RETURN stat. */
  /* Therefore arguments must be enclosed between '' and the Query would become*/
  /* corrupted; */
  /* i.e Return GF_free_sql('SELECT uni_id from university WHERE state='NSW'') */
  /* would generate an error where as a pre-formatted query will be accepted */
  /* i.e Return GF_free_sql('SELECT uni_id from university WHERE state=^NSW^') */
  mi_char1 *pstr, *strreformat, *pstrreformat;
  strreformat = mi_alloc(sizeof(mi_char1) * (strlen(straux) + 1));
  pstrreformat = strreformat;
  pstr=straux;
  while ( (*pstr) )
    {
      if(*pstr=='^')
        *pstrreformat='\'';
      else
        *pstrreformat = tolower(*pstr);
      pstr++;
      pstrreformat++;
    }
} *pstrreformat='\0';
return stcreformat;
}
Appendix B. Web-Datablade Applications and Embedded JavaScripts

B.1 Insert/Update Web-Datablade Applications
   B.1.1 Insert/Update Web-Datablade Applications: State transition diagram
   B.1.2 Insert/Update Web-Datablade Applications: Source Code

B.2 Query Web-Datablade Applications
   B.2.1 Query Web-Datablade Applications: State transition diagram
   B.2.2 Query Web-Datablade Applications: Source Code

B.3 Query Filtering Web-Datablade Applications
   B.3.1 Query Filtering Web-Datablade Applications: State transition diagram
   B.3.2 Query Filtering Web-Datablade Applications: Source Code

B.4 Meta tag generator Web-Datablade Application
   B.4.1 Meta tag generator Web-Datablade Application: State transition Diagram
   B.4.2 Meta tag generator Web-Datablade Application: Source Code

B.1 Insert/Update Web-Datablade Applications
   B.1.1 Insert/Update Web-Datablade Applications: State transition diagram

Figure B1. Insert/Update Web-Datablade Applications: State transition diagram
B.1.2 Insert/Update Web-Datablade Applications: Source Code
B.1.2.1 main_ins

<!--
This script was originally written by Marcelino Alves Martins on January of 1997. Modified and adapted by Carlos F. Enguix on October 1997 -->

<html>
<head>
<title>UniGuide - Submit URL Section</title>

<script LANGUAGE="JavaScript">
<!-- to hide script contents from old browsers

//each node in the tree is an Array with 5+n positions
// node[0] is 0/1 when the node is closed/open
// node[1] is 0/1 when the folder is closed/open
// node[2] is 1 if the children of the node are documents
// node[3] is the name of the folder
// node[4] is the table source of the folder
// node[5]...node[5+n] are the n children nodes

// ***************
// Building the data in the tree

function generateTree()
{
  var aux1, aux2, aux3, aux4

  foldersTree = folderNode("UniGuide")
  aux2 = appendChild(foldersTree, folderNode("University Domain"))
  aux2 = appendChild(aux2, leafNode("Club & Association","club_association"))
  aux2 = appendChild(aux2, leafNode("Residential College","residential_college"))
  aux2 = appendChild(aux2, leafNode("Campus","campus"))
  aux2 = appendChild(aux2, folderNode("Admin. Entity Subdomain"))
  aux3 = appendChild(aux2, leafNode("Admin. Entity","administrative_entity"))
  aux3 = appendChild(aux2, leafNode("Staff","_administrative_entity_sstaff"))
  aux2 = appendChild(aux2, folderNode("Library Subdomain"))
  aux3 = appendChild(aux2, leafNode("Library ","library"))
  aux3 = appendChild(aux2, leafNode("Catalog ","_library_scatalogs"))
  aux3 = appendChild(aux2, leafNode("Staff ","_library_sstaff"))
  aux2 = appendChild(aux2, folderNode("Academic Entity Subdomain"))
  aux3 = appendChild(aux2, leafNode("Academic Entity ",&"academic_entity"))
  aux3 = appendChild(aux2, leafNode("Course ",&"_academic_entity_scourses"))
  aux3 = appendChild(aux2, leafNode("Course Unit ",&"_academic_entity_scourse_units"))
  aux3 = appendChild(aux2, leafNode("Publication ",&"_academic_entity_spublications"))
  aux3 = appendChild(aux2, leafNode("Project ",&"_academic_entity_sprojects"))
  aux3 = appendChild(aux2, leafNode("Staff ",&"_academic_entity_sstaff"))
  aux3 = appendChild(aux2, leafNode("Student ",&"_academic_entity_sstudents"))
  aux3 = appendChild(aux2, leafNode("Research Entity ",&"academic_entity_research_entity"))
  aux1 = appendChild(foldersTree, folderNode("Research Entity Domain"))
  aux2 = appendChild(aux1, leafNode("Research Entity ",&"research_entity"))
  aux2 = appendChild(aux1, leafNode("Course ",&"_research_entity_scourses"))
  aux2 = appendChild(aux1, leafNode("Course Unit ",&"_research_entity_scourse_units"))
  aux2 = appendChild(aux1, leafNode("Publication ",&"_research_entity_spublications"))

<!-- to hide script contents from old browsers -->
</script>
</head>
</html>
"_research_entity_spublications")
aux2 = appendChild(aux1, leafNode("Project  ">_research_entity_sprojects")
aux2 = appendChild(aux1, leafNode("Staff ">_research_entity_sstaff")
aux2 = appendChild(aux1, leafNode("Student  ">_research_entity_students")
aux2 = appendChild(aux1, leafNode("University ">research_entity_university")
}

// Auxiliary function to build the node
function folderNode(name)
{  
  var arrayAux
  arrayAux = new Array
  arrayAux[0] = 0
  arrayAux[1] = 0
  arrayAux[2] = 0
  arrayAux[3] = name

  return arrayAux
}

// Auxiliary function to build the node
// The entries in arrayAux[5]..array[length-1] are strings
function leafNode(name, table_source)
{  
  var arrayAux
  arrayAux = new Array
  arrayAux[0] = 0
  arrayAux[1] = 0
  arrayAux[2] = 1
  arrayAux[3] = name

  return arrayAux
}

//this way the generate tree function becomes simpler and less error prone
function appendChild(parent, child)
{  
  parent[parent.length] = child
  return child
}

/re redraws the left frame
function redrawTree()
{
  var doc = top.leftFrame.window.document

doc.clear()

doc.write("<body bgcolor='white'>")
redrawNode(foldersTree, doc, 0, 1, ""

doc.write('"<br>
doc.close()
}

//recursive function over the tree structure called by redrawTree
function redrawNode(foldersNode, doc, level, lastNode, leftSide)
```javascript
var j=0;
var i=0;
var strURL='<?MIVAR>$WEB_HOME<?/MIVAR>?MIvalObj=';

doc.write("<table border=0 cellspacing=0 cellpadding=0>")
doc.write("<tr><td valign = middle nowrap>")
doc.write(leftSide)
if (level>0)
if (lastNode) //the last 'brother' in the children array
    doc.write("<img src=" + strURL + "last width=16 height=22>")
    leftSide = leftSide + "<img src=" + strURL + "blank width=16 height=22>"
} else
    doc.write("<img src=" + strURL + "node width=16 height=22>")
    leftSide = leftSide + "<img src=" + strURL + "vertical width=16 height=22>"
}
displayIconAndLabel(foldersNode, doc)
doc.write("</table>")

if (foldersNode.length > 5 && foldersNode[0]) //there are sub-nodes and the folder is open
if (!foldersNode[2]) //for folders with folders
    level=level+1
for (i=5; ifoldersNode.length;i++)
    if (i==foldersNode.length-1)
        redrawNode(foldersNode[i], doc, level, 1, leftSide)
    else
        redrawNode(foldersNode[i], doc, level, 0, leftSide)
} else //for folders with documents
for (i=5; ifoldersNode.length;i++)
    doc.write("<table border=0 cellspacing=0 cellpadding=0 valign=center>")
doc.write("<tr><td nowrap>")
    if (i==foldersNode.length - 1)
        doc.write("<img src=" + strURL + "last width=16 height=22>")
    else
        doc.write("<img src=" + strURL + "node width=16 height=22>")
    doc.write(foldersNode[i])
    doc.write("</table>")
}
}

//builds the html code to display a folder and its label
function displayIconAndLabel(foldersNode, doc)
```
{
    var strURL="<MIVAR>$WEB_HOME</MIVAR>?MIValObj=";
    var strURLForm="<MIVAR>$WEB_HOME</MIVAR>?MIVal=form_ins&v_table_source=";

    if(foldersNode[4]!="")
        doc.write("<A HREF=" + strURLForm + foldersNode[4] + " target=rightFrame><img src=")
    else
    if (foldersNode[3] == 'UniGuide')
        doc.write(strURL + "uniguide width=24 height=22 border=noborder"></a")
    else
    {
        if (foldersNode[2])
            doc.write(strURL + "form width=24 height=22 border=noborder"></a")
        else
        {
            if (foldersNode[1])
                doc.write(strURL + "opened width=24 height=22 border=noborder"></a")
            else
                doc.write(strURL + "closed width=24 height=22 border=noborder"></a")
        }
    }
    doc.write("<td valign=middle align=left nowrap>")
    doc.write("<small>" + foldersNode[3] + "</small>"
}

//when a parent is closed all children also are
function closeFolders(foldersNode)
{
    var i=0
    if (!foldersNode[2])
    {
        for (i=5; i< foldersNode.length; i++)
            closeFolders(foldersNode[i])
    }
    foldersNode[0] = 0
    foldersNode[1] = 0
}

//recursive over the tree structure
//called by openbranch
function clickOnFolderRec(foldersNode, folderName)
{
    var i=0
    if (foldersNode[3] == folderName)
    {
        if (foldersNode[0])
            closeFolders(foldersNode)
        else
        {
            foldersNode[0] = 1
            foldersNode[1] = 1
        }
    } else
    {
        if (!foldersNode[2])
            
}
for (i=5; i<foldersNode.length; i++)
    clickOnFolderRec(foldersNode[i], folderName)
}

// called when the user clicks on a folder
function openBranch(branchName)
{
    clickOnFolderRec(foldersTree, branchName)
    if (branchName=="UniGuide" && foldersTree[0]==0)
        top.rightFrame.location="<?MIVAR>$WEB_HOME<?/MIVAR>?MIval=rfra_ins"
    timeOutId = setTimeout("redrawTree()",100)
}

// Draw all the tree when loading the doc.
function openAllFolders(foldersNode)
{
    // Opens all the folders of the tree structure
    var i=0

    if (!foldersNode[2])
    {
        for (i=5; i<foldersNode.length; i++)
            openAllFolders(foldersNode[i])
    }

    foldersNode[0] = 1
    foldersNode[1] = 0
}

// called after this html file is loaded
function initializeTree()
{
    var doc = top.leftFrame.window.document;

generateTree()
openAllFolders(foldersTree)
redrawTree()
}

var foldersTree = 0
var timeOutId = 0
generateTree()
/*sometimes when the user reloads the document Netscape 3.01 does not trigger
the onLoad event (!!!) */

// end hiding contents from old browsers -->
</script>
</HEAD>
<FRAMESET cols="200,*" onLoad='initializeTree()'>
  <FRAME marginwidth=1 marginheight=1 src="<?MIVAR>$WEB_HOME<?/MIVAR>?MIval=lfra_common" name="leftFrame">
  <FRAME marginwidth=1 marginheight=1 SRC="<?MIVAR>$WEB_HOME<?/MIVAR>?MIval=rfra_ins" name="rightFrame" frameborder="no">
B.1.2.2 form_ins

```
<html>
<head>
<SCRIPT LANGUAGE = "JavaScript">
<!-- No javascript

//global vars
var aobjCols = new Array(); // array of column objects

function obj_instance(intcolindex,strcolsource,strcolname,strcolhelp,bolcolmandatory)
{
  /* functions specifies attributes of a column object */
  this.index = intcolindex;
  this.source = strcolsource;
  this.name = strcolname;
  this.help = strcolhelp;
  this.mandatory = bolcolmandatory;

}

function aobjCols_init(intcolindex,strcolsource,strcolname,strcolhelp,bolcolmandatory)
{
  /* function defines a new column object instance and */
  /* adds it to the array of column objects */
  var obj= new obj_instance(intcolindex,strcolsource,strcolname,strcolhelp,bolcolmandatory);
  aobjCols[intcolindex]=obj;
  return (aobjCols);
}

function LF_clearstatus()
{
  /* clears the contents displayed on the status bar */
  self.status="";
}

function LF_showstatus(intcounter)
{
  /* function displays contextual help on the status bar when */
  /* a form object gains focus */
  self.status = aobjCols[intcounter].help;
}

function LF_showHelp(strHelp)
{
  /* function displays general help when a user clicks on the help button */
  var strMsg;
  strMsg=strHelp + "; ";
```
strMsg = strMsg + 
"\nPlease look at the contextual help shown on the status-bar."
;
alert(strMsg);
}

function LF_AllSpaces(strAux)
{
    /* functions test if the string contains all spaces */
    var inti;
    for(inti=0;inti<=strAux.length - 1;inti++) {
        if(strAux.charAt(inti) != " ")
            return false;
    }
    return true;
}

function LF_Reformat(strtext)
{
    /* function replaces carriage return and line feed chars by spaces */
    var strAux;
    var intpos;
    var inti;
    var str_appVersion;
    strAux = strtext;
    str_appVersion = navigator.appVersion;
    inti = 0;
    if(str_appVersion.indexOf("Win")!=-1) {
        intpos = strAux.indexOf('\r\n',0)
        while(intpos!=-1) {
            // to avoid infinite iteration
            inti++;
            if (inti==150) break;
            strAux = strAux.substring(0,intpos) + ' ' +
                strAux.substring(intpos + 2,strAux.length);
            intpos = strAux.indexOf('\r\n',intpos + 1);
        }
        return strAux;
    }
    else {
        // reformat \n' by ' ' for Unix
        intpos = strAux.indexOf('\n',0)
        while(intpos!=-1) {
            // to avoid infinite iteration
            inti++;
            if (inti==150) break;
            strAux = strAux.substring(0,intpos) + ' ' +
                strAux.substring(intpos + 1,strAux.length);
            intpos = strAux.indexOf('\n',intpos + 1);
        }
        return strAux;
    }
}
function LF_Replace_Chars(strtext)
{
    /* function replaces every occurrence of the char ' and " by ` */

    var strAux;
    var intpos;
    var inti;

    strAux=strtext;

    // Replace ' by`
    inti=0;
    intpos = strAux.indexOf('"',0);
    while(intpos!=-1) {
        // to avoid infinite iteration
        inti++;
        if (inti==150) break;
        strAux = strAux.substring(0,intpos) + ` +
                          strAux.substring(intpos + 1,strAux.length);
        intpos = strAux.indexOf('"',intpos + 1);
    }

    // now replace " by `
    inti=0;
    intpos = strAux.indexOf('"',0);
    while(intpos!=-1) {
        // to avoid infinite iteration
        inti++;
        if (inti==150) break;
        strAux = strAux.substring(0,intpos) + ` +
                          strAux.substring(intpos + 1,strAux.length);
        intpos = strAux.indexOf('"',intpos + 1);
    }

    return strAux;
}

function LF_tstDataIsOk()
{
    /* function tests that all input values are correct: */
    /* tests for mandatory columns, maximum length of */
    /* textarea objects, valid URL's and if valid */
    /* reformats values i.e replace char ' and " by ` */
    /* finally asks for submit confirmation */

    var intcounter;
    var strValue;
    var strSource;
    var strType;
    var intSelected;
    var strMsg;
    var bolReformated;

    for (intcounter=0; intcounter <= document.forms[0].v_object.length-1; intcounter++) {
        if(aobjCols[intcounter+1] == null) continue;
        strValue = document.forms[0].v_object[intcounter].value;
        ...
strType = document.forms[0].v_object[intcounter].type;

// test selected items for select lists
if(strType == "select-one") {
    intSelected = document.forms[0].v_object[intcounter].selectedIndex;
    // reassign list value because when cancelling submit for some
    // unexplained reasons it becomes ="
    strValue= document.forms[0].v_object[intcounter].options[intSelected].value;
    document.forms[0].v_object[intcounter].value = strValue;
}

if(LF_AllSpaces(strValue) && aobjCols[intcounter+1].mandatory){
    // test mandatory column valued
    alert(aobjCols[intcounter+1].name + " is mandatory!");
    return false;
}

if(strType == 'textarea' && strValue.length > 300){
    // test maximum length of textarea objects
    alert(aobjCols[intcounter+1].name + " not valid, max. length is 300 chars!");
    return false;
}

strSource=aobjCols[intcounter+1].source;

// Test if URL is correct
if(strSource.indexOf("url")!=-1) {
    if(strValue.indexOf("http://")==-1 && strValue.indexOf("ftp://") == -1)
    {
        alert("URL must include protocol: 'http://' or 'ftp://'!");
        document.forms[0].v_object[intcounter].value = "http://" + strValue;
        return false;
    }
    if(strValue.indexOf(".") == -1) {
        strMsg= "URL must be a valid Resource Address!
        (i.e. http://www.myuni.edu.au/mydir/mypage.htm)"
        alert(strMsg);
        return false;
    }
}

// Test if E-mail is correct
if(strSource.indexOf("_e_mail")!=-1) {
    if(strValue.length > 0 && strValue!='null' &&
    {strValue.indexOf("@")!=-1 || strValue.indexOf(".")==-1})
    alert("E-mail must include '@' and '!' or be empty or null!");
    return false;
}

if(strValue.length > 0 && strValue.indexOf("mailto:"!=-1){
    alert("Please do not include protocol in E-mail");
    document.forms[0].v_object[intcounter].value =
    strValue.substring(strValue.indexOf("mailto:")) + 7,strValue.length);
    return false;
}

/* default value for non mandatory col.=null */
/* textarea with nothing gives back an "e" */
if((LF_AllSpaces(strValue) ||(strType == 'textarea' && strValue == "e")) &&

Appendix B. Web-Datablade Applications and Embedded JavaScript
```javascript
aobjCols[intcounter+1].mandatory == 0)
    document.forms[0].v_object[intcounter].value = "null";

    // reformat textarea do not include '\n' or '\r\\n'
    bolReformatted = false;
    if(strType == 'textarea' &&
        document.forms[0].v_object[intcounter].value != "null") {
        document.forms[0].v_object[intcounter].value = LF_Reformat(strValue);
        bolReformatted = true;
    }

    // replace ' and " by `.
    if((strValue.indexOf('"')!=-1 || strValue.indexOf('"')!=-1) &&
        document.forms[0].v_object[intcounter].value != "null") {
        if(bolReformatted == true)
            strValue = document.forms[0].v_object[intcounter].value;
        document.forms[0].v_object[intcounter].value = LF_Replace_Chars(strValue);
    }
    return(confirm("Data seems to be correct! Submit URL, Are You Sure?");
}

// end of No Javascript -->
</script>
</HEAD>

<body>

<!-- Activate Error Handler -->
<?mierror err=ErrHandler1 tag=mivar>
<html>
<head><title>Error Page</title></head>
<body>
<h2>Error Page:</h2>
<hr>
<br>
<form>
    <table width=100% align=center>
        <tr>
            <td align=center>
                <input type=button name=btn_goback value="Back" onClick="history.back();">
            </td>
        </tr>
    </table>
</form>
</body>
</html>
<?/mierror>

<font face=arial size=2>
<table width=100%>
    <tr>
        <th bgcolor=gray align=left><font color=silver>UniGuide-->URL Submit Form</font>
    </th>
</tr>
</table>
```
<!-- Capture table information -->
<?mivar name=v_table_source default="club_association">$v_table_source<?/mivar>
<?misql sql="SELECT table_name, table_help
    FROM _tables t1
    WHERE t1.table_source = '$v_table_source';">
$('SETVAR,v_table_help,"$2")
<?/misql>

<!-- Define output parameters -->
<input type=hidden name=v_fields value=$2>
<tr><small>$1:</small><br>$3</td>
</tr>

<!-- Block is activated when form is submitted -->
<?miblock cond=$(AND,$(XST,$v_query),$(EQ,$v_query,yes))>
<?misql sql="SELECT count(*)
    FROM _columns
    WHERE table_source='$v_table_source' AND column_type LIKE
      '%$s%';"">$(SETVAR,v_search,$1)</misql>
<?mivar name=v_loffields separate="^">$v_fields<?/mivar>
<mivar name=v_lofvalues separate="^">$v_object</mivar>

<?miblock cond=$(=,$v_search,0)>
<!-- Does not include sets, simple insert -->
<!-- Insert data directly -->
<?mysqli sql="Return GF_simple_insert('$v_table_source','$v_loffields','$v_lofvalues');">$(SETVAR,$v_sqlinsert,$1)
<?/mysqli>
<?miblock>

<?miblock cond=$(!=,$v_search,0)>
<!-- Does include sets, complex insert -->
<!-- Search and Insert data directly -->
<?mysqli sql="Return GF_complex_insert('$v_table_source','$v_loffields','$v_lofvalues');">$(SETVAR,$v_sqlinsert,$1)
<?/mysqli>
<?miblock>
<?mysqli err=ErrHandler1 sql="$v_sqlinsert"><?/mysqli>

<script language="Javascript">
// give feedback to the user indicating the status of the submission

var stroid="<?mivar>$MI_OID<?/mivar>";
var strResult="<?mivar>$v_sqlinsert<?/mivar>";
if(strResult=="Return 0;" || strResult=="") alert("Error: URL has not been saved");
else {
  if(strResult.indexOf("INSERT")!=-1) {
    if(strresult=="0" && strResult.indexOf("UPDATE")==-1) {
      // simple query
      alert("Error: URL has not been saved");
    } else // insert OK
      alert("URL succesfully saved!");
  } else //is an update
    alert("URL succesfully updated!");
}

</script>

</miblock>
</body>
</html>

B.2 Query Web-Datablade Applications
B.2.1 Query Web-Datablade Applications: State transition diagram
Figure B2. Query Web-Datablade Applications: State transition diagram

B.2.2 Query Web-Datablade Applications: Source Code

B.2.2.1 form_squery

```html
<html>
<head>
<title>UniGuide - Simple Query Section</title>

<script>
//global vars
var aobjCols = new Array(); // array of column objects

function obj_instance(intcolindex, strcolsource, strcolname, strcolhelp) {

    /* specifies attributes of a new column object */

    this.index = intcolindex;
    this.source = strcolsource;
    this.name = strcolname;
    this.help = strcolhelp;
}

function aobjCols_init(intcolindex, strcolsource, strcolname, strcolhelp) {

```
/* creates a new column object and adds it to the array of column objects */

var obj = new obj_instance(intcolindex, strcolsource, strcolname, strcolhelp);
    aobjCols[intcolindex] = obj;
    return (aobjCols);
}

function LF_clearstatus()
{
    /* clears the contents displayed on the status bar */

    self.status="";
}

function LF_showstatus(intcounter)
{
    /* displays contextual help on the status bar */

    self.status = aobjCols[intcounter].help;
}

function LF_showHelp(strHelp)
{
    /* displays general help */

    var strMsg;

    strMsg = strHelp + ", ";
    strMsg = strMsg + "\nPlease look at the contextual help shown on the status-bar.";

    alert(strMsg);
}

function LF_AllSpaces(strAux)
{
    /* functions test if the string contains all spaces */

    var inti;

    for(inti=0;inti<strAux.length - 1;inti++) {
        if(strAux.charAt(inti) != " ")
            return false;
    }
    return true;
}

function LF_Tst_Qry_Values()
{
    /* Activated: onSubmit Form                                     */
    /* function tests if text box has incorrect values, blank only or empty */
    /* are not allowed, '*' means all values are allowed            */

    var intLimit = document.forms[0].v_object.length - 1;
    var intSelected;
    var intCounter;
var strValue;
var strType;
var strfld;
var bolchecked;

bolchecked=false;
for(intCounter=0;intCounter<=intLimit;intCounter++) {
    strValue = document.forms[0].v_object[intCounter].value;
    strType = document.forms[0].v_object[intCounter].type;
//test selected items for select lists
    if(strType == "select-one") {
        intSelected = document.forms[0].v_object[intCounter].selectedIndex;
        /* reassign list value because when cancelling submit for some
           unexplained reasons it becomes ="
        */
        strValue= document.forms[0].v_object[intCounter].options[intSelected].value;
        document.forms[0].v_object[intCounter].value = strValue;
    }
    if (strValue.length == 0 || LF_AllSpaces(strValue))
        document.forms[0].v_object[intCounter].value="";
    if(document.forms[0].v_includes[intCounter].checked) {
        // if a checkbox is checked, assign a value of 1 so it is sent to the
        // server when submitted
        bolchecked =true;
        document.forms[0].v_includes[intCounter].value="1";
        strfld = document.forms[0].v_fields[intCounter].value;
        document.forms[0].v_headers[intCounter].value= strfld;
    } else {
        document.forms[0].v_includes[intCounter].value="0";
        document.forms[0].v_includes[intCounter].checked = true;
    }
}
if(bolchecked==false) {
    alert("Please select at least one column!");
    return false;
}
return true;
}

function LF_mark_hyperlink(strcol)
{
    /* function generates a link for every URL value */

    var str_next_col;
    var str_arg;
    var strexpr1;
    var strexpr2;
    var bolnowrap;

    bolnowrap=false;
    str_next_col = strcol;
    if(str_next_col.length <=30)
        bolnowrap=true;

    if(bolnowrap==true)

str_arg = '<td nowrap>' + strcol + '</td>'
else
    str_arg = '<td>' + strcol + '</td>'

if(strcol.indexOf('http://') != -1 || strcol.indexOf('ftp://') != -1) {
    if(bolnowrap == true)
        str_arg = '<td nowrap>' + '<a href=' + strcol + ' target=_top>' + strcol + '</a></td>'
    else
        str_arg = '<td>' + '<a href=' + strcol + ' target=_top>' + strcol + '</a></td>'
    return str_arg;
}

if(strcol.indexOf('@') != -1 && strcol.indexOf('.') != -1) {
    // It's a valid e-mail address that contains "@" and "."
    if(bolnowrap == true)
        str_arg = '<td nowrap>' + '<a href=mailto:' + strcol + '>' + strcol + '</a></td>'
    else
        str_arg = '<td>' + '<a href=mailto:' + strcol + '>' + strcol + '</a></td>'
    return str_arg;
}

return str_arg;
}</script>
</head>

<body>
<!-- Activate Error Handler -->
<?mierror err=ErrHandler1 tag=mivar>
<html>
<head><title>Error Page</title></head>
<body>
<h2>Error Page:</h2>
<hr>
<br>
<form>
<table width=100% align=center>
<tr align=center>
<input type=button name=btn_goback value="Back" onClick="history.back();"/>
</tr>
</table>
</form>
</body>
</html>
<?/mierror>

<font face=arial size=2>
<table width=100%>
<tr><th bgcolor=gray align=left><font color=silver>UniGuide-> URL Simple Query
<center><strong>Section</strong></center>

<!-- Capture table name and help information -->

<?mivar name=v_table_source default='club_association'?>v_table_source<?/mivar>

<?mysql sql="SELECT table_name, table_help
    FROM _tables t1
    WHERE t1.table_source = '&v_table_source';">
    <strong>$1</strong>
    $(SETVAR,vtable_help,"$2")
</mysql>

<!-- Block is activated when a form has been submitted -->

<?miblock cond=$(AND,\$(XST,$v_genquery),\$(EQ,$v_genquery,yes))>
    <?mivar name=v_loffields separate="^">$v_fields<?/mivar>
    <?mivar name=v_lofincludes separate="^">$v_includes<?/mivar>
    <?mivar name=v_lofoptions separate="^">$v_options<?/mivar>
    <?mivar name=v_lofvalues separate="^">$v_object<?/mivar>
    <?mivar name=v_lofheaders separate="^">$v_headers<?/mivar>

<?mysql sql="SELECT count(*)
    FROM _columns
    WHERE table_source='&v_table_source' AND column_type LIKE '%$s$';">
    $(SETVAR,v_search,$1)<?mysql>
</mysql>

<?miblock cond=$(=,$v_search,0)>
    <!-- Does not include sets, simple query -->
    <?mysql err=errHandler1 sql="Return GF_squery_simple('&v_table_source',
        '&v_loffields', '&v_lofincludes', '&v_lofoptions', '&v_lofvalues');">
        $(SETVAR,v_sqlquery,$1)
    <?mysql>
</miblock>

<?miblock cond=$(!=,$v_search,0)>
    <!-- Does include sets, complex query -->
    <?mysql err=errHandler1 sql="Return GF_squery_complex('&v_table_source',
        '&v_loffields', '&v_lofincludes', '&v_lofoptions', '&v_lofvalues');">
        $(SETVAR,v_sqlquery,$1)
    <?mysql>
</miblock>

</miblock>
</center>

<!-- When form is submitted previous to send it to the server -->

<!-- values are tested and reformatted -->

<FORM method=post action='<mivar>$WEB_HOME</mivar> onSubmit="return LF_Tst_Qry_Values()">
    <input type=hidden name=MIval value=form_squery>
    <?miblock cond=$(XST,$v_table_source)>
        <br>
        <table border align=right>
            <tr><th>Column</th><th>Option</th><th>Search Value</th></tr>
            <?mysql err=ErrHandler1 sql="SELECT c.column_source, c.column_name,
c.squery_column_number,c.column_data_type, c.squery_object_type, c.column_help
FROM _columns c
WHERE c.table_source ="$v_table_source"
ORDER BY c.squery_column_number;"
</input type=hidden name=v_fields value=$1>
<input type=hidden name=v_headers value="">
<tr>
<th nowrap align=right>$2
<input type=checkbox name=v_includes checked></th>
</tr>

<script language = "JavaScript">
// generate dynamically options select list
// for text or timestamp values the LIKE option is added

var str_data_type="$4";
var str_selection="";

str_selection="<td><select name=v_options size=1>";
str_selection= str_selection + "<option value='<'><";
str_selection= str_selection + "<option value='='><=";
str_selection= str_selection + "<option selected value='='>=''=";
str_selection= str_selection + "<option value='>='>>=";
str_selection= str_selection + "<option value='>'>>";
if (str_data_type == "text" || str_data_type == "timestamp")
str_selection= str_selection + "<option value=' LIKE '>LIKE";
str_selection= str_selection + "</option>">
str_selection= str_selection + "</select>";
str_selection= str_selection + "</td>";

document.write(str_selection);
</script>

</table>
</table>
<input type=hidden name=v_genquery value=yes>
<br clear=all><br>
<table align=right>
<tr>
<td><input type=reset value=" Undo "></td>
<td><input type=submit value=" Execute "></td>
<td><input type=button value=" Help "
onclick="LF_showHelp(<?mivar>'$vtable_help'<?/mivar})">"</td>
</tr>
</table>
\$\{IF,\$\{AND,\$\{>=,$MI_CURRENTROW,$v_Begin\},\$\{<,$MI_CURRENTROW,$\{+,$v_Begin,25\}\}\},\$\{SETVAR,\$vRow,"\"str"\"@td bgcolor=white\align=center\"%small\$MI_CURRENTROW%\small\"%small\"/td\"\%}\}\$\}\{IF,\$\{AND,\$\{>=,$MI_CURRENTROW,$v_Begin\},\$\{<,$MI_CURRENTROW,$\{+,$v_Begin,25\}\}\},\$\{SETVAR,\$vRow,\$\{REPLACE,\"$vRow\",\"\",\"\"\}\}\}\$\{IF,\$\{AND,\$\{>=,$MI_CURRENTROW,$v_Begin\},\$\{<,$MI_CURRENTROW,$\{+,$v_Begin,25\}\}\},\$\{SETVAR,\$vRow,\$\{REPLACE,\"$vRow\",\"","\"\"\}\}\}\$\{IF,\$\{AND,\$\{>=,$MI_CURRENTROW,$v_Begin\},\$\{<,$MI_CURRENTROW,$\{+,$v_Begin,25\}\}\},\$vRow\}\<SCRIPT LANGUAGE = "JavaScript">// output only row/set instances that are in the sliding window range // for every URL value generate a link
var str_selection ="";
var str_next_col="";
var intRow = parseInt('$MI_CURRENTROW');
var intBegin = parseInt('$v_Begin');
var intLimit = parseInt('$\{+,$v_Begin,25\}');

// We have a limit of 20 col. on each entity
str_next_col="$1";
if(str_next_col =="" && intRow >=intBegin && intRow < intLimit)
str_selection = str_selection + LF_mark_hyperlink(str_next_col);

str_next_col="$2";
if(str_next_col =="" && intRow >=intBegin && intRow < intLimit)
str_selection = str_selection + LF_mark_hyperlink(str_next_col);

(SKipping Code ….)
------------------------------------------------------------------
str_next_col="$20";
if(str_next_col =="" && intRow >=intBegin && intRow < intLimit)
str_selection = str_selection + LF_mark_hyperlink(str_next_col);

document.write(str_selection);
</SCRIPT>
</tr>
</table>

<! Block defines link to first/previous row/set instances -->
<#miblock cond="$\{>,$v_Begin,25\}">
<#mivar name=v_Label>$\{IF,$\{=,$\{-,$v_Begin,25\},1\},"First 25","Previous 25"\}$</mivar>
<#mivar><a href=$WEB_HOME?MIval=form_squery&v_Begin=$\{-,$v_Begin,25\}&v_Walk="yes"&v_lofheaders=$v_lofheaders&v_genquery="no"&v_sqlquery=$(REPLACE,"$v_sqlquery","","")>$v_Label</a></mivar>
</#miblock>

<! Block defines link to next/last row/set instances -->
<#miblock cond="$\{>,$MI_ROWCOUNT,$\{+,$v_Begin,25\}\}">
<#mivar name=v_Label>$\{IF,$\{<,$MI_ROWCOUNT,$\{+,$v_Begin,49\},"Last 25","Next 25"\}$</mivar>
</#miblock>
B.3 Query Filtering Web-Datablade Applications

B.3.1 Query Filtering Web-Datablade Applications: State transition diagram

![State transition diagram](image)

**Figure B3.** Query Filtering Web-Datablade Applications: State transition diagram
B.3.2 Query Filtering Web-Datablade Applications: Source Code

B.3.2.1 free_sql

```html
<html>
<head><title>UniGuide → Free SQL Section</title>

<script language="javascript">

//global vars
astrcolumns = new Array(); /* array of columns */
intcolcounter = 0; /* number of columns */
strtbl_before=''; /* auxiliar string that stores the previous table */

function LF_AllSpaces(strAux)
{ /* functions test if the string contains all spaces */
    var int;
    for(inti=0;inti<strAux.length - 1;inti++) {
        if(strAux.charAt(inti) != " ")
            return false;
    }
    return true;
}

function LF_feed_column_list(lstObj)
{ /* When end-users click on the table select list this */
    /* function feeds the column select list dynamically */
    /* from the array of columns, searching the columns */
    /* associated to the table */
    var intSelectedIndex;
    var inti, intj, intk;
    var strtable;
    var bolfirst;

    // capture table select list option index
    intSelectedIndex = lstObj.selectedIndex;
    if (intSelectedIndex== -1) return;
```
// capture selected option table select list
strtable = lstObj.options[intSelectedIndex].value;

inti=0;
bolfirst=true;
for(intj=0;intj<astrcolumns.length;intj++) {
  if(astrcolumns[intj].substring(0,strtable.length + 1) == (strtable + '.')) {
    // feed the column select list when table column is detected
    bolfirst = false;
    if(document.frm0.v_columns.options[inti] == null) continue;
    document.frm0.v_columns.options[inti].value = astrcolumns[intj];
    document.frm0.v_columns.options[inti].text = astrcolumns[intj];
    inti++;
  }
  if(astrcolumns[intj].substring(0,strtable.length + 1) != (strtable + '.') &&
    bolfirst==false)
    // When the last table column position + 1 has been detected exit loop
    break;
}
for(intk=inti;intk<document.frm0.v_columns.options.length;intk++) {
  // Fill with blanks the rest options of the column select list
  document.frm0.v_columns.options[intk].value = "";
  document.frm0.v_columns.options[intk].text = "";
}
}

function LF_get_selected_value(listObj)
{
  /* function gets the currently selected option of a */
  /* given select list and outputs it to the textarea */
  var strValue;
  var intSelectedIndex;
  var strsql;
  intSelectedIndex = listObj.selectedIndex;
  strValue = listObj.options[intSelectedIndex].value;
  // paste value into textarea
  strsql = document.frm0.v_sql.value;
  document.frm0.v_sql.value = strsql + strValue;
}

function LF_ltrim(strAux)
{
  /* function filters spaces on the left hand-side of a string */
  var intlength;
  var inti;
  var strlefttrimmed;
  var bolbeforeletter;
  strlefttrimmed="";
  bolbeforeletter=true;
  intlength = strAux.length;
  for(inti=0;inti<intlength;inti++) {
    if(strAux.charAt(inti)==" " && bolbeforeletter) continue;
    else {
      strlefttrimmed = strlefttrimmed + strAux.charAt(inti);
      bolbeforeletter=false;
    }
  }
  return strlefttrimmed;
}

function LF_put_text(strAux)
{
  /* function outputs SQL clauses and other options to a textarea */
  var strsql;
  strsql=document.frm0.v_sql.value;
  if(strAux=='"' || strAux=='\n' || strAux=='unique' || strAux=='distinct') {
    document.frm0.v_sql.value = strsql + strAux;
  }
function LF_test_query()
{
/* function tests if a given query is valid with valid */
/* SQL SELECT statements and if valid reformat the */
/* statement into a valid format for the external */
/* query filtering module GF_free_sql */
var strsql,strAux;
var intpos;
strsql=document.frm0.v_sql.value;
if(LF_AllSpaces(strsql) || strsql.length==0) {
    alert("Empty Query!");
document.frm0.v_sql.value = "SELECT ";
    return false;
}
strsql= LF_ltrim(strsql);
document.frm0.v_sql.value = strsql;
strAux = strsql.toLowerCase();
if(strAux.indexOf("update")!=-1 || strAux.indexOf("delete")!=-1 ||
    strAux.indexOf("insert")!=-1 || strAux.indexOf("create")!=-1 ||
    strAux.indexOf("alter")!=-1 || strAux.indexOf("return")!=-1) {
    alert("Operation not allowed! Only Valid Select statements");
    return false;
}
if(strAux.indexOf("select") ==-1) {
    alert("Please include the word Select in the SQL statement");
document.frm0.v_sql.value = 'SELECT ' + document.frm0.v_sql.value;
    return false;
}
if(strAux.indexOf("from") ==-1) {
    alert("Please include the word FROM in the SQL statement");
    return false;
}
if(strAux.indexOf(";") !=-1) {
    alert("Please do not include ';' in the SQL statement");
    return false;
}
// change NL for spaces
strsql = LF_Reformat_v_sql(strsql)
//Query is correct test if it contains ' and change for`
// so it does not corrupt the SQL statement
strAux=strsql;
intpos = strAux.indexOf("'" );
while(intpos!=-1) {
    strAux = strAux.substring(0,intpos) + '^' +
    strAux.substring(intpos + 1,strAux.length);
    intpos = strAux.indexOf("'", intpos + 1);
// output resulting format
document.frm0.v_sql.value = strAux;
    return true;
}
function LF_reformat(strsql)
{
/* function replaces the char ^ for ' */
/* The internal representation of a query */
/* is formed by ^ chars so the query is not corrupted */
/* so we must reformat again for end-users into ' chars */

var strAux;
var intpos;
var inti;

strAux=strsql;
// reformat ^ by '

intpos = strAux.indexOf("^")
inti=0;
while(intpos!=-1) {
    // to avoid infinite iteration
    inti++;
    if (inti==150) break;
    strAux = strAux.substring(0,intpos) + '\'' +
    strAux.substring(intpos + 1,strAux.length);
    intpos = strAux.indexOf("^",intpos + 1);
}
return strAux;

}

function LF_mark_hyperlink(strcol)
{/*
    function generates a link if the value contains a URL */

var str_next_col;
var str_arg;
var strexpr1;
var strexpr2;
var bolnowrap;

bolnowrap=false;
str_next_col = strcol;
if(str_next_col.length <=30)
bolnowrap=true;

if(bolnowrap==true)
    str_arg= '<td nowrap>' + strcol + '</td>';
else
    str_arg= '<td>' + strcol + '</td>';

if(strcol.indexOf('http://')!=-1 || strcol.indexOf('ftp://')!=-1)
    // http or ftp URL
    if(bolnowrap==true)
        str_arg= '<td nowrap>' + '<a href=' + strcol + '>' + strcol + '</a></td>';
    else
        str_arg= '<td>' + '<a href=' + strcol + '>' + strcol + '</a></td>';
    return str_arg;
}

if(strcol.indexOf('@')!=-1 && strcol.indexOf('.')!=-1)
    if(bolnowrap==true)
        str_arg= '<td nowrap>' + '<a href=mailto:' + strcol + '>' + strcol + '</a></td>';
    else
        str_arg= '<td>' + '<a href=mailto:' + strcol + '>' + strcol + '</a></td>';
    return str_arg;
}

return str_arg;
}

function LF_Replac
{/
    /* function replace all occurrences of a given string in the */
    /* SQL statement */

var str_find;
var str_replace;
var strsql;
var inti, intpos;

strsql=document.frm0.v_sql.value;
str_find='';
str_find = prompt('Enter string to Find and Replace','');
if(str_find == null || str_find == '') return;

str_replace = prompt('Enter string to Replace string: "' + str_find + '" with','');
if(str_replace == null) return;

inti=0;
intpos = strsql.indexOf(str_find,0);
while (intpos!=-1) {
    // to avoid infinite iteration
    inti++;
    if (inti==150) break;
    strsql = strsql.substring(0,intpos) + str_replace +
    strsql.substring(intpos + str_find.length,strsql.length);
    intpos = strsql.indexOf(str_find,intpos + str_find.length);
}

document.frm0.v_sql.value = strsql;
}

function LF_Write(strAux, intRow)
{
    /* function provides feedback for slow browsers indicating */
    /* the currently row/set instance being retrieved */
    var str_tr= strAux + '</tr>';
    document.write(str_tr);
    self.status = 'Currently Getting Row/Set No.: ' + intRow;
}

function LF_Reformat_v_sql(strtext)
{
    /* function replaces all carriage return or line feed chars */
    /* into spaces depending the platform Win or Unix */
    var strAux;
    var intpos;
    var inti;

    strAux=strtext;
    str_appVersion = navigator.appVersion;
    inti=0;
    if(str_appVersion.indexOf("Win")!=-1) {
        intpos = strAux.indexOf('\n');
        while(intpos!=-1) {
            // to avoid infinite iteration
            inti++;
            if (inti==150) break;
            strAux = strAux.substring(0,intpos) + ' ' +
            strAux.substring(intpos + 2,strAux.length);
            intpos = strAux.indexOf('\n',intpos+2);
        }
        return strAux;
    }
    else {
        // reformat 'n' by ' ' for Unix
        intpos = strAux.indexOf('n');
        while(intpos!=-1) {
            // to avoid infinite iteration
            inti++;
            if (inti==150) break;
            strAux = strAux.substring(0,intpos) + ' ' +
            strAux.substring(intpos + 1,strAux.length);
            intpos = strAux.indexOf('n',intpos + 1);
        }
        return strAux;
    }
}

function LF_Create_Select_Option(str_desc, str_query)
{
    /* function generates a select option for the sample queries select list */
var str_Aux;
str_Aux = '<option value="' + LF_reformat(str_query) + '">' + str_desc;
return str_Aux;
}

function LF_Help()
{
var str_Msg;
str_Msg = "The 1st row of buttons outputs commands to the textarea\n} + "The second row of buttons perform specific operations:\n} + "Cls\t\t\t-> Clears the text area\n} + "Replace\t\t\t-> Replaces every occurrence of a string by another string\n} + "Table\t\t\t-> Outputs the currently selected table option\n} + "Column\t\t\t-> Outputs the currently selected column option\n} + "Function/Op.\t\t\t-> Outputs the currently selected function/operator option\n} + "Sample\t\t\t-> Outputs the currently selected sample query option\n} + "Undo\t\t-> Cancels last changes\n} + "Execute\t\t-> Executes SQL statement (Please do not include semicolon)\n} + "Help\t\t-> Displays this help information";
alert(str_Msg);
}
</script>

<!-- Activate Error Handler -->
</head>
<body>
<?mierror err=ErrHandler1 tag=mivar>
<html>
<head><title>Error Page</title></head>
<body>
<h2>Error Page:</h2>
<hr>
<strong>Error Code($MI_ERRORCODE):</strong>
$(IF,$(XST,$MI_ERRORMSG),$MI_ERRORMSG,"Unknown Error")
<hr>
<br>
<form>
<table width=100% align=center>
<tr>
<td align=center>
<input type=button name=btn_goback value="Back" onClick="history.back();">
</td>
</tr>
</table>
</form>
</body>
</html>
<?/mierror>

<!-- When form is submitted, 1st test query before sending info to server -->
<form name="frm0" method=post action="<?mivar>$WEB_HOME<?/mivar>" onSubmit="return LF_test_query()">
<table align=center>
<tr>
<font face=arial size=2>
<th background=gray align=left colspan=3>
<font color=silver>UniGuide - Free SQL</font>
</th>
</tr>
</table>
</form>
</body>
</html>

<script language="javascript">
// generate dynamically from SQL statement table select list
var strselect='';
strselect = '<select name=v_tables size=3 >' + strselect + '  
strselect= strselect + '    <option value="' + LF_reformat(str_query) + '">' + str_desc;
return str_Aux;
}

function LF_Help()
{
var str_Msg;
str_Msg = "nThe 1st row of buttons outputs commands to the textarea\n" + "The second row of buttons perform specific operations:\n" + "Cls\t\t\t-> Clears the text area\n" + "Replace\t\t\t-> Replaces every occurrence of a string by another string\n" + "Table\t\t\t-> Outputs the currently selected table option\n" + "Column\t\t\t-> Outputs the currently selected column option\n" + "Function/Op.\t\t\t-> Outputs the currently selected function/operator option\n" + "Sample\t\t\t-> Outputs the currently selected sample query option\n" + "Undo\t\t-> Cancels last changes\n" + "Execute\t\t-> Executes SQL statement (Please do not include semicolon)\n" + "Help\t\t-> Displays this help information";
alert(str_Msg);
}
</script>

<!-- Activate Error Handler -->
</head>
<body>
<?mierror err=ErrHandler1 tag=mivar>
<html>
<head><title>Error Page</title></head>
<body>
<h2>Error Page:</h2>
<hr>
<strong>Error Code($MI_ERRORCODE):</strong>
$(IF,$(XST,$MI_ERRORMSG),$MI_ERRORMSG,"Unknown Error")
<hr>
<br>
<form>
<table width=100% align=center>
<tr>
<td align=center>
<input type=button name=btn_goback value="Back" onClick="history.back();">
</td>
</tr>
</table>
</form>
</body>
</html>
<?/mierror>

<!-- When form is submitted, 1st test query before sending info to server -->
<form name="frm0" method=post action="<?mivar>$WEB_HOME<?/mivar>" onSubmit="return LF_test_query()">
<table align=center>
<tr>
<font face=arial size=2>
<th background=gray align=left colspan=3>
<font color=silver>UniGuide - Free SQL</font>
</th>
</tr>
</table>
</form>
</body>
</html>

<script language="javascript">
// generate dynamically from SQL statement table select list
var strselect='';
strselect = '<select name=v_tables size=3 >' + strselect + '  
strselect= strselect + '    <option value="' + LF_reformat(str_query) + '">' + str_desc;
strselect = strselect + '<option selected value="university">university';

<?mssql sql="SELECT table_source from _tables
WHERE table_source <> 'university';">
strselect = strselect + '<option value="' + '$1' + '">' + '$1';
</mssql>
strselect= strselect + '</select>'
document.write(strselect);
</script>
</td>
</small><b>Columns:</b>

// generate dynamically column select list
var strselect='';
var intcounter;

strselect = strselect + '   
    
strselect = strselect + '</select>'
document.write(strselect);

</td>
</tr>

<tr><td colspan=2>
<small><b>Columns:</b></small>
</td>
</tr>

</table>

<script language="javascript">
// generate dynamically column select list
var strselect='';
var intcounter;

strselect = strselect + '   
    
for(intcounter=1;intcounter<18;intcounter++) {
    strselect = strselect + '</select>'
document.write(strselect);
}
</script>
</td>
</tr>

<tr><td>
<small><b>Functions/Operators:</b></small>
</td>
</tr>

<!-- Function/operators select list -->
<select name=v_Functions_operators size=2>
<option selected value="">Operators - </option>
<option value="-&gt;">-&gt;</option>
<option value="<"><</option>
<option value="=<">=<</option>
<option value="=>">=></option>
<option value=">>">>></option>
<option value="LIKE">LIKE</option>
<option value="AND">AND</option>
<option value="OR">OR</option>
<option value="NOT">NOT</option>
<option value="avg()">avg()</option>
<option value="count()">count()</option>
<option value="avg()">avg()</option>
<option value="min()">min()</option>
<option value="sum()">sum()</option>
<option value="length()">length()</option>
<option value="upper()">upper()</option>
<option value="Binary Text Functions - 
    
<option value="Concat( , )">Concat( , )</option>
<option value="position( , )">position( , )</option>
<option value="RegexEQ( , )">RegexEQ( , )</option>
<option value="RegexNE( , )">RegexNE( , )</option>
<option value="Timestamp Functions - 
    
<option value="tstamp_to_date()">tstamp_to_date()</option>
<option value="tstamp_to_time()">tstamp_to_time()</option>
<option value="System Functions - 
    
<option value="NonNullValue()">NonNullValue()</option>
<option value="NullValue()">NullValue()
</select>
Sample Queries:

<!-- Sample queries select list -->

<script language="javascript">
// generate dynamically from SQL statement sample queries select list

var str_Select;
str_Select = '<select name=v_Samples size=2>
<option selected value=""> General Queries </option>

<?misql sql="SELECT ID, _desc, query FROM query_samples ORDER by ID;">
str_Select = str_Select + LF_Create_Select_Option("$2","$3");
<?/misql>

str_Select = str_Select + '</select>

document.write(str_Select);
</script>

</td>
</tr>
<tr>
<td colspan=3 align=left>
<small><b>SQL Select Statement:</b></small><br>
Select column, [function(column)] From table [Where condition] [Group By column] [Order By column]
</td>
</tr>
<tr>
<!-- Form buttons 1st row -->

<td colspan=3 align=left>
<input type="button" name="btn_get_select" value="Select" onClick="LF_put_text('SELECT')">
<input type="button" name="btn_get_unique" value="unique" onClick="LF_put_text('unique')">
<input type="button" name="btn_get_distinct" value="distinct" onClick="LF_put_text('distinct')">
<input type="button" name="btn_get_from" value="From" onClick="LF_put_text('FROM')">
<input type="button" name="btn_get_where" value="Where" onClick="LF_put_text('WHERE')">
<input type="button" name="btn_get_group" value="Group By" onClick="LF_put_text('GROUP by')">
<input type="button" name="btn_get_having" value="Having" onClick="LF_put_text('HAVING')">
<input type="button" name="btn_get_order" value="Order By" onClick="LF_put_text('ORDER by')">
<input type="button" name="btn_get_Union" value="Union" onClick="LF_put_text('UNION')">
</td>
</tr>
<tr>
<!-- Text area SQL statements -->

<textarea name=v_sql rows=5 cols=72 wrap=physical>"SELECT "</textarea>

<?mivar name=v_sqlAux> $(IF,$(XST,$v_sql),"$v_sql")</textarea>

<script language="javascript">
// retrieve SQL statement, reformat it and output on the text area
var strsql;
strsql=document.frm0.v_sql.value;
document.frm0.v_sql.value = LF_reformat(strsql);
</script>
</td>
</tr>
<tr>
<!-- Form buttons 2nd row -->

<td colspan=3 align=left>
<input type="button" name="btn_clear" value="Cls" onClick="document.frm0.v_sql.value=''">
<input type="button" name="btn_get_return" value="Replace" onClick="LF_Replace()">
<input type="button" name="btn_get_table" value="Table" onClick="LF_get_selected_value(document.frm0.v_tables)">
<input type="button" name="btn_get_column" value="Column" onClick="LF_get_selected_value(document.frm0.v_columns)">
<input type="button" name="btn_get_function_operator" value="Function/Op." onClick="LF_get_selected_value(document.frm0.v_Functions_operators)">
</td>
</tr>
<input type="button" name="btn_get_sample" value="Sample" onClick="LF_get_selected_value(document.frm0.v_Samples)" />
<input type="reset" name="btn_undo" value="Undo">
<input type="submit" name="btn_execute" value="Execute" onClick="LF_Help()">
</td>
</tr>
</table>

<!-- Ouput parameters -->
<input type=hidden name=v_query value="yes">
<input type=hidden name=MIval value=free_sql>
</form>

<!-- Generate array of columns -->
<?misql sql="SELECT table_source, column_source, squery_column_number
FROM _columns
WHERE column_type <> 'sB'
ORDER By table_source,squery_column_number;">
<script language="javascript">
// when loaded dynamically feed array of columns
LF_feed_array("$1","$2");
</script>
<?/misql>

<!-- Block is activated when form is submitted or pressing sliding -->
<!-- windows link -->
<?miblock cond=$(OR,$(XST,$v_query),$v_Walk)>
<?misql err=ErrHandler1 sql="Return GF_free_sql('$v_sql');">
$(SETVAR,v_Columnlist,$1)
<?/misql>
<?mivar name=v_position>$(POSITION,"$v_Columnlist","^^No Errors")<?/mivar>

<!-- Block activated when errors -->
<?miblock cond=$(=,$v_position,0)>
<?mivar name=verror delimit="^^ replace=" ": "">$v_Columnlist<?/mivar>
<br,strong><?mivar>$verror<?/mivar></strong>
<?/miblock>

<!-- Block activated when no errors -->
<?miblock cond=$(>,$v_position,0)>
<table align=center width=100%>
<tr>
<th bgcolor=gray align=left colspan=3>
<font color=silver>UniGuide-Free SQL Query Results</font>
</th>
</tr>
</table>

<!-- Filter the no errors part and retain column list values -->
<?mivar name=v_Header delimit="^^No Errors" Replace="">$v_Columnlist
<?/mivar>

<!-- Reformat particular column header values -->
<?miblock cond=$(>,$(POSITION,"$v_Header","_desc"),0)>
<?mivar name=vCatDesc><br><pre>cat_desc$(STRFILL, ,69)</pre><?/mivar>
<?mivar name=v_Header delimit="cat_desc" Replace="$vCatDesc">$v_Header
<?/mivar>
<?mivar name=vCourseDesc><br><pre>course_desc$(STRFILL, ,68)</pre><?/mivar>
<?mivar name=v_Header delimit="course_desc" Replace="$vCourseDesc">$v_Header
<?/mivar>
<?mivar name=vCourseUnitDesc><br><pre>cunit_desc$(STRFILL, ,67)</pre><?/mivar>
<?mivar name=v_Header delimit="cunit_desc" Replace="$vCourseUnitDesc">$v_Header
<?/mivar>
```
<tmivar name=vPubDesc><br><pre>pub_desc$(STRFILL, ,69)</pre><?/tmivar>
<tmivar name=proDesc><br><pre>pro_desc$(STRFILL, ,69)</pre><?/tmivar>
<tmivar name=Desc><br><pre>_desc$(STRFILL, ,72)</pre><?/tmivar>
<tmivar name=Stud><br><pre>student_research_interests$(STRFILL, ,51)</pre><?/tmivar>
<tmivar name=Staff><br><pre>staff_research_interests$(STRFILL, ,53)</pre><?/tmivar>
<table border>
<tr bgcolor=white>
<th nowrap>Row No.</th>
<th><?tmivar>$v_Header<?/tmivar></th>
</tr>
</table>
```
<!-- We have a limit of 30 col. on SQL query -->

str_next_col="$1";
if(str_next_col == "" && intRow >= intBegin && intRow < intLimit)
    str_selection = str_selection + LF_mark_hyperlink(str_next_col);

str_next_col="$2";
if(str_next_col == "" && intRow >= intBegin && intRow < intLimit)
    str_selection = str_selection + LF_mark_hyperlink(str_next_col);

(Skipping Code...)

--------------------------------------------------------------------------------------------------

str_next_col="$30";
if(str_next_col == "" && intRow >= intBegin && intRow < intLimit)
    str_selection = str_selection + LF_mark_hyperlink(str_next_col);
if(intRow >= intBegin && intRow < intLimit)
    LF_Write(str_selection, intRow);

</SCRIPT>
</table>
<br>

<?/miblock><?/miblock><?/miblock>

B.4 Meta tag generator Web-Datablade Application
B.4.1 Meta tag generator Web-Datablade Application: State transition diagram
Figure B4. Meta tag generator Web-Datablade Application: State transition diagram

B.4.2 Meta tag generator Web-Datablade Application: Source Code

B.4.2.1 form_meta

```html
<html>
<head>
<script LANGUAGE = "JavaScript">

// global vars
var intIndex=0; // counter of number of column objects
var aobjCols = new Array(); // array of column objects
var str_Table=""; // stores table name value
var intLimit=0; // number of total columns

function obj_instance(intcolindex,strcolsource,strcolname,strcolhelp,bolcolmandatory)
{
    /* function specifies column object attributes */
    this.column_number = intcolindex;
    this.source = strcolsource;
    this.name = strcolname;
    this.help = strcolhelp;
    this.mandatory = bolcolmandatory;
}

function aobjCols_init(intcolindex,strcolsource,strcolname,strcolhelp,bolcolmandatory)
{
    /* function creates a new column object instance */
    /* adds it to the array of column objects */

    var obj = new
    obj_instance(intcolindex,strcolsource,strcolname,strcolhelp,bolcolmandatory);
    aobjCols[intIndex]= obj;

    intIndex++;
```
function LF_clearstatus()
{
    /* clears the contents displayed on the status bar */
    self.status="";
}

function LF_showstatus(intCounter)
{
    /* display contextual help when a form object gains focus */

    var inti;

    // find the column number to display help
    for(inti=0;inti<=intLimit;inti++) {
        if(aobjCols[inti] == null) continue;
        if(aobjCols[inti].column_number == intCounter) {
            self.status = aobjCols[inti].help;
            break;
        }
    }
}

function LF_showHelp(strHelp)
{
    /* function shows general help */

    var strMsg;

    strMsg=strHelp + ". ";
    strMsg=strMsg + "\nPlease look at the contextual help shown on the status-bar.";

    alert(strMsg);
}

function LF_AllSpaces(strAux)
{
    /* functions test if the string contains all spaces */

    var inti;

    for(inti=0;inti<=strAux.length - 1;inti++) {
        if(strAux.charAt(inti) != " ")
            return false;
    }
    return true;
}

function LF_Reformat(strtext)
{
    /* function replaces all carriage return and line feed chars by spaces */
    return (aobjCols);
}
var strAux;
var intpos;
var inti;
var str_appVersion;

strAux=strtext;
str_appVersion = navigator.appVersion;
inti=0;
if(str_appVersion.indexOf("Win")!==-1) {  
  intpos = strAux.indexOf('\r\n',0)  
  while(intpos!=-1) {  
    // to avoid infinite iteration  
    inti++;
    if (inti==150) break;
    strAux = strAux.substring(0,intpos) + ' ' +
     strAux.substring(intpos + 2,strAux.length);
    intpos = strAux.indexOf('\r\n',intpos + 1);
  }
  return strAux;
}
else {  
  // reformat '\n' by ' ' for Unix  
  intpos = strAux.indexOf('\n',0)  
  while(intpos!=-1) {  
    // to avoid infinite iteration  
    inti++;
    if (inti==150) break;
    strAux = strAux.substring(0,intpos) + ' ' +
     strAux.substring(intpos + 1,strAux.length);
    intpos = strAux.indexOf('\n',intpos + 1);
  }
  return strAux;
}
}

function LF_Replace_Chars(strtext)  
{  
  /* function replaces every occurrence of the chars ' and " by ` */  
  var strAux;
  var intpos;
  var inti;

  strAux=strtext;

  // Replace ' by `  
  inti=0;
  intpos = strAux.indexOf('"',0);
  while(intpos !=-1) {  
    // to avoid infinite iteration  
    inti++;
    if (inti==150) break;
    strAux = strAux.substring(0,intpos) + '"' +
     strAux.substring(intpos + 1,strAux.length);
    intpos = strAux.indexOf('"',intpos + 1);
  }

  // now replace " by `
inti=0;
intpos = strAux.indexOf("\",0);
while(intpos !=-1) {
// to avoid infinite iteration
inti++;
if (inti==150) break;
strAux = strAux.substring(0,intpos) + ' ' +
strAux.substring(intpos + 1,strAux.length);
intpos = strAux.indexOf("\",intpos + 1);
}
return strAux;
}

function LF_tstDataIsOk()
{
/* function tests if column values are correct */
/* tests for mandatory column, max. length form */
/* objects, valid URLs and reformats values */
var intcounter;
var strValue;
var strSource;
var strType;
var intSelected;
var strMsg;
var bolReformatted;
for (intcounter=0; intcounter <= intLimit; intcounter++) {
if(aobjCols[intcounter] == null) continue;

strValue = document.forms[0].v_object[intcounter].value;
strType = document.forms[0].v_object[intcounter].type;

//test selected items for select lists
if(strType == "select-one") {
intSelected = document.forms[0].v_object[intcounter].selectedIndex;
/*reassign list value because when cancelling submit for some
unexplained reasons it becomes "" */
strValue= document.forms[0].v_object[intcounter].options[intSelected].value;
document.forms[0].v_object[intcounter].value = strValue;
}

if(LF_AllSpaces(strValue) && aobjCols[intcounter].mandatory){
// test mandatory columns
alert(aobjCols[intcounter].name + " is mandatory!");
return false;
}

if(strType == 'textarea' && strValue.length > 300){
// test max. length form object
alert(aobjCols[intcounter].name+ " not valid, max. length is 300 chars!");
return false;
}

strSource=aobjCols[intcounter].source;

//Test if URL is correct
if(strSource.indexOf("url") != -1) {
    if(strValue.indexOf("http://") == -1 && strValue.indexOf("ftp://") == -1) {
        alert("URL must include protocol: 'http://' or 'ftp://'!");
        document.forms[0].v_object[intcounter].value = "http://" + strValue;
        return false;
    }
    if(strValue.indexOf(".") == -1) {
        strMsg = "URL must be a valid Resource Address! (i.e. http://www.myuni.edu.au/mydir/mypage.htm)"
        alert(strMsg);
        return false;
    }
}

// Test if E-mail is correct
if(strSource.indexOf("e_mail") != -1) {
    if(strValue.length > 0 && strValue != null && (strValue.indexOf("@") == -1 || strValue.indexOf("." ) == -1)) {
        alert("E-mail must include '@' and '.' or be empty or null!");
        return false;
    }
    if(strValue.length > 0 && strValue.indexOf("mailto:") == -1) {
        alert("Please do not include protocol in E-mail")
        document.forms[0].v_object[intcounter].value = strValue.substring(strValue.indexOf("mailto:" ) + 7, strValue.length);
        return false;
    }
}

/* default value for non mandatory col null */
/* textarea with nothing gives back an "e" */
if((LF_AllSpaces(strValue) || (strType == 'textarea' && strValue == "e")) && aobjCols[intcounter].mandatory == 0) {
    document.forms[0].v_object[intcounter].value = "null";
}

// reformat textarea do not include '\n' or '\r\n'
bolReformatted = false;
if(strType == 'textarea' &&
    document.forms[0].v_object[intcounter].value != "null") {
    document.forms[0].v_object[intcounter].value = LF_Reformat(strValue);
    bolReformatted = true;
}

// replace ' and " by '
if((strValue.indexOf("'" ) == -1 || strValue.indexOf("" " ) == -1) &&
    document.forms[0].v_object[intcounter].value != "null") {
    if(bolReformatted == true)
        strValue = document.forms[0].v_object[intcounter].value;
    document.forms[0].v_object[intcounter].value = LF_Replace_Chars(strValue);
}

return true;

function LF_Make_Spaces(intLength)
{
/* function generates spaces */

var str_Aux;
var intEnd;

str_Aux = "";
if(intLength < 20) {
    intEnd = 20 - (intLength + 1);
    for(inti=0;inti<=intEnd;inti++)
        str_Aux = str_Aux + ' ';
}
return str_Aux;
}

function LF_Generate()
{
    /* function generates UniGuide meta tags when a user clicks on the */
    /* generate button. Before it tests if values are correct */

    var inti;
    var int_Len;
    var str_Meta;
    var str_Column_Value;
    var str_Column_Source;
    var str_Column_Name;
    var str_Line;
    var bol_Mandatory;

    if(LF_tstDataIsOk()== true) {
        // specify header of meta tag
        str_Meta = "<!-- mandatory columns marked with * -->\n" +
        "<!-- Please Enter Values inside ' ' -->\n" +
        "<meta name=" + str_Table + '" scheme="UniGuide"\n' +
        'content="\n';

        for(inti=0;inti<=intLimit;inti++) {
            // generate individual column entries
            str_Column_Value  = document.frm_Meta.v_object[inti].value;
            str_Column_Name   = aobjCols[inti].name;
            str_Column_Source = aobjCols[inti].source;
            int_Len = str_Column_Source.length;
            bol_Mandatory     = aobjCols[inti].mandatory;

            str_Line = "(~ " + str_Column_Source + LF_Make_Spaces(int_Len) + "]=" +
            if(bol_Mandatory == true)
                // mandatory columns are marked with an asterisk
                str_Line = str_Line + '*';
            str_Line = str_Line + str_Column_Name + "= '";
            str_Line = str_Line + str_Column_Value + "' ~)";

            if(inti != intLimit)
                // separate column attribute/value with a comma
                str_Line = str_Line + ",\n";
        } else
            // terminate the meta tag
            str_Line = str_Line + ");\n";
str_Line = str_Line + '";'>
str_Meta = str_Meta + str_Line;
}

document.frm_Meta.v_meta.value = str_Meta;
}

// end of No Javascript -->
</script>
</HEAD>

<body>

<!-- Activate Error Handler -->
<?mierror err=ErrHandler1 tag=mivar>
<html>
<head><title>Error Page</title></head>
<body>
<h2>Error Page:</h2>
<hr>
<strong>Error Code($MI_ERRORCODE):</strong>$MI_ERRORMSG<br>
<hr>
<br>
<form>
<table width=100% align=center>
<tr align=center>
<td align=center>
<input type=button name=btn_goback value="Back" onClick="history.back();"/>
</td>
</tr>
</table>
</form>
</body>
</html>
</mierror>

<?mivar name=v_table_source default="_research_entity_spublications">$v_table_source<?/mivar>

<script language="javascript">
// set str_Table value needed for the name attribute of the meta tag
str_Table = "<?mivar>v_table_source<?/mivar>";
</script>

<!-- Capture table information -->
<?misql sql="SELECT table_name, table_help
FROM _tables t1
WHERE t1.table_source = '{v_table_source}';

$(SETVAR,v_table_help,"$2")

<?/misql>

<?FORM name="frm_Meta" method=post action="<?mivar>$WEB_HOME<?/mivar>">
<table border=0 align=right>
<tr>
<td><small>$1:</small><br>$3</td>
</tr>
</table>
</FORM>

<script language="Javascript">
//dynamically generate column object entry
aobjCols_init($4,"$2","$1","$5",$6);
</script>

<!-- Define meta tag text area -->

<br clear=all>
<table border=0 align=left>
<tr>
<td><br><hr></td>
</tr>
</table>
<br clear=all><br>
<table align=right>
<tr>
<td><small>Meta tag:</small><br>
<textarea name=v_meta rows=10 cols=72 wrap=physical>
</textarea>
</td>
</tr>
</table>
<br clear=all>
<!-- Buttons part -->

</script>
<tr>
<td><input type=reset value=" Undo "></td>
<td><input type=button value=" Generate " onclick="LF_Generate()"></td>
<td><input type=button value=" Help " onclick="LF_showHelp('<?mivar>$v_table_help<?/mivar>')"></td>
</tr>
</table>
</form>
<br><br><br><br>
<script language="Javascript">
  // capture number of maximum available columns
  intLimit = document.frm_Meta.v_object.length - 1;
</script>
Appendix C. UniGuide Indexing Agent

C.1 UniGuide Indexing Agent SQL3 Generator modules: Source Code
   C.1.1 DML Statement Generation Modules: Row instances
   C.1.2 DML Statement Generation Modules: Set instances

C.2 UniGuide Indexing Agent: Hard Drive Version
   C.2.1 JAVA™ Source Code: app_Index_Robot_HD.java

C.1 UniGuide Indexing Agent SQL3 Generator modules: Source Code
   C.1.1 DML Statement Generation Modules: Row instances
   Structure Chart GF_robot_simple_insert.c:
Source Code: GF_robot_simple_insert.c
/* GF_robot_simple_insert.c */
#include "stdio.h"
#include "string.h"
#include "mi.h"
/* prototypes */
m_i_integer get_pk_cols(MI_CONNECTION *conn, /* connection to default database */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *astrpkcols[15]); /* array of cols. components PK */
m_i_integer test_row_exists(MI_CONNECTION *conn, /* connection to db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *astrwhere); /* where SQL statement */
mi_char1 *get_where_clause(mi_char1 *strtbl, /* name of table */
    mi_char1 *astrflds[15], /* array of columns */
    mi_char1 *astrvalues[15], /* array of values */
    mi_char1 *astrpkcols[15]); /* array of comp. of pk */
mi_char1 *get_update_statement(mi_char1 *strtbl, /* name of table */
    mi_char1 *astrflds[15], /* array of cols */
    mi_char1 *astrvalues[15], /* array of values */
    mi_char1 *astrwhere, /* array of comp. pk */
    mi_char1 *str_last_modified); /* last modified value */
mi_char1 *get_insert_statement(mi_char1 *strtbl, /* name of table */
    mi_char1 *astrflds[15], /* array of columns */
    mi_char1 *astrvalues[15], /* array of values */
    mi_char1 *str_last_modified); /* last modified value */
mi_char1 *get_last_modified_value(MI_CONNECTION *conn, /* connection to db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *strwhere); /* array of comp. of pk */
mi_char1 *change_to_type(mi_char1 *strvalue); /* value to change */
mi_char1 *get_column_value(MI_CONNECTION *conn, /* connect. to def. DB */
    mi_char1 *strsql); /* SQL statement */
void get_words(mi_char1 *straux, /* string del. by "^" */
    mi_char1 *astraux[15]); /* array of strings */
/* end of prototypes */
mi_text *GF_robot_simple_insert(mi_text *strtblarg,mi_text *strfldsarg,mi_text *strvaluesarg,mi_text *str_last_modified_arg)
{
    MI_CONNECTION *conn; /* conn. to default database */
    mi_char1 *str_last_modified; /* last modified attr. */
    mi_char1 *str_stored_last_modified; /* last modified attr. */
    mi_char1 *strsql; /* SQL statement returned */
    mi_char1 *strwhere; /* WHERE clause */
    mi_char1 *strtbl; /* name of table */
    mi_char1 *strflds; /* list of columns */
    mi_char1 *strvalues; /* list of values */
mi_char1 *astrflds[15];    /* array of columns       */
mi_char1 *astrvalues[15];  /* array of values          */
mi_char1 *astrpkcols[15];  /* array of cols. comp. PK   */
mi_integer inti, int_row_exists, interror;

/* allocate memory dynamically */
str_last_modified    = mi_alloc(sizeof(mi_char1) * 50);
str_stored_last_modified = mi_alloc(sizeof(mi_char1) * 50);
strtbl    = mi_alloc(sizeof(mi_char1) * 50);
strflds   = mi_alloc(sizeof(mi_char1) * 300);
strvalues = mi_alloc(sizeof(mi_char1) * 1000);
strsql    = mi_alloc(sizeof(mi_char1) * 1500);
strwhere  = mi_alloc(sizeof(mi_char1) * 500);

/* allocate memory dynamically to array of chars */
for(inti=0;inti<15;inti++) {
    astrflds[inti]    = mi_alloc(sizeof(mi_char1) * 50);
    astrvalues[inti]  = mi_alloc(sizeof(mi_char1) * 300);
    astrpkcols[inti]  = mi_alloc(sizeof(mi_char1) * 50);
}

/* convert text types into strings */
mi_text_to_buffer(strtblarg,strtbl);
mi_text_to_buffer(strfldsarg,strflds);
mi_text_to_buffer(strvaluesarg,strvalues);
mi_text_to_buffer(str_last_modified_arg,str_last_modified);

/* opens current opened database */
conn=mi_open(NULL,NULL,NULL);

generate array of columns to insert values*/
get_words(strflds,astrflds);

/* generate array of values associated to each column */
get_words(strvalues,astrvalues);

/* generate array of cols that are components of the key */
interror = get_pk_cols(conn,strtbl,astrpkcols);
if(interror == 0) {
    strcpy(strsql,"Return 0; ");
    return mi_string_to_text(strsql);
} else {

    /* generate WHERE clause of SQL statement */
    strcpy(strwhere,get_where_clause(strtbl,astrflds,astrvalues,astrpkcols));

    /* generate SQL query to test if row exists in the table */
    int_row_exists = test_row_exists(conn,strtbl,strwhere);
    if(int_row_exists == 1) {
        /* row exists, test last_modified attribute */
        strcpy(strstored_last_modified,get_last_modified_value(conn,strtbl,strwhere));
        if(strcmp(str_last_modified,strstored_last_modified) <=0) {
            /* captured last_modified value is older or equal than the stored one */
            /* therefore we do not update any column value */
            strcpy(strsql,"Return 0; ");
            return mi_string_to_text(strsql);
        } else {
        
        }
    } else {

}
else
    /* captured last modified value is newer than the stored one */
    /* a DML Update statement must be generated */
    strsql = get_update_statement(strtbl,astrflds,astrvalues,strwhere,str_last_modified);
}
else
    /* Row does not exist, it will insert new row with currently captured timestamp value */
    strsql = get_insert_statement(strtbl,astrflds,astrvalues,str_last_modified);

/* close the connection to the default database */
mi_close(conn);

/* deallocate memory */
mi_free(strtbl);
mi_free(astrflds);
mi_free(strvalues);
mi_free(strwhere);
mi_free(str_last_modified);
mi_free(str_stored_last_modified);

/* deallocate memory to array of chars */
for(inti=0;inti<15;inti++) {
    mi_free(astrflds[inti]);
    mi_free(astrvalues[inti]);
    mi_free(astrpkcols[inti]);
}

/* convert into text type again */
return mi_string_to_text(strsql);
}

mi_int get_pk_cols(MI_CONNECTION *conn, /* connection to default database */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *astrpkcols[15]) /* array of cols. components PK */
{
    /* Returns an array of columns that are component of the key */

    MI_ROW *row;

    mi_char1 *strcolvalue, *strsql;
    mi_integer intlen,interr,intresult,inti;

    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 500);

    /* generate SQL command */
    /* get the columns that are components of the primary key */
    strncpy(strsql,"SELECT c.column_source FROM _columns c";)
    strcat(strsql," WHERE c.table_source='");
    strcat(strsql,strtbl);
    strcat(strsql,"' AND c.column_pk = 1;";)

    /*execute query */
    mi_exec(conn,strsql,0);
inti = 0;
while((intresult = mi_get_result(conn)) != MI_NO_MORE_RESULTS) {
    switch(intresult) {
    case MI_ERROR: 
        return 0;
    case MI_ROWS: 
        /* for each row, get only one column */
        while ((row=mi_next_row(conn,&interr)) != NULL) {
            /* get the value of the single column */
            switch (mi_value(row,0,&strcolvalue,&intlen)) {
                case MI_ERROR: 
                    return 0;
                case MI_NULL_VALUE: 
                    return 0;
                case MI_NORMAL_VALUE: 
                    strcpy(astrpkcols[inti],strcolvalue);
                    inti++;
                    break;
                default: return 0;
            } 
        } 
        break;
    }
    /* deallocate memory */
    mi_free(strsql);
    return 1;
}

mi_integer test_row_exists(MI_CONNECTION *conn, mi_char1 *strtbl, mi_char1 *strwhere) {
    /* Tests if a given row instance exists */
    mi_integer inti, intj;
    mi_char1 *strnumrows, *strsql;
    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 500);
    strcpy(strsql,"SELECT count(*) FROM ");
    strcat(strsql,strtbl);
    strcat(strsql, " WHERE ");
    strcat(strsql,strwhere);
    strcat(strsql,";");
    /* execute query */
    strnumrows = get_column_value(conn,strsql);
    /* deallocate memory */
mi_free(strsql);
/* return number of rows retrieved, should be 0 or 1 */
if(!strcmp(strnumrows,"0"))
  return 0;
else
  return 1;
}

mi_char1 *get_last_modified_value(MI_CONNECTION *conn, /* connection to db */
                                mi_char1 *strtbl,    /* name of table */
                                mi_char1 *strwhere)  /* WHERE clause */
{
    /* Retrieves last_modified value of a given row instance */

    mi_char1    *strvalue, *strsql;
    /* allocate memory dynamically */
    strsql     = mi_alloc(sizeof(mi_char1) * 500);
    strcpy(strsql,"");  
    strcat(strsql,"SELECT last_modified FROM ");
    strcat(strsql,strtbl);
    strcat(strsql," WHERE ");
    strcat(strsql,strwhere);
    strcat(strsql,";" );

    /*execute query */
    strvalue = get_column_value(conn,strsql);
    /* deallocate memory */
    mi_free(strsql);
    return strvalue;
}

mi_char1 *get_where_clause(mi_char1 *strtbl,    /* name of table */
                            mi_char1 *astrflds[15],   /* array of columns */
                            mi_char1 *astrvalues[15], /* array of values */
                            mi_char1 *astrpkcols[15]) /* array of comp. of pk */
{
    /* Generates WHERE clause */
    /* NOTE: all components of the PK are to date of type text that is why we can */
    /* convert into uppercase, if it is a number then we must cast to text ::text */

    mi_integer    inti, intj;
    mi_char1    *strvaluetype, *strwhere, *straux;
    /* allocate memory dynamically */
    strwhere = mi_alloc(sizeof(mi_char1) * 500);
    strvaluetype = mi_alloc(sizeof(mi_char1) * 500);
    strcpy(strwhere,"");
for(inti=0;inti<15;inti++) {
    if (!astrpkcols[inti][0])
        break;
    if(inti > 0)
        strcat(strwhere, " AND ");
    straux=strstr(astrpkcols[inti], "url");
    if (straux!=NULL)
        strcat(strwhere, astrpkcols[inti]);
    else {
        strcat(strwhere, "upper (");
        strcat(strwhere, astrpkcols[inti]);
        strcat(strwhere, ") ");
    }
    strcat(strwhere, ");
    for(intj=0;intj<15;intj++) {
        if(!astrflds[intj][0])
            break;
        if(!strcmp(astrflds[intj], astrpkcols[inti])) {
            strvaluetype = change_to_type(astrvalues[intj]);
            if (straux!=NULL)
                strcat(strwhere, strvaluetype);
            else {
                strcat(strwhere, "upper (");
                strcat(strwhere, strvaluetype);
                strcat(strwhere, ") ");
            }
            break;
        }
    }
}

/* deallocate memory */
mi_free(strvaluetype);

/* return WHERE clause of SQL statement */
return strwhere;

mi_char1 *get_update_statement(mi_char1 *strtbl,
        mi_char1 *astrflds[15],
        mi_char1 *astrvalues[15],
        mi_char1 *strwhere,
        mi_char1 *str_last_modified)
        /* last modified value */
        {
        /* Generates an SQL DML Update statement for a row instance */

        mi_char1      *strset, *strvaluetype, *strsql;
        mi_integer    inti;

        /* allocate memory dynamically */
        strsql  = mi_alloc(sizeof(mi_char1) * 1500);
        strset  = mi_alloc(sizeof(mi_char1) * 700);

        /* generate SET clause SQL statement */
        strcpy(strset, "SET ");
for(inti=0;inti<15;inti++) {
    if(!astrvalues[inti][0]) break;
    if(inti==1) {
        strcat(strset,",");
        strcat(strset, astrflds[inti]);
        strcat(strset, " = ");
        if(!strcmp(astrvalues[inti],"null")) {
            strcat(strset, "null");
        } else {
            strvaluetype=change_to_type(astrvalues[inti]);
            strcat(strset, strvaluetype);
        }
    }
}

/* generate WHERE clause of SQL statement */
strcpy(strsql,"UPDATE ");
strcat(strsql,strtbl);
strcat(strsql,strset);
strcat(strsql,"last_modified='");
strcat(strsql,str_last_modified);
strcat(strsql," WHERE ");
strcat(strsql,strwhere);
strcat(strsql,"' WHERE ");
strcat(strsql,strwhere);
strcat(strsql,"; ");

/* deallocate memory */
mi_free(strset);
/* return UPDATE SQL statement */
return strsql;
}

mi_char1 *get_insert_statement(mi_char1 *strtbl, mi_char1 *astrflds[15], mi_char1 *astrvalues[15], mi_char1 *str_last_modified) {
    mi_char1 *strvaluetype, *strsql, *strvalues, *strflds;
    mi_integer inti;

    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 1500);
    strvalues = mi_alloc(sizeof(mi_char1) * 1000);
    strflds = mi_alloc(sizeof(mi_char1) * 400);

    strcpy(strsql,""),
    strcpy(strvalues,""),
    strcpy(strflds,""),

    for(inti=0;inti<15;inti++) {
        if(!astrvalues[inti][0]) break;
        if(inti==1) {
            strcat(strvalues,"",");
            strcat(strflds,"",");
        }
    }
}
strcat(strflds, astrflds[inti]);
if(!strcmp(astrvalues[inti], "null"))
    strcat(strvalues, "null");
else {
    strvaluetype = change_to_type(astrvalues[inti]);
    strcat(strvalues, strvaluetype);
}
}

strcpy(strsql, "INSERT INTO ");
strcat(strsql, strtbl);
strcat(strsql, "(");
strcat(strsql, strflds);
strcat(strsql, ",last_modified) VALUES (");
strcat(strsql, strvalues);
strcat(strsql, ",");  
strcat(strsql, str_last_modified);
strcat(strsql, ");");

/* deallocate memory */
mi_free(strvalues);
mi_free(strflds);

/* return INSERT SQL statement */
return strsql;

mi_char1 * change_to_type(mi_char1 *strvalue)  /* value to change */
{
    /* Reformats value: adds to all values: ' ' */
    MI_ROW *row;

    mi_char1 *strexpr;

    /* allocate memory dynamically */
    strexpr = mi_alloc(sizeof(mi_char1) * 350);
    if(!strcmp(strvalue, "null"))
        strcpy(strexpr, "null");
    else {
        strcpy(strexpr, ");
    strcat(strexpr, strvalue);
    strcat(strexpr, ");
    }
    return strexpr;

    mi_char1 * get_column_value(MI_CONNECTION *conn,  /* connect. to def. database */
    mi_char1 *strsql) /* SQL statement to execute */
    {
        /* Executes an SQL statement that returns a single column value */

        MI_ROW *row;

int intlen, interr, intresult;
mi_char1 *strvalue;

/* execute query */
mi_exec(conn, strsql, 0);

while((intresult = mi_get_result(conn)) != MI_NO_MORE_RESULTS) {
    switch(intresult) {
        case MI.Rows:
            /* retrieve the single required row */
            row = mi_next_row(conn, &interr);

            /* get the value of the single column */
            mi_value(row, 0, &strvalue, &intlen);
    }
}
return strvalue;
}

void get_words(mi_char1 *straux, /* string delimited by "^" */
               mi_char1 *astraux[15]) /* array of strings */
{
    /* Generates an array of strings from a ^ delimited list */

    mi_integer inti;
    mi_char1 *pstr, *pstrword, *strword;

    /* the max. length that can have a values is 300 */
    strword = mi_alloc(300 * sizeof(mi_char1));
    pstr = straux;
    inti = 0;
    while (*pstr) {
        pstrword = strword;
        while (*pstr != '^' && *pstr) {
            *pstrword = *pstr;
            pstrword++;
            pstr++;
        }
        if (*pstr) pstr++;
        *pstrword = '\0';
        /* allocate dynamically memory for string depending on len of */
        /* length of the string */
        astraux[inti] = mi_alloc((strlen(strword) + 1) * sizeof(mi_char1));
        strcpy(astraux[inti], strword);
        inti++;
    }
    /* deallocate memory */
    mi_free(strword);
}
C.1.2 DML Statement Generation Modules: Set instances
Structure Chart GF_robot_complex_insert.c:

NOTE: Calls to lower reusable modules are not included in order to facilitate the reading of the Structure Chart

Figure C2. Structure Chart GF_robot_complex_insert.c
Structure Chart get_complex_insert_statement:

Structure Chart get_complex_update_statement:

Figure C3. Structure Chart get_complex_insert_statement

Figure C4. Structure Chart get_complex_update_statement
Source Code: GF_robot_complex_insert.c

/* GF_robot_complex_insert.c */

#include "stdio.h"
#include "string.h"
#include "mi.h"

/* prototypes */

mi_char1 *get_last_modified_value(MI_CONNECTION *conn, /* connection to db */
           mi_char1 *strtbl, /* name of table */
           mi_char1 *strparenttbl, /* parent table */
           mi_char1 *strwhere, /* exterior where clause */
           mi_char1 *strsetwhere, /* interior where clause */
           mi_char1 *str_last_modified_column); /* last modified column */

mi_char1 *get_last_modified_column(MI_CONNECTION *conn, /* conn to def. db */
           mi_char1 *strtbl); /* name of table */

mi_char1 *get_set_name(MI_CONNECTION *conn, /* conn to def. db */
           mi_char1 *strtbl); /* name of table */

mi_integer insert_parent_tuple(MI_CONNECTION *conn, /* conn to def. db. */
           mi_char1 *strtbl, /* name of table */
           mi_char1 *strparenttbl, /* name of parent table */
           mi_char1 *astrflds[15], /* array of columns */
           mi_char1 *astrvalues[15]); /* array of values */

mi_char1 *get_complex_update_statement(MI_CONNECTION *conn, /* conn. to def db */
           mi_char1 *strtbl, /* name of table */
           mi_char1 *strparenttbl, /* parent table */
           mi_char1 *astrflds[15], /* array of columns */
           mi_char1 *astrvalues[15], /* array of values */
           mi_char1 *strwhere, /* ext. where */
           mi_char1 *strsetwhere, /* int. where set */
           mi_char1 *str_last_modified_column, /* last_modified column */
           mi_char1 *str_last_modified_value); /* last_modified value */

mi_char1 *get_complex_insert_statement(MI_CONNECTION *conn, /* conn. to def db */
           mi_char1 *strtbl, /* name of table */
           mi_char1 *strparenttbl, /* parent table */
           mi_char1 *astrflds[15], /* array of columns */
           mi_char1 *astrvalues[15], /* array of values */
           mi_char1 *strwhere, /* ext. WHERE */
           mi_char1 *str_last_modified_column, /* last_modified column */
           mi_char1 *str_last_modified_value); /* last_modified value */

mi_char1 get_column_type(MI_CONNECTION *conn, /* conn. to def db */
           mi_char1 *strtbl, /* name of table */
           mi_char1 *strcol); /* name of column */

mi_char1 *get_ext_where_clause(MI_CONNECTION *conn, /* conn to def. db */
           mi_char1 *strtbl, /* name of table */
           mi_char1 *astrflds[15], /* array of columns */
           mi_char1 *astrvalues[15]); /* array of values */

mi_char1 *get_parent_table(MI_CONNECTION *conn, /* conn. to def db */
           mi_char1 *strtbl); /* name of table */

mi_integer test_parent_row_exists(MI_CONNECTION *conn, /* connection to db */
           mi_char1 *strparenttbl, /* parent table */
           mi_char1 *strwhere); /* array of comp. pk */

mi_integer test_setrow_exists(MI_CONNECTION *conn, /* connection to db */
           mi_char1 *strtbl, /* name of table */
           mi_char1 *strparenttbl, /* parent table */
           mi_char1 *strwhere, /* array of columns */
           mi_char1 *strsetwhere); /* WHERE inside set */

mi_integer get_pk_cols(MI_CONNECTION *conn, /* conn. to def. db */
           mi_char1 *strtbl, /* name of table */
           mi_char1 *astrpkcols[15]); /* array cols.comp.PK */

mi_char1 *get_where_clause(mi_char1 *strtbl, /* name of table */
           mi_char1 *astrflds[15], /* array of columns */
           mi_char1 *astrvalues[15], /* array of values */
           mi_char1 *strsetwhere); /* WHERE inside set */
mi_char1 *astrpkcols[15]);    /* array of comp. pk */

mi_char1 *change_to_type(mi_char1 *strvalue);    /* value to change */

mi_char1 *get_column_value(MI_CONNECTION *conn,    /* connect. to def. db */
    mi_char1 *strsql);    /* SQL stat. to execute */

void get_words(mi_char1 *straux,    /* string delim. by "^" */
    mi_char1 *astraux[15]);    /* array of strings */

/* end of prototypes */

mi_text *GF_robot_complex_insert(mi_text *strtblarg,mi_text *strfldsarg,mi_text *strvaluesarg,mi_text *str_last_modified_arg)
{
    MI_CONNECTION *conn;    /* connection to open database */
    mi_char1 *str_last_modified_column;    /* %last_modified column */
    mi_char1 *str_last_modified_value;    /* %last_modified value */
    mi_char1 *str_stored_last_modified_value;    /* %last_modified value */
    mi_char1 *strsql;    /* complex SQL statement returned */
    mi_char1 *strwhere;    /* exterior where clause SQL stat.*/
    mi_char1 *str_setwhere;    /* interior where clause(set) SQL statement */
    mi_char1 *strparenttbl;    /* name of parent table that holds the set */
    mi_char1 *strtbl;    /* name of table */
    mi_char1 *strflds;    /* list of columns */
    mi_char1 *strvalues;    /* list of values */
    mi_char1 *astrflds[15];    /* array of columns */
    mi_char1 *astrvalues[15];    /* array of values */
    mi_char1 *astrpkcols[15];    /* array of cols. that are components PK */

    mi_text_to_buffer(str_last_modified_arg,str_last_modified_value);
    mi_text_to_buffer(strtblarg,strtbl);
    mi_text_to_buffer(strfldsarg,strflds);
    mi_text_to_buffer(strvaluesarg,strvalues);

    /* opens current opened database */
    conn=mi_open(NULL,NULL,NULL);

    /* generate array of columns to insert values*/
    get_words(strflds,astrflds);
    /* generate array of values associated to each column */
    get_words(strvalues,astrvalues);

    /* generate array of cols that are components of the pseudo keys */
    interror = get_pk_cols(conn,strtbl,astrpkcols);
    if(interror == 0) {
        /* deallocate memory */
```
mi_free(strtbl);
mi_free(strflds);
mi_free(strvalues);
mi_free(strparenttbl);
mi_free(strsetwhere);
mi_free(strwhere);

/* deallocate memory to array of chars */
for(int i=0; i<15; i++) {
    if(strlen(astrpkcols[i])>0) mi_free(astrpkcols[i]);
    if(strlen(astrvalues[i])>0) mi_free(astrvalues[i]);
    if(strlen(astrflds[i])>0) mi_free(astrflds[i]);
}
strcpy(strsql, "Return 0;");
/* convert into text type again */
return mi_string_to_text(strsql);
}

/* generate interior WHERE clause of the set on pseudo PKs set */
strcpy(strsetwhere, get_where_clause(strtbl, astrflds, astrvalues, astrpkcols));

/* generate exterior WHERE clause of the parent table that holds the set */
strcpy(strwhere, get_ext_where_clause(conn, strtbl, astrflds, astrvalues));

/* get name of parent table */
strcpy(strparenttbl, get_parent_table(conn, strtbl));

/* test parent row exists to insert tuple in parent table if it does not exist */
int_row_exists = test_parent_row_exists(conn, strparenttbl, strwhere);
if(int_row_exists == 0) {
    introwsaffected = insert_parent_tuple(conn, strtbl, strparenttbl, astrflds, astrvalues);
    if(introwsaffected == 0) {
        /* deallocate memory */
        mi_free(strtbl);
        mi_free(strflds);
        mi_free(strvalues);
        mi_free(strparenttbl);
        mi_free(strsetwhere);
        mi_free(strwhere);
        /* deallocate memory to array of chars */
        for(int i=0; i<15; i++) {
            if(strlen(astrpkcols[i])>0) mi_free(astrpkcols[i]);
            if(strlen(astrvalues[i])>0) mi_free(astrvalues[i]);
            if(strlen(astrflds[i])>0) mi_free(astrflds[i]);
        }
        strcpy(strsql, "Return 0;");
        /* convert into text type again */
        return mi_string_to_text(strsql);
    }
}

/* Parent row does not exist, therefore child set row does not exist as well */
strsql = get_complex_insert_statement(conn, strtbl, strparenttbl, astrflds, astrvalues, strwhere, str_last_modified_value);
}
else {
    /* generate SQL query to test if set row exists in the table */
    int_set_instance_exists = test_setrow_exists(conn, strtbl, strparenttbl, strwhere, strsetwhere);
    if(int_set_instance_exists == 1) {
        /* get last_modified column and store value so we save re-executing the query */
        strcpy(str_last_modified_column, get_last_modified_column(conn, strtbl));
        /* get now %last_modified value */
        strcpy(str_stored_last_modified_value,
        get_last_modified_value(conn, strtbl, strparenttbl, strwhere, strsetwhere, str_last_modified_column));
        if(strcmp(str_last_modified_value, str_stored_last_modified_value) <=0) {
            /* captured last_modified value is older or equal than the stored one */
            /* therefore we do not update any column value */
            strcpy(strsql, "Return 0;");
            return mi_string_to_text(strsql);
        }
    }
    ...
else  /* captured last modified value is newer than the stored one */  
/* a nested DML Update statement must be generated */  
strsql =  
get_complex_update_statement(conn,strtbl,strparenttbl,astrflds,astrvalues,  
strwhere,strsetwhere,str_last_modified_column,str_last_modified_value);  
}  
else  /* Set instance does not exist, it will insert new row with captured timestamp */  
strsql=  
get_complex_insert_statement(conn,strtbl,strparenttbl,astrflds,astrvalues,  
strwhere,str_last_modified_value);  
}  
/* close the connection to the default database */  
mi_close(conn);  
/* deallocate memory */  
mi_free(str_last_modified_value);  
mi_free(str_last_modified_value);  
mifree(str_last_modified_column);  
mifree(strtbl);  
mifree(strflds);  
mifree(strvalues);  
mifree(strparenttbl);  
mifree(strsetwhere);  
mifree(strwhere);  
/* deallocate memory to array of chars */  
for(inti=0;inti<15;inti++)  
if(strlen(astrpkcols[inti])>0) mi_free(astrpkcols[inti]);  
if(strlen(astrvalues[inti])>0) mi_free(astrvalues[inti]);  
if(strlen(astrflds[inti])>0) mi_free(astrflds[inti]);  
}  
/* convert into text type again */  
return mi_string_to_text(strsql);  
}  

mi_integer insert_parent_tuple(MI_CONNECTION *conn,  
/* conn to def. db. */  
mi_char1 *strtbl,  
/* name of table */  
mi_char1 *strparenttbl,  
/* name of parent table */  
mi_char1 *astrflds[15],  
/* array of columns */  
mi_char1 *astrvalues[15])  
/* array of values */  
{  
/* Inserts tuple in parent table that holds the set */  
mi_char1 strcoltype, *strvalue_type, *strvalues, *strflds, *strsql;  
mi_integer inti, intsearchcols, intresult, introwsaffected;  

/* allocate memory dynamically */  
strsql = mi_alloc(sizeof(mi_char1) * 700);  
strflds = mi_alloc(sizeof(mi_char1) * 200);  
strvalues = mi_alloc(sizeof(mi_char1) * 300);  
strcpy(strvalues,"");  
strcpy(strflds,"");  
strcpy(strsql,"");  

intsearchcols=0;  
for(inti=0;inti<15;inti++)  
if(strlen(astrvalues [inti]) [0]) break;  
strcoltype=get_column_type(conn,strtbl,astrflds[inti]);  
switch(strcoltype)  
{  
case 'i':  
continue;  
case 's':  
intsearchcols++;  
if(intsearchcols>1) {  
strcpy(strflds,"");  
strcpy(strvalues,"");  
}  
strcpy(strflds,astrflds[inti]);  
strvalue_type=change_to_type(astrvalues[inti]);  

/* I am not sure what these are doing */  
/* some function to change the type of the value */  
strvalue_type=convert_type(strvalue_type);  
strcpy(strvalues,strvalue_type);  
strcpy(strsql,strvalues);  
strvalue_type="";}  
*/
strcat(strvalues,strvalue_type);
    break;
}

/* generate SQL command */
/* insert tuple parent table */
strcpy(strsql,"INSERT INTO ");
strcat(strsql,strparenttbl);
strcat(strsql,"(");
strcat(strsql,strflds);
strcat(strsql","url) VALUES(");
strcat(strsql,strvalues);
strcat(strsql,"'URL to be updated';");

/*execute query */
mi_exec(conn,strsql,0);
while((intresult = mi_get_result(conn))!=MI_NO_MORE_RESULTS) {
    switch(intresult)
    {
    case MI_ERROR:
        introwsaffected = 0;
        break;
    case MI_DML:
        introwsaffected = mi_result_row_count(conn);
        if (introwsaffected == MI_ERROR)
            introwsaffected = 0;
        break;
    default:
        introwsaffected = 0;
        break;
    }
}

/* deallocate memory */
mi_free(strflds);
mi_free(strvalues);
mi_free(strsql);

return introwsaffected;

mi_char1 *get_ext_where_clause(MI_CONNECTION *conn,  /* conn to def. db */
                                mi_char1 *strtbl,       /* name of table */
                                mi_char1 *astrflds[15], /* array of columns */
                                mi_char1 *astrvalues[15]) /* array of values */
{
    /* Generates exterior WHERE clause */

    mi_char1 strcoltype, *strvalue_type, *strwhere;
    mi_integer inti, intsearchcols;

    /* allocate memory dynamically */
    strwhere = mi_alloc(sizeof(mi_char1) * 500);
    strcpy(strwhere,"");
    intsearchcols=0;
    for(inti=0;inti<15;inti++) {
        if(!astrvalues[inti][0]) break;
        intcoltype=get_column_type(conn,strtbl,astrflds[inti]);
        switch(strcoltype) {
        case 'i':
            continue;
            case 's':
                intsearchcols++;
                if(intsearchcols>1)
                    strcat(strwhere," AND ");
                if(!strcmp(astrflds[inti],"url"))
                    strcat(strwhere,astrflds[inti]);
                else {
                    strcat(strwhere,"upper(");
                    strcat(strwhere,astrflds[inti]);
                    strcat(strwhere,"\"]");
                }
            break;
        case 'b':
            break;
        case 'd':
            break;
    }
}

/* allocate memory dynamically */
strcpy(strwhere,"");
intsearchcols=0;
for(inti=0;inti<15;inti++) {
    if(!astrvalues[inti][0]) break;
    intcoltype=get_column_type(conn,strtbl,astrflds[inti]);
    switch(strcoltype) {
    case 'i':
        continue;
        case 's':
            intsearchcols++;
            if(intsearchcols>1)
                strcat(strwhere," AND ");
            if(!strcmp(astrflds[inti],"url"))
                strcat(strwhere,astrflds[inti]);
            else {
                strcat(strwhere,"upper(");
                strcat(strwhere,astrflds[inti]);
                strcat(strwhere,"\"]");
            }
        break;
    case 'b':
        break;
    case 'd':
        break;
    }
}

} strcat(strwhere,"=");
strvalue_type=change_to_type(astrvalues[inti]);
if(!strcmp(astrflds[inti],"url"))
strcat(strwhere,strvalue_type);
else {
strcat(strwhere,"upper(");
strcat(strwhere,strvalue_type);
strcat(strwhere,")");
} break;
}

return strwhere;
}

mi_integer get_pk_cols(MI_CONNECTION *conn, /* conn. to def. db */
                    mi_char1 *strtbl, /* name of table */
                    mi_char1 *astrpkcols[15]) /* array of cols.comp.PK */
{
    /* Returns an array of columns that are component of the key */
    MI_ROW *row;

    mi_char1 *strcolvalue, *strsql;
    mi_integer intlen,interr,intresult,inti;

    /* allocate memory dynamically */
    strsql  = mi_alloc(sizeof(mi_char1) * 300);

    /* generate SQL command */
    /* get the columns that are components of the primary key */
    strcat(strsql,"SELECT c.column_source FROM _columns c");
    strcat(strsql," WHERE c.table_source='");
    strcat(strsql,strtbl);
    strcat(strsql,"' AND c.column_pk = 1;");
    /*execute query */
    mi_exec(conn,strsql,0);

    inti = 0;
    while((intresult = mi_get_result(conn))!=MI_NO_MORE_RESULTS) {
        switch(intresult)
        {
        case MI_ERROR:
            mi_free(strsql);
            return 0;
        case MI_ROWS:
            /* for each row, get only one column */
            while ((row=mi_next_row(conn,&interr)) != NULL) {
                /* get the value of the single column */
                switch (mi_value(row,0,&strcolvalue,&intlen))
                {
                case MI_ERROR:
                    mi_free(strsql);
                    return 0;
                case MI_NULL_VALUE:
                    mi_free(strsql);
                    return 0;
                case MI_NORMAL_VALUE:
                    astrpkcols[inti] = strcolvalue;
                    inti++;
                    break;
                default:
                    mi_free(strsql);
                    return 0;
                }
            } break;
        }
    } break;

    /* deallocate memory */
    mi_free(strsql);
mi_integer test_parent_row_exists(MI_CONNECTION *conn, /* connection to db */
   mi_char1 *strparenttbl, /* parent table */
   mi_char1 *strwhere) /* ext. WHERE clause */
{
    /* Tests if a parent row table that holds the set exists */
    mi_integer inti, intj;
    mi_char1 *strnumrows, *strsql;
    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 500);
    strcpy(strsql,"");
    strcat(strsql,"SELECT count(*) FROM ");
    strcat(strsql, strparenttbl);
    strcat(strsql, " WHERE ");
    strcat(strsql, strwhere);
    strcat(strsql,";");
    /* execute query */
    strnumrows = get_column_value(conn,strsql);
    /* deallocate memory */
    mi_free(strsql);
    /* return number of rows retrieved, should be 0 or 1 */
    if(!strcmp(strnumrows,"0")) return 0;
    else return 1;
}

mi_integer test_setrow_exists(MI_CONNECTION *conn, /* connection to db */
   mi_char1 *strtbl, /* name of table */
   mi_char1 *strparenttbl, /* parent table */
   mi_char1 *strwhere, /* ext. WHERE clause */
   mi_char1 *strsetwhere) /* int. WHERE clause */
{
    /* Tests set row exists */
    mi_integer inti, intj;
    mi_char1 *strnumrows, *strsql;
    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 500);
    strcpy(strsql,"");
    strcat(strsql,"SELECT count(*) FROM ");
    strcat(strsql, strtbl);
    strcat(strsql," WHERE poid = (SELECT oid FROM ");
    strcat(strsql, strparenttbl);
    strcat(strsql," WHERE ");
    strcat(strsql,strwhere);
    strcat(strsql,") AND ");
    strcat(strsql,strsetwhere);
    strcat(strsql,";");
    /* execute query */
    strnumrows = get_column_value(conn,strsql);
    /* deallocate memory */
    mi_free(strsql);
    /* return number of rows retrieved, should be 0 or 1 */
    if(!strcmp(strnumrows,"0")) return 0;
    else return 1;
}

mi_char1 *get_where_clause(mi_char1 *strtbl, /* name of table */
   mi_char1 *astrflds[15], /* array of columns */
   mi_char1 *astrvalues[15], /* array of values */
   mi_char1 *astrpkcols[15]) /* array of comp. of pk */
/* Generates interior where clause of SQL statement */

mi_integer inti, intj;
mi_char1 *strvaluetype, *strwhere, *straux;

/* allocate memory dynamically */
strwhere = mi_alloc(sizeof(mi_char1) * 300);
strvaluetype = mi_alloc(sizeof(mi_char1) * 350);
strcpy(strwhere, "");
for(inti=0;inti<15;inti++) {
  if (!astrpkcols[inti][0])
    break;
  if(inti > 0)
    strcat(strwhere, " AND ");
  straux=strstr(astrpkcols[inti],"url");
  if (straux!=NULL)
    strcat(strwhere, astrpkcols[inti]);
  else {
    strcat(strwhere, "upper(");
    strcat(strwhere, astrpkcols[inti]);
    strcat(strwhere, ")");
  }
  strcat(strwhere, "=");
}
for(intj=0;intj<15;intj++) {
  if(!astrflds[intj][0])
    break;
  if(!strcmp(astrflds[intj],astrpkcols[inti])) {
    strvaluetype = change_to_type(astrvalues[intj]);
    if (straux!=NULL)
      strcat(strwhere, strvaluetype);
    else {
      strcat(strwhere, "upper(");
      strcat(strwhere, strvaluetype);
      strcat(strwhere, ")");
    }
  } else {
  break;
  }
}

/* deallocate memory */
mi_free(strvaluetype);

/* return WHERE clause of SQL statement */
return strwhere;

mi_char1 *get_complex_update_statement(MI_CONNECTION *conn, /* conn. to def db */
    mi_char1 *strtbl, /* name of table */
    mi_char1 *strparenttbl, /* parent table */
    mi_char1 *astrflds[15], /* array of columns */
    mi_char1 *astrvalues[15], /* array of values */
    mi_char1 *strwhere, /* ext. where */
    mi_char1 *strsetwhere, /* int. where set */
    mi_char1 *str_last_modified_column, /* last_modified column */
    mi_char1 *str_last_modified_value) /* last_modified value */

{ /* Generates complex SQL set instance update */

    mi_integer inti, intinsertcols;

    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 1500);
    strsetupdate = mi_alloc(sizeof(mi_char1) * 700);
    strsetname = mi_alloc(sizeof(mi_char1) * 50);
    strcpy(strsetupdate, "");  
    strcpy(strsql, "");
    intinsertcols=0;

}
for(inti=0;inti<15;inti++) {
    if(!astrvalues[inti][0]) break;
    strcoltype=get_column_type(conn,strtbl,astrflds[inti]);
    switch(strcoltype) {
    case 'i':
        intinsertcols++;
        if(intinsertcols>1)
            strcat(strsetupdate,"","");
        strcat(strsetupdate,astrflds[inti]);
        strcat(strsetupdate," = ");
        strvaluetype=change_to_type(astrvalues[inti]);
        strcat(strsetupdate,strvaluetype);
        break;
    case 's':
        continue;
    }
}

/* get name of set */
strcpy(strsetname,get_set_name(conn,strtbl));

strcpy(strsql,"UPDATE ");
strcat(strsql,strparenttbl);
strcat(strsql," SET(UPDATE ");
strcat(strsql,strsetname);
strcat(strsql,".");
strcat(strsql,strsetupdate);
strcat(strsql,str_last_modified_column);
strcat(strsql,"=");
strcat(strsql,str_last_modified_value);
strcat(strsql," WHERE ");
strcat(strsql,strwhere);
strcat(strsql," WHERE ");
strcat(strsql,str_last_modified_column);
strcat(strsql,"=");
strcat(strsql,str_last_modified_value);
strcat(strsql," WHERE ");
strcat(strsql,strwhere);
strcat(strsql,"");  
/* deallocate memory */
mi_free(strsetupdate);
mi_free(strsetname);  
/* return UPDATE SQL statement */
return strsql;
}

mi_char1 *get_complex_insert_statement(MI_CONNECTION *conn, /* conn. to def db */
                                      mi_char1 *strtbl, /* name of table */
                                      mi_char1 *strparenttbl, /* parent table */
                                      mi_char1 *astrflds[15], /* array of columns */
                                      mi_char1 *astrvalues[15], /* array of values */
                                      mi_char1 *strwhere, /* ext. WHERE */
                                      mi_char1 *str_last_modified_value) /* last_modified value */
{
    /* Generates complex SQL statement insert set instance */

    mi_char1 *strvalue_type, *strsql, *strvalues, *strflds, *str_last_modified;
    mi_char1 *strtablename, strcoltype;
    mi_integer inti, intinsertcols;

    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 2000);
    strvalues = mi_alloc(sizeof(mi_char1) * 1000);
    strflds = mi_alloc(sizeof(mi_char1) * 400);
    strtablename = mi_alloc(sizeof(mi_char1) * 50);
    str_last_modified = mi_alloc(sizeof(mi_char1) * 50);

    strcpy(strsql,"");
    strcpy(strvalues,"");
    strcpy(strflds,"");  
    intinsertcols=0;
    for(inti=0;inti<15;inti++) {
        if(!astrvalues[inti][0]) break;
strcoltype = get_column_type(conn, strtbl, astrflds[inti]);
switch (strcoltype) {  
  case 'i':
    intinsertcols++;
    if (intinsertcols > 1) {
      strcat (strflds, ",");
      strcat (strvalues, ",");
    }
    strcat (strflds, astrflds[inti]);
    strvalue_type = change_to_type(astrvalues[inti]);
    strcat (strvalues, strvalue_type);
    break;
  case 's':
    continue;
  }
/* get name of set */
strcpy (strsetname, get_set_name(conn, strtbl));
/* get name of last_modified column of the set */
strcpy (str_last_modified, get_last_modified_column(conn, strtbl));
strsql = "UPDATE ";
strcat (strsql, strparenttbl);
strcat (strsql, " SET(INSERT INTO ");
strcat (strsql, strparenttbl);
strcat (strsql, ",");
strcat (strsql, strsetname);
strcat (strsql, "(");
strcat (strsql, astrflds);
strcat (strsql, ",");
strcat (strsql, str_last_modified);
strcat (strsql, ") VALUES(");
strcat (strsql, strvalues);
strcat (strsql, ");
strcat (strsql, str_last_modified_value);
strcat (strsql, ") WHERE ");
strcat (strsql, strwhere);
strcat (strsql, ";");
/* deallocate memory */
mi_free (strvalues);
mi_free (strflds);
mi_free (str_last_modified);
mi_free (strsetname);
/* return complex INSERT SQL statement */
return strsql;
}

mi_char1 get_column_type(MI_CONNECTION *conn, /* conn. to def db */
                         mi_char1 *strtbl, /* name of table */
                         mi_char1 *strcol) /* name of column */
{
  /* Retrieves the column type: 's' search or 'i' insert */
  mi_char1 *strvalue;
  mi_char1 *strsql;
  /* allocate dynamically memory */
  strsql = mi Alloc (sizeof(mi_char1) * 200);
  /* generate SQL command */
  strcpy (strsql, "SELECT c.column_type FROM _columns c");
  strcat (strsql, " WHERE c.table_source = ");
  strcat (strsql, strtbl);
  strcat (strsql, " AND c.column_source = ");
  strcat (strsql, strcol);
  strcat (strsql, ");
  /* execute query */
  strvalue = get_column_value (conn, strsql);
  /* deallocate memory */
  mi_free (strsql);
mi_char1 *get_parent_table(MI_CONNECTION *conn, mi_char1 *strtbl)
{
    mi_char1 *strvalue;
    mi_char1 *strsql;

    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 300);
    /* generate SQL command */
    strcpy(strsql,"SELECT t.parent_table FROM _tables t";
    strcat(strsql," WHERE t.table_source='";
    strcat(strsql,strtbl);
    strcat(strsql,"';";
    /* execute query */
    strvalue = get_column_value(conn,strsql);
    /* deallocate memory */
    mi_free(strsql);
    return strvalue;
}

mi_char1 *get_set_name(MI_CONNECTION *conn, mi_char1 *strtbl)
{
    mi_char1 *strvalue;
    mi_char1 *strsql;

    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 200);
    /* generate SQL command */
    strcpy(strsql,"SELECT t.set_name FROM _tables t";
    strcat(strsql," WHERE t.table_source='";
    strcat(strsql,strtbl);
    strcat(strsql,"';";
    /* execute query */
    strvalue = get_column_value(conn,strsql);
    /* free memory */
    mi_free(strsql);
    return strvalue;
}

mi_char1 *get_last_modified_column(MI_CONNECTION *conn, mi_char1 *strtbl)
{
    mi_char1 *strvalue;
    mi_char1 *strsql;

    /* allocate memory dynamically */
    strsql = mi_alloc(sizeof(mi_char1) * 200);
    /* generate SQL command */
    strcpy(strsql,"SELECT unique(c1.column_name) FROM columns c1, tables t1";
    strcat(strsql," WHERE c1.column_member = t1.table_type AND t1.table_name='";
    strcat(strsql,strtbl);
    strcat(strsql,"% last_modified';";
}
mi_char1 *get_last_modified_value(MI_CONNECTION *conn, mi_char1 *strtbl, mi_char1 *strparenttbl, mi_char1 *strwhere, mi_char1 *strsetwhere, mi_char1 *str_last_modified_column) {
/* Retrieves last_modified value from a set instance */

mi_char1 *strsql;
/* allocate memory dynamically */
strsql = mi_alloc(sizeof(mi_char1) * 500);
strcpy(strsql,"SELECT ");
strcat(strsql,str_last_modified_column);
strcat(strsql," FROM ");
strcat(strsql,strtbl);
strcat(strsql," WHERE poid = (SELECT oid FROM ");
strcat(strsql,strparenttbl);
strcat(strsql," WHERE ");
strcat(strsql,strwhere);
strcat(strsql," AND ");
strcat(strsql,strsetwhere);
strcat(strsql,"; ");

/*execute query */
strvalue = get_column_value(conn,strsql);
/* free memory */
mi_free(strsql);
return strvalue;
}

mi_char1 *change_to_type(mi_char1 *strvalue) {
/* Reformats value: adds to all values: '' */

MI_ROW *row;
mi_char1 *strexpr;
/* allocate memory dynamically */
stexpr = mi_alloc(sizeof(mi_char1) * 350);
if(!strcmp(strvalue,"null"))
    strcpy(strexpr,"null");
else {
    strcpy(strexpr,"'");
    strcat(strexpr,strvalue);
    strcat(strexpr,");
    }
    return strexpr;
}

mi_char1 *get_column_value(MI_CONNECTION *conn, mi_char1 *strsql) {
/* execute query */
strvalue = get_column_value(conn,strsql);

/* free memory */
mi_free(strsql);
return strvalue;
}
MI_ROW *row;
int intlen,interr,intresult;
michar1 *strvalue;

/*execute query */
mi_exec(conn,strsql,0);

while((intresult = mi_get_result(conn))!=MI_NO_MORE_RESULTS) {
    switch(intresult) {
    case MI_ROWS:
        /* retrieve the single required row */
        row=mi_next_row(conn,&interr);

        /*get the value of the single column */
        mi_value(row,0,&strvalue,&intlen);
    }
    return strvalue;
}

void get_words(michar1 *straux, /* string delimited by "^" */
    michar1 *astraux[15]) /* array of strings */
{
    /* Generates an array of strings from a ^ delimited list */

    mi_integer inti;
michar1 *pstr,*pstrword,*strword;

    /* the max. length that can have a values is 300 */
    strword = mi_alloc(300 * sizeof(michar1));
    pstr=straux;
    inti=0;
    while (*pstr) {
        pstrword=strword;
        while(*pstr!='^' && *pstr) {
            *pstrword=*pstr;
            pstrword++;
            pstr++;
        }
        if(*pstr) pstr++;
        *pstrword='\0';
        /* allocate dynamically memory for string depending on len of */
        /* length of the string */
        astraux[inti]= mi_alloc(strlen(strword)+1) * sizeof(michar1));
        strcpy(astraux[inti],strword);
        inti++;
    }
    /* deallocate memory */
    mi_free(strword);
}
C.2 UniGuide Indexing Agent: Hard Drive Version
C.2.1 JAVA™ Source Code: app_Index_Robot_HD.java

/* app_Index_Robot_HD */
/* import statements */
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.text.SimpleDateFormat;
import java.text.FieldPosition;
import java.sql.*;
import java.util.*;

/**
 * Applet implements a breadth-first indexing robot.
 * Extracts all the URLs referenced inside the document: href tags, image maps, and frames.
 * and also extracts metadata, UniGuide meta tags if present in the Web page.
 * In addition stores entity instances detected on Web pages in an ILLUSTRRA ORDB.
 *
 * @author Carlos F. Enguix
 * @version 1.9, 26/7/98
 * @since JDK1.1.
 * @see java.applet.Applet
 *
 * public class app_Index_Robot_HD extends Applet
 implements KeyListener, ActionListener, Runnable
 {
  /**
   * No active URL_Scanner_Obj Threads.
   */
  private int int_No_Active_Threads;

  /**
   * No milliseconds elapsed since indexing robot was activated.
   */
  private long lng_Elapsed_Time;

  /**
   * Timestamp value when indexing robot was activated.
   */
  private java.util.Date dat_Start;

  /**
   * Table of URL_Scanner_Obj Threads.
   */
  private Vector tTable;

  /**
   * Customised Objects */

  /**
   * Applet Main Thread, Generates URL_Scanner_Obj Threads.
   */
  private Thread App_Main_Thread;

  /**
   * Daemon stops URL_Scanner_Obj threads periodically
   */
  private Stop_URL_Scanner Stop_URL_Scanner_Daemon;

  /**
   * JDBC Interface to URL_catalogue and URL_to_visit tables
   */
  private JDBC_URL_Saver JDBC_URL_Saver_Obj;

  /**
   * GUI objects */
  // Label Objects
  private Label lbl_URL;
  private Label lbl_URLs_Visited;
  private Label lbl_Metadata0, lbl_Metadata1;
  private Label lbl_Columns1, lbl_Columns2;
private Label lbl_Robot_Statistics0, lbl_Robot_Statistics1;
private Label lbl_System_Statistics0, lbl_System_Statistics1;

/**
 * Accepts Starting URL to scan
 */
private TextField txtf_URL;

// TextArea Objects
/**
 * Presents URL visited information.
 */
private TextArea txta_URLs_Visited;

/**
 * Presents Meta tags information.
 */
private TextArea txta_Metadata;

/**
 * Presents Robot Statistics.
 */
private TextArea txta_Robot_Statistics;

/**
 * Presents System Statistics.
 */
private TextArea txta_System_Statistics;

/**
 * Panel Changes Color on certain operations.
 */
private Panel pan_Separator;

// Button Objects
/**
 * Button stops thread execution.
 */
private Button btn_Stop;

/**
 * Button stops application and exits.
 */
private Button btn_Exit;

// GridBag Layout
private GridBagLayout gb_Layout;    // GridBag layout
private GridBagConstraints gb_Constraints;    // Gridbag constraints

/**
 * Generates the applet GUI and initialises vars.
 * @see #Add_Component
 */
public void init()
{
    String str_Label1, str_Label2;
    gb_Layout = new GridBagLayout();
    setLayout(gb_Layout);
    gb_Constraints = new GridBagConstraints();

    // Starting URL part
    lbl_URL = new Label("Starting URL:");
    txtf_URL = new TextField(70);
    txtf_URL.addKeyListener(this);

    // List of URLs visited part
    lbl_URLs_Visited = new Label("List of URLs Visited:");
    str_Label2 = "No URL";
    str_Label1 = "Last Modified";
    lbl_Columns1 = new Label(str_Label1);
    }
lbl_Columns2 = new Label(str_Label2);
txta_URLs_Visited = new TextArea(10,125);

// List of Meta-tags part
lbl_Metadata0 = new Label("List of Meta tags processed:");
str_Label1="NoProcessed Instance Type Status";
lbl_Metadata1 = new Label(str_Label1);
txta_Metadata = new TextArea(10,125);

// Robot Statistics part
lbl_Robot_Statistics0  = new Label("Robot Statistics: ");
str_Label1="NoVisited NoEntries Catalogue NoEntries Waiting List NoErrors NoMeta-tags processed NoTuples Inserted/Updated";
lbl_Robot_Statistics1  = new Label(str_Label1);
txta_Robot_Statistics  = new TextArea(2,125);

// System Statistics part
lbl_System_Statistics0 = new Label("System Statistics: ");
str_Label1="NoActive Threads NoStopped Threads(Time Exceeded) Date & Time Initiated Time Elapsed (Milliseconds)";
lbl_System_Statistics1 = new Label(str_Label1);
txta_System_Statistics = new TextArea(2,125);

//panel is needed to separate textarea from buttons part
pan_Separator = new Panel();

// Buttons part
btn_Stop = new Button("Stop");
btn_Stop.addActionListener(this);
btn_Exit = new Button("Exit");
btn_Exit.addActionListener(this);

// Grid bag constraints
gb_Constraints.anchor=GridBagConstraints.CENTER;
gb_Constraints.fill = GridBagConstraints.HORIZONTAL;

Add_Component(lbl_URL,gb_Layout,gb_Constraints,0,0,1,1);
Add_Component(txtf_URL,gb_Layout,gb_Constraints,0,1,1,1);
Add_Component(lbl_URLs_Visited,gb_Layout,gb_Constraints,1,0,1,1);
Add_Component(lbl_Columns1,gb_Layout,gb_Constraints,2,0,1,1);
Add_Component(lbl_Columns2,gb_Layout,gb_Constraints,2,1,1,1);
Add_Component(txta_URLs_Visited,gb_Layout,gb_Constraints,3,0,2,1);
Add_Component(lbl_Metadata0,gb_Layout,gb_Constraints,4,0,1,1);
Add_Component(lbl_Metadata1,gb_Layout,gb_Constraints,5,0,2,1);
Add_Component(txta_Metadata,gb_Layout,gb_Constraints,6,0,2,1);
Add_Component(lbl_Robot_Statistics0,gb_Layout,gb_Constraints,7,0,2,1);
Add_Component(lbl_Robot_Statistics1,gb_Layout,gb_Constraints,8,0,2,1);
Add_Component(txta_Robot_Statistics,gb_Layout,gb_Constraints,9,0,2,1);
Add_Component(lbl_System_Statistics0,gb_Layout,gb_Constraints,10,0,2,1);
Add_Component(lbl_System_Statistics1,gb_Layout,gb_Constraints,11,0,2,1);
Add_Component(txta_System_Statistics,gb_Layout,gb_Constraints,12,0,2,1);
Add_Component(pan_Separator,gb_Layout,gb_Constraints,13,0,2,1);
Add_Component(btn_Stop,gb_Layout,gb_Constraints,14,0,1,1);
Add_Component(btn_Exit,gb_Layout,gb_Constraints,14,1,1,1);

App_Main_Thread = null;

/**
 * keyReleased event associated to text field txta_URLs_Visited.<p>
 * When the user presses enter in the textfield the applet main thread is started.
* @param key_Event The KeyEvent object generated
* /
public void keyReleased (KeyEvent event) {
    if(event.getKeyCode() == KeyEvent.VK_ENTER && App_Main_Thread == null) {
        // Reset text areas
        txta_URLs_Visited.setText("");
        txta_Metadata.setText("");
        txta_Robot_Statistics.setText("");
        txta_Robot_Statistics.setText("");
        txta_System_Statistics.setText("");

        // initialise memory tables
tTable = new Vector(15);

        // Begin applet main thread
        App_Main_Thread = new Thread(this,"App_Main_Thread");
        App_Main_Thread.setPriority(Thread.NORM_PRIORITY + 2);
        App_Main_Thread.start();
    }
}

// declare this in order to avoid error messages from compiler
public void keyPressed(KeyEvent e) { }
public void keyTyped(KeyEvent e) { }

/**
* actionPerformed event associated to buttons.
* <ul>
* <li>If stop button is pressed, then threads are stopped <p>
* <li>If exit button is pressed, terminates program execution
* </ul>
* @param event The ActionEvent object generated
*/
public void actionPerformed(ActionEvent event) {
    Object obj_source;
    obj_source = event.getSource();
    if(obj_source == btn_Exit) {
        // Terminate application
        System.exit(0);
    } else if(obj_source == btn_Stop) {
        // Stop URL_Scanner_Obj Threads
        if(Stop_URL_Scanner_Daemon != null) {
            Stop_URL_Scanner_Daemon.stop();
            Stop_URL_Scanner_Daemon = null;
        }
        if(App_Main_Thread != null) {
            App_Main_Thread.stop();
            App_Main_Thread = null;
        }

        JDBC_URL_Saver_Obj.Close_Connection();
        JDBC_URL_Saver_Obj = null;
        System.gc();
    }
}

/**
* Main thread is executed spanning URLScanner Threads.<p>
* An initial URL is scanned and from there for every link a URL_Scanner_Obj thread
* is created, always first asking for permission to span a new thread. If the number
* of URL_Scanner thread is less than the maximum allowed then a new URL_Scanner thread
* can be started, otherwise it will wait until notified.
* @see #Get_Domain
* @see java.net.URL
* @see URL_Scanner
* @see ThreadLimit
* @see JDBC_URL_Saver
* @see Robot_Stats
*/
public void run()
{  
  final int K_int_Max_Threads = 2; // Constant: no. Maximum URL_Scanner_Obj Threads
  int int_URLs_catalogue; // No URLs fetched
  int int_URLs_to_Scan; // No URLs in URLs_to_visit table
  int int_URLs_visited; // No URLs visited
  int int_Reaction_Time; // Reaction Time to threads
  int int_vTable_Size; // Current no. of URLs to visit
  int int_Counter; // Counter for periodical Robot Statistics
  int int_Counter2; // Counter to reset URL visited textarea
  
  String str_Host; // Host being accessed
  String str_Domain; // Domain name of starting URL
  // i.e. www.uow.edu.au -> uow.edu.au
  
  String str_URL; // URL string
  java.util.Date dat_Now; // Current date
  long lng_Starting_Time; // Starting time indexing robot was activated
  
  URL URL_app; // URL to scan
  URL_Scanner First_URL_Scanner; // 1st URL_Scanner_Obj Thread
  URL_Scanner Next_URL_Scanner; // URL_Scanner_Obj Thread
  ThreadLimit Gate; // Gate prevents more than K_int_Max_Threads
  // in memory executing concurrently
  
  Robot_Stats Robot_Stats_Obj; // Robot Statistics

  try {
    // Capture Starting URL
    URL_app = new URL(txtf_URL.getText());
    str_Host = URL_app.getHost();
    // Get Domain String: i.e www.uow.edu.au -> uow.edu.au
    // so the scanning process includes all servers of a given University
    str_Domain = Get_Domain(str_Host);
    // Initialise Robot Statistics
    Robot_Stats_Obj = new Robot_Stats(0,0,0,0,0,0,txta_Robot_Statistics);
    // Initialise JDBC Interface to URLs tables
    JDBC_URL_Saver_Obj = new JDBC_URL_Saver(Robot_Stats_Obj);
    // Drop and create tables URLs_catalogue and URLs_to_visit
    JDBC_URL_Saver_Obj.init();
    str_URL = URL_app.toString();
    JDBC_URL_Saver_Obj.Save_URL(str_URL);
    str_URL = JDBC_URL_Saver_Obj.Get_URL();
    int_URLs_to_Scan = 1;
    // Capture starting time
    dat_Start = dat_Now = new java.util.Date();
    lng_Starting_Time = dat_Now.getTime();
    lng_Elapsed_Time = int_No_Active_Threads = 0;
    // Set maximum number of concurrent URL_Scanner_Obj threads = 2
    // JDBC DRIVER supports at the most 3 simultaneous connections
    Gate = new ThreadLimit(K_int_Max_Threads);
    int_URLs_visited = 1;
    int_Counter = int_Counter2 = 0;
    int_Reaction_Time = 1000000;
    // Scan 1st URL
    First_URL_Scanner = new URL_Scanner(URL_app, Gate, str_Domain, txta_URLs_Visited,
                                          txta_Metadata, int_URLs_visited, Robot_Stats_Obj, tTable, pan_Separator);
    First_URL_Scanner.start();
    try {
      First_URL_Scanner.Get_MetaData_Statistics(0,"None Detected Yet",
                                              "Scanning For UniGuide Meta tags...");
  
  } catch (Exception e) {
    e.printStackTrace();
  }
}
First_URL_Scanner.join();
} catch (InterruptedException I_excep) {}
First_URL_Scanner = null;
System.gc();

// Initialise Daemon that stops hanged/delayed URL_Scanner_Obj threads
// Daemon runs periodically with maximum priority
// Priorities are pre-emptive so we force to execute
Stop_URL_ScannerDaemon = new Stop_URL_Scanner(Gate, tTable, txta_System_Statistics, pan_Separator);
Stop_URL_ScannerDaemon.setPriority(Thread.MAX_PRIORITY);
Stop_URL_ScannerDaemon.setDaemon(true);
Stop_URL_ScannerDaemon.start();

// Capture number of URLs to visit
int_vTable_Size = JDBC_URL_Saver_Obj.Get_URLs_to_visit_Count();
do {
  try {
    if(int_vTable_Size >0) { // If URLs to visit
      // Gate is open is not max. number of Threads is reached
      // Otherwise main thread stops until receive notification
      // of permission
      Gate.Get_Permit();
      // Get next URL to visit
      str_URL = JDBC_URL_Saver_Obj.Get_URL();
      if(str_URL != null) {
        URL_app = new URL(str_URL);
        int_URLs_visited++;
        // Generate a new URL_Scanner_Obj thread
        Next_URL_Scanner = new URL_Scanner(URL_app, Gate, str_Domain, txta_URLs_Visited,
        txta_Metadata, int_URLs_visited, Robot_Stats_Obj, tTable, pan_Separator);
        Next_URL_Scanner.start();
      }
    }
  }
  catch (MalformedURLException MalURL_Excep) {
    // URL is incorrect add 1 to errors
    Robot_Stats_Obj.Set_Value('E',1);
    Robot_Stats_Obj.Get_Statistics();
  }
  finally {
    // Capture Elapsed time
    dat_Now = new java.util.Date();
    lng_Elapsed_Time = dat_Now.getTime();
    lng_Elapsed_Time = lng_Elapsed_Time - lng_Starting_Time;
    // Capture no. of current active threads
    int_No_Active_Threads = ThreadLimit.Get_Thread_Number();
    // Display system statistics
    Get_System_Statistics();
    int_Counter++;
    if(int_Counter == 5) { // Every 5 cycles Robot stats.
      // Output statistics to textarea
      int_URLs_catalogue = JDBC_URL_Saver_Obj.Get_URLs_Catalogue_Count();
      int_vTable_Size = JDBC_URL_Saver_Obj.Get_URLs_to_visit_Count();
      Robot_Stats_Obj.Set_All_Values(int_URLs_visited, int_URLs_catalogue,
      int_vTable_Size, -1, -1, -1);
      // Capture current number of URLs in Waiting List
      int_URLs_to_Scan = int_vTable_Size;
      int_Counter = 0;
    }
  }
} // end-try
catch (MalformedURLException MalURL_Excep) {
  // URL is incorrect add 1 to errors
  Robot_Stats_Obj.Set_Value('E',1);
  Robot_Stats_Obj.Get_Statistics();
}
332255

}
int_Counter2++;
if(int_Counter2 == 50) {
// Refresh textArea so it does not get out of memory
txta_URLs_Visited.setText("");
int_Counter2 = 0;
}
if(int_No_Active_Threads == 0 && int_URLs_to_Scan == 0)
int_Reaction_Time--;
else
int_Reaction_Time = 60000; // 1 minute of reaction time
// Display Robot Statistics
Robot_Stats_Obj.Get_Statistics();
}
} while(int_Reaction_Time != 0);
}
catch( MalformedURLException url_excep) {
showStatus("Exception: " + url_excep.toString());
}
}
/**
* Displays System statistics, mainly number of concurrent threads and elapsed time.
*
* @see Stop_URL_Scanner#Get_No_Threads_Stopped
*/
public void Get_System_Statistics()
{
int int_No_Stopped_Threads =
Stop_URL_Scanner_Daemon.Get_No_Threads_Stopped();
txta_System_Statistics.setText("\t" + int_No_Active_Threads +
"\t\t\t" + int_No_Stopped_Threads + "\t\t\t" + dat_Start +
"\t\t" + lng_Elapsed_Time);
}
/**
* Generates a string representing the common domain name of a URL. <p>
* For example the initial URL was <code>"www.uts.edu.au"</code> and the result is
* <code>"uts.edu.au"</code>. Therefore all servers of the university will have a
* common domain name.
*
* @param str_Host
Initial Host string to be converted into a Domain name
* @return
String representing the common domain name to all servers
*
of a given University
*/
public String Get_Domain(String str_Host)
{
int int_Index;
int int_pos, int_dots;
String str_aux;
int_Index = str_Host.indexOf(".");
if(int_Index == -1)
//we are probably handling the file protocol
str_aux = "file";
else {
int_pos = str_Host.length() - 1;
int_dots = 0;
while(int_dots != 3 && int_pos != -1) {
if(str_Host.charAt(int_pos) == '.')
int_dots++;
int_pos--;
}
str_aux = str_Host.substring(int_pos + 2);
}
return str_aux;


/
/**
* Adds AWT components to a GridBag FlowLayout. Components are display on the
* intersection of row/column values.
*
* @param comp_loc    The component to be created
* @param gbl_loc     The GridBagLayout object
* @param gbc_loc     The GridBag Constraints object
* @param int_row     The row to be inserted
* @param int_col     The column to be inserted
* @param int_width   The width of the component
* @param int_height  The height of the component
*/
public void Add_Component(Component comp_loc,
                         GridBagLayout gbl_loc,
                         GridBagConstraints gbc_loc,
                         int int_row,
                         int int_col,
                         int int_width,
                         int int_height)
{
    gbc_loc.gridx = int_col;
    gbc_loc.gridy = int_row;
    gbc_loc.gridwidth = int_width;
    gbc_loc.gridheight = int_height;
    gbl_loc.setConstraints(comp_loc,gbc_loc);
    add(comp_loc);
}
} // end-class

/**
* Class Stop_URL_Scanner implements an independent thread that stops hanged/delayed
* URL_Scanner_Obj threads. Used mainly to define a resident daemon.
*
*/
class Stop_URL_Scanner extends Thread{
/**
* No of threads stopped
*/
private static int int_No_Stopped_Threads;
/**
* Gate to set permit to other URL_Scanners.
*/
private ThreadLimit Gate;
/**
* Table of URL_Scanner threads
*/
private Vector tTable;
/**
* System statistics textarea.
*/
private TextArea txta_System_Statistics;
/**
* Panel provides user feedback changing color when a URL_Scanner thread is stopped.
*/
private Panel pan_Separator;
/**
* Initialises object mainly with the table that contains the URL_Scanner threads to be
* checked.
*
*/
public void init()
{
    @param Gate    Needed to set permit for other URL_Scanner threads
    @param tTable  Vector of URL_Scanner threads
    @param txta_System_Statistics  System Statistics textarea
}
* @param pan_Separator Panel Object set to red when a Thread has been stopped
*/
public Stop_URL_Scanner(ThreadLimit Gate, Vector tTable, TextArea txta_System_Statistics, Panel pan_Separator)
{
    int_No_Stopped_Threads = 0;
    this.Gate = Gate;
    this.tTable = tTable;
    this.txta_System_Statistics = txta_System_Statistics;
    this.pan_Separator = pan_Separator;
}

/**
* Stop_URL_Scanner thread is activated every 30 seconds with maximum priority in order to ensure that
* it will have processor time more or less exactly when awaken (pre-emptive priorities scheduling).
* When a URL_Scanner thread is stopped the daemon will give permit to other in order to proceed.
*/
public void run()
{
    // Constants
    final int K_int_Max_Time;  // Max. milliseconds a URL_Scanner_Obj thread can be in memory
    final int K_int_Max_Seconds;  // Max. seconds a URL_Scanner_Obj thread can be in memory
    final int K_int_Max_Sleep;  // Max. milliseconds the thread sleeps
    boolean bol_aux;  // Auxiliar var
    java.util.Date dat_Now;  // Current date
    long lng_now;  // No milliseconds now
    long lng_Time_Activated;  // No milliseconds when URL_Scanner_Obj activated
    Enumeration enum_Active_Threads;  // Enumeration of URL_Scanner_Obj Threads
    URL_Scanner URL_Scanner_Obj;  // Current URL_Scanner Object
    K_int_Max_Seconds = 600;
    K_int_Max_Time = K_int_Max_Seconds * 1000;
    K_int_Max_Sleep = 30000;
    while (true) {
        // Get current time
        dat_Now = new java.util.Date();
        lng_now = dat_Now.getTime();
        // Get current URL_Scanner threads in memory
        enum_Active_Threads = tTable.elements();
        txta_System_Statistics.append("\n Stop_URL_Scanner Daemon Searching" +
        " for Hanged URL_Scanner_Obj Threads...");
        while (enum_Active_Threads.hasMoreElements()) {
            txta_System_Statistics.append(".");
            URL_Scanner_Obj = (URL_Scanner) enum_Active_Threads.nextElement();
            if (URL_Scanner_Obj != null) {
                // Get time activated URL_Scanner
                lng_Time_Activated = URL_Scanner_Obj.Get_Time_Activated();
                if (lng_now - lng_Time_Activated >= K_int_Max_Time) {
                    // If in memory more than the permitted time remove it
                    try {
                        bol_aux = tTable.removeElement(URL_Scanner_Obj);
                    }
                    catch (NoSuchElementException NSE_excep) {} finally {
                        pan_Separator.setBackground(Color.red);
                        txta_System_Statistics.append("\n Permitted Thread Time Exceeded: " +
                        " URL_Scanner_Obj Activated At ->" + lng_Time_Activated + " has been stopped\n")
                        // Stop the URL_Scanner thread
                        URL_Scanner_Obj.stop();
                        URL_Scanner_Obj = null;
                        // Give permit to other URL_Scanner threads
                        Gate.Set_Permit();
                    } // End: finally
                } // End: if
            } // End: if
        } // End: while
    } // End: while
} // End: public void run()
int_No_Stopped_Threads++;
    pan_Separator.setBackground(Color.lightGray);
}
} // end-of-if
} // end-of-while
System.gc();
// Give processor time to other threads
Thread.yield();
try {
    // Sleep for a while, reactivate when sleeping time has finished
    Thread.sleep(K_int_Max_Sleep);
} catch( InterruptedException IO_excep ) {} //end
} //end-of-while
} //end-of-run
/**
* Returns No of URL_Scanner threads stopped since the indexing robot has been started.
* @return int Number of URL_Scanner threads stopped
*/
public static int Get_No_Threads_Stopped()
{
    return int_No_Stopped_Threads;
}
} //end-of-class
/**
* class ThreadLimit is used to control the number of maximum URL_Scanner threads
* executing concurrently in memory.
* class ThreadLimit {
/**
* No of maximum URL_Scanner threads permitted
*/
    final int K_int_Max_Threads;
/**
* Current no. of URL_Scanner threads
*/
    static int intThreads;
/**
* Initialises object with a maximum number of permitted concurrent threads.
* @param K_int_Max_Threads Number of maximum URL_Scanner threads
*/
    public ThreadLimit(int K_int_Max_Threads)
    {
        this.K_int_Max_Threads = K_int_Max_Threads;
        this.intThreads = 0;
    }
/**
* Returns number of Current URL_Scanner threads executing concurrently.
* @
*/
    public static int Get_Thread_Number()
    {
        return intThreads;
    }
/**
* Implements a semaphore, when there are less than max. permitted URL_Scanner threads
* a URL_Scanner may continue executing, otherwise it will have to wait.
* @
*/
    public synchronized void Get_Permit()
    {
        while(true) {
            if((intThreads + 1) <= K_int_Max_Threads) {
                intThreads++;
                break;
            } try {
                wait();
            } catch( InterruptedException IO_excep ) {} //end
        }
    }
catch (InterruptedException I_excep) {} }
}

/**
 * Implements a semaphore, when a URL Scanner thread terminates it will notify
 * all others that it has finished, so others can continue executing.
 */
public synchronized void Set_Permit() {
    intThreads--;  
    if(intThreads<0)
        intThreads = 0;  
    notifyAll();
} }  
//end-Class

/**
 * class Robot_Stats keeps tracks of Robot Statistics.
 */
class Robot_Stats {
/**
 * No URLs visited.
 */
private int int_No_Visited;
/**
 * No URLs catalogue.
 */
private int int_No_URLs_Catalogue;
/**
 * No URLs to visit.
 */
private int int_No_URLs_to_Visit;
/**
 * No Errors generated.
 */
private int int_No_Errors;
/**
 * No UniGuide Meta tags processed.
 */
private int int_No_Meta;
/**
 * No Tuples Inserted/Updated.
 */
private int int_No_Tuples;
/**
 * Textarea Robot Statistics.
 */
private TextArea txta_Robot_Statistics;
/**
 * Initialises a Robot_Statistics instance object normally with 0's.
 */
public Robot_Stats( int int_No_Visited, int int_No_URLs_Catalogue,
        int int_No_URLs_to_Visit, int int_No_Errors, int int_No_Meta,
        int int_No_Tuples, TextArea txta_Robot_Statistics) {
    this.int_No_Visited = int_No_Visited;
    this.int_No_URLs_Catalogue= int_No_URLs_Catalogue;
    this.int_No_URLs_to_Visit = int_No_URLs_to_Visit;
    this.int_No_Errors = int_No_Errors; 
    this.int_No_Meta = int_No_Meta;
/**
 * Sets individuals Robot statistics values.
 * 
 * @param chr_Attribute Code of the attribute to be modified
 * @param int_Value Value to be set
 *
 * public synchronized void Set_Value(char chr_Attribute, int int_Value)
 {
     switch(chr_Attribute) {
         case 'V':
             int int_No_Visited = int_Value;
             break;
         case 'H':
             int_No_URLs_Catalogue = int_Value;
             break;
         case 'W':
             int_No_URLs_to_Visit = int_Value;;
             break;
         case 'E':
             int_No_Errors = int_No_Errors + int_Value;
             break;
         case 'M':
             int_No_Meta = int_No_Meta + int_Value;
             break;
         case 'T':
             int_No_Tuples = int_No_Tuples + int_Value;
             break;
         default: //ignore
             break;
     }
 }
 */

/**
 * Gets individual Robot Statistics values.
 * 
 * @param chr_Attribute Code of the attribute to be retrieved
 * @return int representing individual Robot Statistics value captured
 *
 * public synchronized int Get_Value(char chrAttribute)
 {
     int int_Value;
     switch(chrAttribute) {
         case 'V':
             int_Value = int_No_Visited;
             break;
         case 'H':
             int_Value = int_No_URLs_Catalogue;
             break;
         case 'W':
             int_Value = int_No_URLs_to_Visit;
             break;
         case 'E':
             int_Value = int_No_Errors;
             break;
         case 'M':
             int_Value = int_No_Meta;
             break;
         case 'T':
             int_Value = int_No_Tuples;
             break;
         default: //return error code
             int_Value = -1;
             break;
     }
     return int_Value;
 }
 */

/**
 * Sets all Robot Statistics values.
 */
public synchronized void Set_All_Values(int int_No_Visited_val, int int_No_URLs_Catalogue_val, int int_No_URLs_to_Visit_val, int int_No_Errors_val, int int_No_Meta_val, int int_No_Tuples_val) {
    // Substitution part
    if(int_No_Visited_val != -1)
        int_No_Visited = int_No_Visited_val;
    if(int_No_URLs_Catalogue_val != -1)
        int_No_URLs_Catalogue = int_No_URLs_Catalogue_val;
    if(int_No_URLs_to_Visit_val != -1)
        int_No_URLs_to_Visit = int_No_URLs_to_Visit_val;

    // Incremental part
    if(int_No_Errors_val != -1)
        int_No_Errors = int_No_Errors + int_No_Errors_val;
    if(int_No_Meta_val != -1)
        int_No_Meta = int_No_Meta + int_No_Meta_val;
    if(int_No_Tuples_val != -1)
        int_No_Tuples = int_No_Tuples + int_No_Tuples_val;
}

public synchronized void Get_Statistics() {
    String str_Stats;
    str_Stats = String.valueOf(int_No_Visited) + "\t\t" +
                String.valueOf(int_No_URLs_Catalogue) + "\t\t" +
                String.valueOf(int_No_URLs_to_Visit) + "\t\t" +
                String.valueOf(int_No_Errors) + "\t\t" +
                String.valueOf(int_No_Meta) + "\t\t" +
                String.valueOf(int_No_Tuples);

    txta_Robot_Statistics.setText(str_Stats);
}

** Implements the JDBC Interface to the tables URL_catalogue and URL_to_visit.**

private Robot_Stats Robot_Stats_Obj;
private Connection conn_ILLUSTRA;

* Initialises object instance with a Robot Statistics object.
* @param Robot_Stats_Obj  Robot Statistics Object
*/
public JDBC_URL_Saver(Robot_Stats_Obj)
{
    this.Robot_Stats_Obj = Robot_Stats_Obj;
}

/**
 * Opens a connection to an ILLUSTRA Database.
 */
public void Open_Connection()
{
    String str_JDBC_URL;  // JDBC URL
    conn_ILLUSTRA = null;
    try {
        // for security reasons we do not include URL, user & password
        str_JDBC_URL = "jdbc:<subprotocol>://<host.domain>:<port>/<databasename>";
        Class.forName( "mijdbc.MIDriver" );
        // Open connection
        conn_ILLUSTRA = DriverManager.getConnection(str_JDBC_URL, "_User", "_Password");
    } catch (Exception JDBC_Excep) { System.out.println(JDBC_Excep.toString()); }
}

/**
 * Drops and creates the URL_catalogue and URL_to_visit tables.
 */
public void init()
{
    int int_Found;
    int int_Rows_Affected;
    String str_SQL;
    String str_Found;
    Statement stat_DDL;
    conn_ILLUSTRA = null;
    try {
        Open_Connection();

        // Phase I DML statement
        stat_DDL = conn_ILLUSTRA.createStatement();

        str_SQL = "SELECT count(table_name) FROM tables WHERE " +
                    "table_name='URL_catalogue';";

        // Get no. of tables whose name is URL_catalogue
        str_Found = Get_Column_Value(stat_DDL, str_SQL);
        int_Found = Integer.valueOf(str_Found).intValue();
        if(int_Found > 0) {
            // If URL_catalogue table found then drop table
            str_SQL = "DROP TABLE URL_catalogue;";
            int_Rows_Affected = stat_DDL.executeUpdate(str_SQL);
        }

        str_SQL = "SELECT count(table_name) FROM tables WHERE " +
                    "table_name='URL_to_visit';";

        // Get no. of tables whose name is URL_to_visit
        str_Found = Get_Column_Value(stat_DDL, str_SQL);
        int_Found = Integer.valueOf(str_Found).intValue();
        if(int_Found > 0) {
            // If URL_to_visit table found then drop table
            str_SQL = "DROP TABLE URL_to_visit;";
            int_Rows_Affected = stat_DDL.executeUpdate(str_SQL);
        }

        // Create URL_catalogue table
        str_SQL = "CREATE TABLE URL_catalogue (";

        // Create URL_to_visit table
        str_SQL = "CREATE TABLE URL_to_visit (";
    }
}
"CREATE TABLE URL_catalogue (url text not null primary key);"
int_Rows_Affected = stat_DDL.executeUpdate(str_SQL);

// Create primary key index
str_SQL = "CREATE INDEX URL_catalogue_pk_idx ON URL_catalogue USING btree(url);"
int_Rows_Affected = stat_DDL.executeUpdate(str_SQL);

// Create URL_to_visit table
str_SQL = "CREATE TABLE URL_to_visit (url text not null primary key);"
int_Rows_Affected = stat_DDL.executeUpdate(str_SQL);

// Create primary key index
str_SQL = "CREATE INDEX URL_to_visit_pk_idx ON URL_to_visit USING btree(url);"
int_Rows_Affected = stat_DDL.executeUpdate(str_SQL);

// Create secondary index on oid, needed to select minimum oid
str_SQL = "CREATE INDEX URL_to_visit_oid_idx ON URL_to_visit USING btree(oid);"
int_Rows_Affected = stat_DDL.executeUpdate(str_SQL);
}
catch (Exception JDBC_Excep) {
    System.out.println(JDBC_Excep.
    throw (SQLException)
*/
public String Get_Column_Value(Statement stat_loc, String str_SQL_Statement)
throws SQLException
{
    String strValue;
    ResultSet rs_loc;
    strValue="";
    // Execute SELECT statement
    rs_loc = stat_loc.executeQuery(str_SQL_Statement);
    // Retrieve Value: NULL or a valid string
    if(rs_loc.next())
        strValue = rs_loc.getString(1);
    return strValue;
}

/**
* Saves a given URL in both the URL_catalogue & URL_to_visit tables.
* @param str_url URL to be saved in the URL_catalogue & URL_to_visit tables
* @return int representing error code generated from the INSERT statement:
* <ul>
* <li>0 indicates no errors
* <li>1 an error has been detected
* </ul>
*/
public int Save_URL(String str_url)
{
    int inti;
    int intLimit;
    int int_Rows_Affected;
    }
int int_err;
String str_SQL;
String str_JDBC_URL;
Statement stat_DML;

int_err = 1;

try {
    
    Open_Connection();
    
    stat_DML = conn_ILLUSTRA.createStatement();
    
    str_SQL = "INSERT INTO URL_catalogue VALUES('" + str_url + ");";
    
    int_Rows_Affected = stat_DML.executeUpdate(str_SQL);
    
    if(int_Rows_Affected == 1) {
        
        str_SQL = "INSERT INTO URL_to_visit VALUES('" + str_url + ");";
        
        int_Rows_Affected = stat_DML.executeUpdate(str_SQL);
        
        if(int_Rows_Affected == 1) {
            int_err = 0;
        
    }
    }
    
    catch (Exception JDBC_Excep) {
        Robot_Stats_Obj.Set_Value('E',1);
    }
    
    finally {
    
        Close_Connection();
        return int_err;
    }
    
}/**
 * Retrieves next URL to scan (oldest one) from the URL_to_visit table and deletes
 * the row from the table at the same time.
 * @return String representing the URL to visit retrieved
 */
public String Get_URL()
{
    String str_WHERE;
    String str_url;
    String str_SQL;
    String str_JDBC_URL;
    Statement stat_SELECT;

    str_url = null;

    try {
        
        Open_Connection();
        
        stat_SELECT = conn_ILLUSTRA.createStatement();
        
        // Get the oldest URL_to_visit row, selecting the one that has the minimum OID
        // We retrieve the value at the same time it is deleted
        str_WHERE = "oid = (SELECT min(oid::text) FROM URL_to_visit)";
        
        str_SQL = "SELECT url, (DELETE FROM URL_to_visit WHERE " + str_WHERE + ") FROM URL_to_visit WHERE " + str_WHERE + ");";
        
        str_url = Get_Column_Value(stat_SELECT,str_SQL);
    }
    
    catch (Exception JDBC_Excep) {
        Robot_Stats_Obj.Set_Value('E',1);
        
        System.out.println("Stuffed Remove");
    }
}
finally {
    Close_Connection();
    return str_url;
}

/**
 * Retrieves number of URLs_to_visit rows or URLs to visit.
 * @return int representing the current number of URLs to visit in the
 * URL_to_visit table
 */
public int Get_URLs_to_visit_Count()
{
    int int_Count;
    String str_Count;
    String str_JDBC_URL;
    String str_SQL;
    String str_Found;
    Statement stat_SELECT;

    int_Count = 0;
    try {
        Open_Connection();
        stat_SELECT = conn_ILLUSTRA.createStatement();
        str_SQL = "SELECT count(url) FROM URL_to_visit;";
        // Get number of URLs to visit left
        str_Count = Get_Column_Value(stat_SELECT,str_SQL);
        int_Count = Integer.valueOf(str_Count).intValue();
    }
    catch (Exception JDBC_Excep) {
        Robot_Stats_Obj.Set_Value("E",1);
        Robot_Stats_Obj.Get_Statistics();
    }
    finally {
        Close_Connection();
        return int_Count;
    }
}

/**
 * Returns number of URLs_catalogue rows or URLs in catalogue.
 * @return int representing number of current rows in the URL_catalogue table
 */
public int Get_URLs_Catalogue_Count()
{
    int int_Count;
    String str_Count;
    String str_JDBC_URL;
    String str_SQL;
    String str_Found;
    Statement stat_SELECT;

    int_Count = 0;
    try {
        Open_Connection();
        stat_SELECT = conn_ILLUSTRA.createStatement();
        str_SQL = "SELECT count(url) FROM URL_catalogue;";
        // Get number of URLs in catalogue
        str_Count = Get_Column_Value(stat_SELECT,str_SQL);
        int_Count = Integer.valueOf(str_Count).intValue();
    }
catch (Exception JDBC_Excep) {
    Robot_Stats_Obj.Set_Value('E',1);
    Robot_Stats_Obj.Get_Statistics();
}
finally {
    Close_Connection();
    return int_Count;
}

/**
* Returns a boolean value indicating if a given URL exists in the URL_catalogue table.
* @param str_url URL to be tested if it exists in the URL_catalogue
* @return <code>true</code> indicates URL exists in the URL_catalogue table
*/
public boolean Exists_URL(String str_url) {
    int int_Found;
    boolean bol_Found;
    String str_JDBC_URL;
    String str_SQL;
    String str_Found;
    Statement stat_SELECT;

    bol_Found = true;
    try {
        Open_Connection();
        stat_SELECT = conn_ILLUSTRA.createStatement();
        str_SQL = "SELECT count(url) FROM URL_catalogue WHERE url='" + str_url + "';";
        // Get number of URLs as str_URL in the URL_catalogue
        str_Found = Get_Column_Value(stat_SELECT,str_SQL);
        int_Found = Integer.valueOf(str_Found).intValue();
        if(int_Found == 0)
            bol_Found = false;
    } catch (Exception JDBC_Excep) {
        Robot_Stats_Obj.Set_Value('E',1);
        Robot_Stats_Obj.Get_Statistics();
    }
    finally {
        Close_Connection();
        return bol_Found;
    }
}

/**
* Closes a Connection to an ILLUSTRA Database.
*/
public void Close_Connection() {
    if(conn_ILLUSTRA != null)
        try {
            conn_ILLUSTRA.close();
        } catch (Exception JDBC_Excep2) {}
}

/**
* Implements the JDBC Interface to any table/set table of the UniGuide schema.
*/
class JDBC_MetaData_Saver {
* URL to be saved.
*/
private String str_URL_app;

/**
* Name of table to be saved.
*/
private String str_Table;

/**
* List of Column names to be saved.
*/
private String str_Column_Names_List;

/**
* List of Column Values to be saved.
*/
private String str_Column_Values_List;

/**
* Last Modified timestamp attribute (YYYY-MM-DD HH:MM:SS.9999).
*/
private String str_Last_Modified;

/**
* Robot Statistics Object.
*/
private Robot_Stats Robot_Stats_Obj;

/**
* ILLUSTRA DB Connection.
*/
private Connection conn_ILLUSTRA;

/**
* Initialises the JDBC Interface with column values needed to generate SQL statements that
* interact with any of the UniGuide schema tables/set tables.
* *
* @param str_URL_app URL to be saved
* @param str_Table Name of table to be saved
* @param str_Column_Names_List List of Column names be saved
* @param str_Column_Values_List List of Column Values be saved
* @param str_Last_Modified Last Modified timestamp attribute
* @param Robot_Stats Robot_Stats_Obj Robot Statistics Object
*/
public JDBC_MetaData_Saver(String str_URL_app, String str_Table, String str_Column_Names_List, String str_Column_Values_List, String str_Last_Modified, Robot_Stats Robot_Stats_Obj) {
    this.str_URL_app = str_URL_app;
    this.str_Table = str_Table;
    this.str_Column_Names_List = str_Column_Names_List;
    this.str_Column_Values_List = str_Column_Values_List;
    this.str_Last_Modified = str_Last_Modified;
    this.Robot_Stats_Obj = Robot_Stats_Obj;
}

/**
* Opens a Connection to an ILLUSTRA Database.
* *
* @param str_JDBC_URL JDBC URL
*/
public void Open_Connection() {
    try {
        // for security reasons we do not include URL, user & password
        str_JDBC_URL = "jdbc:<subprotocol>://<host.domain>:<port>/<databasename>";
        Class.forName("mijdbc.MIDriver");

        // Open connection
        conn_ILLUSTRA = DriverManager.getConnection(str_JDBC_URL,"carlos","");
    } catch (Exception JDBC_Excep) {
        System.out.println(JDBC_Excep.toString());
    }
}
/** Retrieves a column value from the execution of an SQL SELECT statement.
 * 
 * @param stat_loc Statement Object 
 * @param str_SQL_Statement SQL SELECT statement 
 * @return String represents a column value retrieved from SQL 
 * @exception SQLException 
 * If an error occurs when executing SELECT statement 
 */

public String Get_Column_Value(Statement stat_loc, String str_SQL_Statement) throws SQLException {
    String strValue;
    ResultSet rs_loc;
    strValue = "";
    // Execute SELECT statement
    rs_loc = stat_loc.executeQuery(str_SQL_Statement);
    // Value is NULL or a valid string
    if (rs_loc.next())
        strValue = rs_loc.getString(1);
    return strValue;
}

/** DML SQL Statements are generated in two phases:
 * <ul>
 * <li>Phase I: Determine the type of construct being processed (row/set instance).
 * After this the required Return statement that generates a DML Insert/Update SQL
 * statement is executed
 * </li>
 * <li>Phase II: the previously generated DML statement is executed in the case of an
 * Insert/Update statement.
 * Otherwise the DML statement is ignored.
 * </li>
 * Insert statements are generated when new entity instances are detected.
 * Update statements are generated when a given entity instance has been modified,
 * indicated by
 * the last_modified timestamp value which is newer than the one stored in the ILLUSTRADB
 * 
 * @return String representing status code of the two Phase DML statement:
 *         Detected, Inserted, Updated or Error 
 */

public String Set_SQL_Statement() {
    int intRowsAffected;
    String str_Code_loc; // Status Code returned: Detected or Inserted or Updated or Error
    String str_DML_PhaseI; // Phase I : Generate a DML SQL statement
    String str_DML_PhaseII; // Phase II: Execute DML SQL statement
    String str_set_name; // Set name is valid or null
    String str_url_Column; // %url column of a given table
    Statement stat_DML; // Statement Object

    // By default
    str_Code_loc = "Error";
    str_DML_PhaseII = "";

    try {
        Open_Connection();

        // Phase I DML statement
        stat_DML = conn_ILLUSTRA.createStatement();
        // Test if is a set or table
        str_DML_PhaseI = "SELECT set_name FROM _tables WHERE table_source='" + str_Table + "';";
    }

}
str_set_name = "error";

// Get set_name value to test if it is a set or row instance
str_set_name = Get_Column_Value(stat_DML,str_DML_PhaseI);

// Define the url column associated to the set or row instance
if(str_set_name == null) {
    str_set_name = "null";
    str_URL_Column = "url";
} else { // search the url column of the set table
    str_DML_PhaseI = "SELECT column_source FROM _columns WHERE table_source='" + str_Table + "' AND column_source LIKE '%_url';"
    str_URL_Column = Get_Column_Value(stat_DML,str_DML_PhaseI);
}

//If not error
if(!str_set_name.equalsIgnoreCase("error")) {
    // Call the required external module
    if(str_set_name.equals("null"))
        // row instance
        str_DML_PhaseI = "Return GF_robot_simple_insert('";
    else
        // set instance
        str_DML_PhaseI = "Return GF_robot_complex_insert('";

    // common part
    str_DML_PhaseI = str_DML_PhaseI + str_Table + "," + str_URL_Column + "^" + str_Column_Names_List + "^" + str_URL_app + "^" + str_Column_Values_List + "^" + str_Last_Modified + ");"

    // Get DML Statement generated in PhaseI
    str_DML_PhaseII = Get_Column_Value(stat_DML,str_DML_PhaseI);

    // If no errors then execute DML statement
    if(!str_DML_PhaseII.equals("Return 0;")) {
        //stat_DML_PhaseII = conn_ILLUSTRA.createStatement();
        intRowsAffected = stat_DML.executeUpdate(str_DML_PhaseII);
        if(intRowsAffected == 0)
            str_Code_loc = "Error: Rejected Transaction";
        else
            Robot_Stats_Obj.Set_Value('T',1);
            Robot_Stats_Obj.Get_Statistics();

        // Return status code
        if(str_DML_PhaseII.indexOf("INSERT") != -1)
            // Is an Insert
            str_Code_loc = "Inserted new " + String.valueOf(intRowsAffected) + " entity instance/s";
        else // Is an update
            str_Code_loc = "Updated " + String.valueOf(intRowsAffected) + " entity instance/s";
    }
}

// Closes a Connection to an ILLUSTR A Database.
/*

public void Close_Connection()
if(conn_ILLUSTRA != null)  
    try {  
    conn_ILLUSTRA.close();  
    }  
    catch (Exception JDBC_Excep2) {  }
} // end-of-class

/**  
 * Parses retrieved Web pages and detects, extracts and stores UniGuide  
 * meta tags if present and fetches all links referenced in the Web page contents.  
 */
class URL_Scanner extends Thread  
{
    /**  
     * No of characters in Web page.  
     */
    private int int_Limit;
    /**  
     * No URLs visited.  
     */
    private int int_URLs_visited;
    /**  
     * No Errors generated, normally MalformedURL exceptions.  
     */
    private int int_Errors;
    /**  
     * Milliseconds when URL_Scanner thread was activated.  
     */
    private long lng_Time_Activated;
    /**  
     * Boolean indicates if HREF BASE tag is present.  
     */
    private boolean bol_Base;
    /**  
     * String copy stores original Web page contents.  
     */
    private String str_Web_page_copy;
    /**  
     * String copy stores lowercase Web page contents.  
     */
    private String str_Web_page_lower;
    /**  
     * Domain string to restrict search.  
     */
    private String str_Domain;
    /**  
     * Last modified timestamp attribute.  
     */
    private String str_Last_Modified;
    /**  
     * URL to scan (detect, extract metadata and links).  
     */
    private URL URL_app;
    /**  
     * Base URL needed to create Absolute URLs from relatives ones.  
     */
    private URL URL_Base;
    /**  
     * Text area URLs visited information.  
     */
private TextArea txta_URLs_Visited;
/**
* Text area Metadata Info.
*/
private TextArea txta_Metadata;
/**
* Table of URL_scanner threads.
*/
private Vector tTable;
/**
* Gate to set permit for other URL_Scanner when done.
*/
private ThreadLimit Gate;
/**
* Robot Statistics Object.
*/
private Robot_Stats Robot_Stats_Obj;
/**
* Panel object.
*/
private Panel pan_Separator;
/**
* Initialises a URL_Scanner thread mainly with the URL to be scanned, a ThreadLimit
* object to set permit to other URL_Scanners when done and so on.
* @param URL_app URL to scan (detect, extract metadata and links)
* @param Gate Gate to set permit for other URL_Scanner when done
* @param str_Domain Domain string to restrict search
* @param txta_URLs_Visited Text area URLs visited info
* @param txta_Metadata Text area Metadata Info
* @param int_URLs_visited No URLs visited
* @param Robot_Stats_Obj Robot Statistics Object
* @param tTable Table of URL_scanner threads
* @param pan_Separator Panel provides end-user feedback changing colour
* when a new URL is visited or a meta tag is detected
*/
URL_Scanner(URL URL_app, ThreadLimit Gate, String str_Domain, TextArea txta_URLs_Visited, TextArea txta_Metadata, int int_URLs_visited, Robot_Stats Robot_Stats_Obj, Vector tTable, Panel pan_Separator) {
    java.util.Date dat_Now;
    dat_Now = new java.util.Date(); // Capture current date
    lng_Time_Activated = dat_Now.getTime(); // Capture time milliseconds when activated
    this.setName(String.valueOf(lng_Time_Activated)); // Set name time activated to thread
    this.URL_app = URL_app;
    this.Gate = Gate;
    this.str_Domain = str_Domain;
    this.txta_URLs_Visited = txta_URLs_Visited;
    this.txta_Metadata = txta_Metadata;
    this.int_URLs_visited = int_URLs_visited;
    this.Robot_Stats_Obj = Robot_Stats_Obj;
    this.tTable = tTable;
    this.pan_Separator = pan_Separator;
    int_Errors = 0; // No errors
}
/**
* Opens a remote connection to the URL to be scanned, retrieves and parses the Web page
* contents capturing UniGuide meta tags if present and links referenced in order to permit other
* URL_Scanners to navigate other
* URLs.
public void run()
{
    final int K_intMaxValue = Integer.MAX_VALUE;
    // Constant Max. value an integer can have

    boolean bol_aux; // Auxiliar boolean var

    String str(Line); // Current captured line
    StringBuffer strbuf_HTMLText; // Appends dynamically text as being read

    InputStream input; // Aux. InputStream required as input for BufferedReader
    BufferedReader dataInput; // BufferedReader can read line by line the data stream
    URLConnection URL_conn; // Needed to retrieve header information

    // Add current URL Scanner thread to table
    tTable.addElement(this);

    try {
        strbuf_HTMLText = new StringBuffer();
        strbuf_HTMLText.ensureCapacity(5000);
        strbuf_HTMLText.setLength(0);
        // Get URLConnection object needed to retrieve header information
        URL_conn = URL_app.openConnection();
        // Get last modified value
        str_Last_Modified = Get_Last_Modified_Attr(URL_conn);
        // One more URL visited and to be scanned
        pan_Separator.setBackground(Color.green);
        txta_URLs_Visited.append(String.valueOf(int_URLs_visited) + "\t" + URL_app.toString() + Get_Spaces(URL_app.toString().length()) + str_Last_Modified + "\n");
        pan_Separator.setBackground(Color.lightGray);
        // get InputStream from URL.
        input = URL_conn.getInputStream();
        // InputStreamReader receives as input an InputStream and
        // converts the InputStream into a Reader, required input for BufferedReader.
        dataInput = new BufferedReader(new InputStreamReader(input));
        // Get line from stream
        str_Line = dataInput.readLine();
        while (str_Line != null) {
            if ((strbuf_HTMLText.length() + str_Line.length()) < K_intMaxValue) {
                strbuf_HTMLText.append(str_Line + "\n");
                str_Line = dataInput.readLine();
            } else
                // discard rest so it does not crash with out of memory
                break;
        }
        // Close stream
        dataInput.close();
        // Copy Web page contents to string
        str_Web_page_copy = strbuf_HTMLText.toString();
        strbuf_HTMLText = null;
        // Make a lowercase copy for string comparation
        str_Web_page_lower = str_Web_page_copy.toLowerCase();
        // Get number of characters to scan
        int_Limit = str_Web_page_lower.length();
        // Detect UniGuide meta tags, if present process them
        Get_MetaData();
        // Fetch HREF links in Web page contents
        Get_URLs();
        // Update number of errors generated when fetching URLs
        Robot_Stats_Obj.Set_Value('E', int_Errors);
    }
}
// Output Robot statistics
Robot_Stats_Obj.Get_Statistics();
}
} catch (IOException IO_excep) {
txta_URLs_Visited.append("\nException: " + IO_excep.toString() + "\n\n";
Robot_Stats_Obj.Set_Value('E',1);
}
} catch (OutOfMemoryError Mem_Err) {
txta_URLs_Visited.append("\nError: " + Mem_Err.toString() + "\n\n";
Robot_Stats_Obj.Set_Value('E',1);
}
} finally {
// One less URL_Scanner thread let others proceed
Gate.Set_Permit();
try {
// Remove from table of URL_Scanner;
bol_aux = tTable.removeElement(this);
} catch (NoSuchElementException NSE_excep) {}
}
}

/**
 * Retrieves the time that was activated the URL_Scanner thread needed to determine
 * if it has been in memory for long time.
 * @return Time (milliseconds) when URL_Scanner Thread was activated
 * @see Stop_URL_Scanner
 */
public long Get_Time_Activated()
{
return lng_Time_Activated;
}

/**
 * Returns n spaces needed to display formatted strings.
 * @param int_Length Length of a string to be filled with spaces
 * @return A String with (91 - intLength) spaces
 */
public String Get_Spaces(int int_Length)
{
StringBuffer strbuf_Aux;
int inti,intLimit;
if(int_Length >=91) return "";
else {
intLimit = 91 - int_Length;
strbuf_Aux = new StringBuffer(intLimit);
for(inti=0;inti<intLimit;inti++)
strbuf_Aux.append(' ');
return strbuf_Aux.toString();
}
}

/**
 * Fills with 0's for numbers in the range of 0..9 on the left hand-side.
 * For example: 9 is converted to 09.
 * This format is required for ILLUSTRATA time
 * stamp data values.
 * @param int_Number Number to be converted
 * @return A String representing a number filled on the left hand-side with 0
 */
public String Get_Digits(int int_Number)
{
String strAux;
if(int_Number < 10)
strAux = "0" + String.valueOf(int_Number);
else
strAux = String.valueOf(int_Number);
return strAux;
}
* Retrieves the last_modified timestamp value from the http header, needed for the
  ILLUSTRATA ORDB
* in order to determine if a given row/set instance has been modified, in case of
  having
* been previously inserted in the DB.
* @param URL_conn URLConnection permits the retrieval of header information
* @return A String representing the last_modified timestamp value when
  Web page was last modified
* see java.net.URLConnection
*/
public String Get_Last_Modified_Attr(URLConnection URL_conn)
{
  int int_Year; // Year timestamp value
  int int_Month; // Month
  int int_Day; // Day
  int int_Hour; // Hour
  int int_minute; // Minute
  int int_Second; // Second
  int int_MilliSecond; // Milliseconds
  java.util.Date date_Last_Modified; // Date last_modified attribute Web page
  Calendar cal_Last_Modified; // Calendar needed to retrieve individual
  // timestamp values
  String str_Date; // Date timestamp value
  String str_Time; // Time timestamp value
  String str_Last_Modified; // Final valid ILLUSTRATA timestamp value
  // Capture last_modified attribute
date_Last_Modified = new java.util.Date(URL_conn.getLastModified());
  // Convert it into a valid format
cal_Last_Modified = new GregorianCalendar();
cal_Last_Modified.setTime(date_Last_Modified);
  // Get year
  int int_Year = cal_Last_Modified.get(Calendar.YEAR);
  if (int_Year == 1969) { // if last modified has the default value 1969, set it to current timestamp
    date_Last_Modified = new java.util.Date();
cal_Last_Modified.setTime(date_Last_Modified);
    int int_Year = cal_Last_Modified.get(Calendar.YEAR);
  }
  // Get Month: zero-based, and rest
  int int_Month = cal_Last_Modified.get(Calendar.MONTH) + 1;
  int int_Day = cal_Last_Modified.get(Calendar.DAY_OF_MONTH);
  int int_Hour = cal_Last_Modified.get(Calendar.HOUR_OF_DAY);
  int int_Minute = cal_Last_Modified.get(Calendar.MINUTE);
  int int_Second = cal_Last_Modified.get(Calendar.SECOND);
  int int_MilliSecond = cal_Last_Modified.get(Calendar.MILLISECOND);
  // Format valid date timestamp value
  str_Date = int_Year + "-" + Get_Digits(int_Month) + "-" + Get_Digits(int_Day);
  // Format valid time timestamp value
  str_Time = Get_Digits(int_Hour) + ":" + Get_Digits(int_Minute) + ":" + Get_Digits(int_Second) + "." + Get_Digits(int_MilliSecond);
  // Final timestamp value
  str_Last_Modified = str_Date + " " + str_Time;
  return str_Last_Modified;
}
/**
* Parses URL text and captures metadata associated to the Web page if metadata is
* present.
* The format of UniGuide metatags is:
* <pre>
* meta name="Table_Name | Set_Name" Scheme="UniGuide"
* content=" (~ Column_name1 [Alias1]= 'Column_Value1' ~),
* ---------------
* (~ Column_nameN [AliasN]= 'Column_ValueN' ~)
* </pre>
*/
Finally row/set instances may be inserted/updated into the ILLUSTR A ORDB depending on the timestamp value.

* @see JDBC_MetaData_Saver

```java
public void Get_MetaData()
{
    int int_aux;        // Aux int
    int intpos_Meta;    // Pos. Meta tag
    int intpos_Scheme;  // Pos. Scheme Attr.
    int intpos_UniGuide;  // Pos. UniGuide string value
    int intpos_End_Meta;  // Pos. end of UniGuide meta tag
    int intpos_Current;  // Current position
    int int_Begin;       // Beginning extraction position
    int int_End;         // Ending extraction position
    int inti;           // Counter
    int intpos_End_Separator;  // Pos. column separator "~"
    int intpos_Column;  // Pos. column name
    int intpos_Value;   // Pos. column value
    int intpos_Next_Tag; // Pos. next UniGuide meta tag
    int int_No_Meta;    // No meta tag processed
    boolean bol_Found;  // Aux. boolean to indicate an attr. has been found
    boolean bol_Exit;   // Aux. boolean to indicate exit of a loop
    boolean bol_Column_Found; // Indicates if a column has been found
    boolean bol_Name_Attr_Is_Valid; // UniGuide meta tag name attribute is valid
    boolean bol_UniGuide_Metatag;  // UniGuide meta tag has been detected
    String str_Name_Value;    // Name attr. stores table/set table name
    String str_temp;          // Temp string
    String str_Code;          // Code of JDBC operation
    StringBuffer strbuf_Column_Name_List;  // Column name list delimited by ^
    StringBuffer strbuf_Column_Value_List;  // Column value list delimited by ^

    JDBC_MetaData_Saver JDBC_MetaData_Saver_Obj;  // JDBC interface to UniGuide Scheme tables

    // init vars.
    str_temp = new String();
    strbuf_Column_Name_List = new StringBuffer(150);
    strbuf_Column_Value_List = new StringBuffer(500);
    str_Name_Value = ""
    str_Code = ""

    // No UniGuide meta tag found by default
    bol_UniGuide_Metatag = false;
    intpos_Scheme = intpos_UniGuide = intpos_End_Meta = -1;

    // First search for meta-tags
    intpos_Meta = str_Web page_lower.indexOf("<meta",0);
    if(intpos_Meta!=-1) {
        // Search for Scheme attr.
        intpos_Scheme = str_Web page_lower.indexOf("scheme",intpos_Meta);
        if(intpos_Scheme!=-1) {
            // intpos_Meta < intpos_Scheme implicit
            // Search for UniGuide Scheme value
            intpos_UniGuide = str_Web page_lower.indexOf("uniguide",intpos_Scheme);
            if(intpos_UniGuide!=-1 && intpos_Scheme < intpos_UniGuide) {
                // intpos_Scheme < intpos_UniGuide implicit
                intpos_End_Meta = str_Web page_lower.indexOf("">",intpos_UniGuide);
                intpos_Next_Tag = str_Web page_lower.indexOf("<",intpos_UniGuide);
                if(intpos_End_Meta!=-1 && (intpos_Next_Tag!=-1 && intpos_End_Meta <
                intpos_Next_Tag)) {
                    intpos_Next_Tag = -1)
                    // intpos_UniGuide < intpos_End_Meta implicit
                    // It is a valid UniGuide meta tag properly closed
                    bol_UniGuide_Metatag = true;
```
void setSchemeAttr()
{
if(bol_Name_Attr_Is_Valid == true) {
    // search for scheme attr value
    bol_Column_Found = false;
    intpos_Column = str_Web_page_lower.indexOf("(~", intpos_Scheme);
    if(intpos_Column < intpos_Scheme) {
        // Extract name attr. value (table/set table name)
        str_Name_Value = str_Web_page_copy.substring(int_Begin, intpos_Current);
        bol_Name_Attr_Is_Valid = true;
    }
}
}

if(bol_Name_Attr_Is_Valid == true) {
    // search for columns and values
    // search for 1st column from uniguide scheme value and after first ":="
    bol_Column_Found = false;
    intpos_Column = str_Web_page_lower.indexOf("(~", intpos_Scheme + 8);
    if(intpos_Column < intpos_Scheme) {
        // Extract name attr. value (table/set table name)
        str_Name_Value = str_Web_page_copy.substring(int_Begin, intpos_Current);
        bol_Name_Attr_Is_Valid = true;
    }
}
if(intpos_Column!=-1 && intpos_Column < intpos_End_Meta) {
    bol_Column_Found = true;
    strbuf_Column_Name_List.setLength(0);
    strbuf_Column_Value_List.setLength(0);
}

while(bol_Column_Found == true) {
    //extract Column name from position "(~" + 2
    bol_E = false;
    //jump begin separator (~
    int_Begin = intpos_Column + 2;
    int_aux = int_Begin;

    //filter spaces and find 1st char <> of space
    for(inti=int_Begin;inti<int_Limit && bol_Exit == false;inti++) {
        switch (str_Web page_lower.charAt(inti)) {
            case ' ': case '\t':
                continue;
            default:
                int_aux = inti;
                bol_Exit = true;
                break;
        }
    }
    int_Begin = int_aux;

    //begin to extract column name value until space or '](begin of alias) is
    bol_E = false;
    for(inti=int_Begin;inti<int_Limit && bol_E == false;inti++) {
        switch (str_Web page_lower.charAt(inti)) {
            case ' ': case '][': case '\t':
                bol_E = true;
                intpos_Column = inti;
                break;
            default:
                break;
        }
    }

    // Column Name list delimited by ^
    if(strbuf_Column_Name_List.length() > 0)
        strbuf_Column_Name_List.append('^');

    // Extract Column name value
    str_temp = str_Web page_copy.substring(int_Begin,intpos_Column);
    strbuf_Column_Name_List.append(str_temp);

    //extract Column value
    //search for ' char where value of the column begins
    intpos_Value = str_Web page_lower.indexOf('"',intpos_Column + 1);
    intpos_End_Separator = str_Web page_lower.indexOf('"~"',intpos_Column + 1);
    int_Begin = int_End = -1;
    if(intpos_Value!=-1 && intpos_End_Separator!=-1 && intpos_Value <
        intpos_End_Separator) {
        // Properly close column value
        bol_E = false;
        int_Begin = intpos_Value + 1;
        //begin to extract column value until ' or End_Separator is found
        int_End = str_Web page_lower.indexOf('"',int_Begin);
        if(int_End == -1 || int_End > intpos_End_Separator) {
            //There is no ' to end the value so get values until separator is found
            int_End = str_Web page_lower.indexOf('"~"',int_Begin);
        }
        if(int_Begin == -1 || int_End == -1) {
            // Column value was not properly closed between "
            str_temp = "null";
        } else {
            // Capture column value
            str_temp = str_Web page_copy.substring(int_Begin,int_End);
            str_temp = str_temp.trim();
            // If noting specified then null value by default
            if(str_temp.length()==0)
                str_temp = "null";
        }
    }
    // Append to column value list delimited by ^
    if(strbuf_Column_Value_List.length() > 0)
        strbuf_Column_Value_List.append('^');
strbuf_Column_Value_List.append(str_temp);

// search for next column
bol_Column_Found = false;
if(intpos_End_Separator!= -1) {
    // we continue the search because the meta tag column was closed properly
    intpos_Column = str_Web_page_lower.indexOf("(~", intpos_End_Separator + 1);
    if(intpos_Column!=-1 && intpos_Column < intpos_End_Meta)
        // there is still more columns to process
        bol_Column_Found = true;
}
} // endwhile bol_Column_Found

// add 1 to number of meta tags processed
Robot_Stats_Obj.Set_Value('M',1);

// Pass parameters to the database and test code returned
JDBC_MetaData_Saver_Obj = new JDBC_MetaData_Saver(URL_app.toString(),str_Name_Value,
strbuf_Column_Name_List.toString(),strbuf_Column_Value_List.toString(),str_Last_Modified,
Robot_Stats_Obj);
// Execute DML Phase1/PhaseII statements
str_Code = JDBC_MetaData_Saver_Obj.Set_SQL_Statement();
JDBC_MetaData_Saver_Obj = null;
System.gc();
}
} // endif bol_Name_Attr_Is_Valid

// Display Meta data stats.
int_No_Meta = Robot_Stats_Obj.Get_Value('M');
Get_MetaData_Statistics(int_No_Meta,str_Name_Value,str_Code);
panSeparator.setBackground(Color.lightGray);

// search for the next meta-tag
bol_UniGuide_Metatag = false;
intpos_Meta = str_Web_page_lower.indexOf("<meta", intpos_End_Meta + 1);
if(intpos_Meta!=-1) {
    intpos_Scheme = str_Web_page_lower.indexOf("scheme", intpos_Meta);
    if(intpos_Scheme!=-1) {
        // intpos_Meta < intpos_Scheme implicit
        intpos_UniGuide = str_Web_page_lower.indexOf("uniguide", intpos_Scheme);
        if(intpos_UniGuide!=-1 & intpos_Scheme < intpos_UniGuide) {
            // intpos_Scheme < intpos_UniGuide implicit
            intpos_End_Meta = str_Web_page_lower.indexOf(">", intpos_UniGuide);
            if(intpos_End_Meta!=-1 & intpos_End_Meta < intpos_Next_Tag)
                // intpos_Next_Tag == -1)
                    bol_UniGuide_Metatag = true;
        }
    }
} // endwhile bol_UniGuide_Metatag

/**
 * Parses Web page contents and extracts URLs from href tags, image maps
 * and frames and converts them into absolute URLs taking into account if
 * the Web page contains a base href tag or the location of the URL of
 * the parent Web page.
 */

public void Get_URLs()
{
    int intpos_base; // Pos. HREF BASE

    // init global vars
    bol_Base = false;
    URL_Base = URL_app;

    // init local vars
    intpos_base=0;

    // Get Base HREF if it exists, otherwise base URL will be the parent Web page
    intpos_base = Get_Base_HREF();
if(bol_Base == false) {
    // Reset to URL_app as there is no base href or it is invalid
    URL_Base = URL_app;
    intpos_base = 0;
}

// get URLs from HREF tags: links and image maps
Get_HREF_URLs(intpos_base);

// get URLs from frames src
Get_Frames_URLs();
} // end-Get_URLs

/**
* Adds a new absolute URL to the ILLUSTRAT ORDB.
* Converts relative URLs into absolute if relative and stores Absolute URLs into the
* URL_catalogue and URL_to_visit tables via the JDBC interface.
* 
* @param URL_Base_locBase URL needed to generate absolute URLs for relative URLs
* Has a null value for Absolute URl
* @param str_URL_loc URL string which may represent an absolute or relative URL
* @see JDBC_URL_Saver
* */
public void Add_URL_Absolute(URL URL_Base_loc, String str_URL_loc)
{
    int int_err; // Error Code indicating validity of an absolute URL
    int int_Code; // Code returned by an insert operation
    boolean bol_Found; // Boolean indicates if a URL exists in DB
    String str_URL; // URL string to be saved
    URL URL_Absolute; // Absolute URL formed

    try {
        if(URL_Base_loc == null)
            // Absolute URL
            URL_Absolute = new URL(str_URL_loc);
        else
            // Relative URL
            URL_Absolute = new URL(URL_Base_loc,str_URL_loc);

        // Check if URL is valid: inside the University domain and pointing to a Web page
        int_err = Check_Absolute_URL(URL_Absolute);

        if(int_err!=0) {
            str_URL = URL_Absolute.toString();

            // Create JDBC Interface
            JDBC_URL_Saver JDBC_URL_Saver_loc = new JDBC_URL_Saver(Robot_Stats_Obj);

            // Search URL in the URL_catalogue table
            bol_Found = JDBC_URL_Saver_loc.Exists_URL(str_URL);

            // Save if not found in ORDB
            if (bol_Found == false)
                int_Code = JDBC_URL_Saver_loc.Save_URL(str_URL);

            JDBC_URL_Saver_loc = null;
            System.gc();
        }
    }
    catch (MalformedURLException url_excep) {
        txta_URLs_Visited.append("Exception: "+url_excep.toString() + "\n\n");
        int_Errors++;
    }
}

/**
* Fetches all URLs referenced by links (HREF tags) and image maps from the contents
* of a Web page.
* 
* @param intpos_base Starting position to fetch URLs
* '<ul>
* '<li>' For Web pages with a BASE HREF tag it has a value > 0
* '<li>' Otherwise has a value of 0
* '</ul>
public void Get_HREF_URLs(int intpos_base) {
    int inti; // Counter
    int int_Begin; // Starting pos. extraction URL
    int intpos_HREF; // Pos. HREF tag
    int intpos_Current; // Current pos.
    int intpos_Anchor; // Pos. interior Anchor #
    int int_URL_type; // URL type 0 Incorrect, 1 Absolute 2 Relative
    boolean bol_Found; // Indicates an item has been found
    boolean bol_Exit; // Indicates loop exit
    boolean bol_Interior; // Indicates if link is an interior link, therefore
    // it should be discarded
    String str_URL; // URL string captured
    String str_URL_aux; // Lowecase URL string captured

    // search for links and clickable images with href
    if (bol_Base == true) // skip HREF from base HREF tag
        intpos_HREF= str_Web page_lower.indexOf("href",intpos_base);
    else //start from beginning
        intpos_HREF= str_Web page_lower.indexOf("href",0);

    // while there are hrefs tags
    while(intpos_HREF!=-1) {
        // skip HREF string
        intpos_Current = intpos_HREF + 4;
        bol_Found = bol_Exit = false;
        // search for "=" character if href tag is valid
        for(inti=intpos_Current;inti<int_Limit && bol_Exit == false;inti++) {
            switch (str_Web page_lower.charAt(inti)) {
                case ' ':
                    // ignore spaces and continue parsing
                    continue;
                case '=':
                    // valid href tag found exit
                    bol_Found = true;
                    intpos_Current = inti + 1;
                    bol_Exit = true;
                    break;
                default:
                    // invalid href tag exit
                    bol_Exit = true;
                    break;
            }
        }
        // If href tag is valid continue parsing
        if(bol_Found == true) {
            // Now filter spaces, ' and "
            bol_Interior = bol_Exit = false;
            // search for first character <> space, ' and "
            for(inti=intpos_Current;inti<int_Limit && bol_Exit == false;inti++) {
                switch (str_Web page_lower.charAt(inti)) {
                    case ' ': case '"': case '\':
                        // ignore spaces and " and continue parsing
                        continue;
                    case '#':
                        // href is an interior link, exit
                        bol_Interior = true;
                        bol_Exit = true;
                        break;
                    default:
                        // found 1st char of URL, exit
                        intpos_Current = inti;
                        bol_Exit = true;
                        break;
                }
            }
            // If not an interior link, continue parsing
            if(bol_Interior == false) {
                bol_Exit = false;
                // start from 1st character of URL
            }
        }
    }
}
int_Begin = intpos_Current;
// search for last character URL
for(inti=intpos_Current;i<int_Limit && bol_Exit == false;i++) {
    switch (str_Web_page_lower.charAt(inti)) {
        case ' ': case '"': case '\': case '>':
            // if space, " or end of href tag > is found, end of parsing
            bol_Exit = true;
            intpos_Current = inti;
            break;
        default:
            // continue parsing chars of URL
            break;
    }
}

// extract URL href tag from starting pos.(int_Begin) to end(intpos_Current)
str_URL = str_Web_page_copy.substring(int_Begin,intpos_Current);
// same URL but lowercase in order to test later
str_URL_aux = str_Web_page_lower.substring(int_Begin,intpos_Current);

//test if it contains an anchor
intpos_Anchor = str_URL.indexOf("#");
if(intpos_Anchor!=-1)
    // Discard anchor
    str_URL= str_URL.substring(0,intpos_Anchor);

// Check if URL is correct and what type it is if correct
if((int_URL_type = Check_URL_Type(str_URL_aux)) != 0) {
    if(int_URL_type == 2) {
        // URL is relative
        if(bol_Base == true)
            // BASE HREF tag
            Add_URL_Absolute(URL_Base,str_URL);
        else
            // Non Base HREF tag
            Add_URL_Absolute(URL_app,str_URL);
    }
    else
        // int_URL_type = 1 Absolute
        Add_URL_Absolute(null,str_URL);
}
} //end -bol_Interior
} //end -bol_Found

// Search for next href tag
intpos_HREF = str_Web_page_lower.indexOf("href",intpos_Current);
} //end

/**
 * Fetches all URLs referenced by frame src tags from the contents of a Web page.
 * @see #Add_URL_Absolute
 */
public void Get_Frames_URLs()
{
    int inti;
    // Counter
    int int_Begin;
    // Starting pos. extraction URL
    int int_URL_type;
    // URL type 0 Incorrect, 1 Absolute 2 Relative
    int intpos_FRAME;
    // Pos. frame tag
    int intpos_Current;
    // Current pos.
    int intpos_Anchor;
    // Pos. Anchor in captured URL string
    boolean bol_Found;
    // Indicates item has been found
    boolean bol_EXIT;
    // Indicates loop exit

    String str_URL;
    // URL string captured
    String str_URL_aux;
    // Lowecase URL string captured

    // search for frames
    intpos_FRAME= str_Web_page_lower.indexOf("<frame",0);
    // While frame tag is found
    while(intpos_FRAME!=-1) {
        //there are frames in the Web page, then search the source Web page
        intpos_Current= str_Web_page_lower.indexOf("src",intpos_FRAME);
        if(intpos_Current!=-1) {
            //
bol_Found = bol_Exit = false;
// Skip src string
intpos_Current = intpos_Current + 3;
// search for the "=" character
for (int=intpos_Current;int<int_Limit && bol_Exit == false;int++) {
    switch (str_Web page_lower.charAt(int)) {
        case '=':  
            bol_Found = true;
            intpos_Current = int + 1;
            bol_Exit = true;
            break;
        default:  
            bol_Exit = true;
            break;
    }
}
// Frame tag is correct, continue parsing
if (bol_Found == true) {
    // Search for the first character <> space, ' OR "
    bol_Exit = false;
    for (int=intpos_Current;int<int_Limit && bol_Exit == false;int++) {
        switch (str_Web page_lower.charAt(int)) {
            case '"': case ' ': case '\':
                continue;
            default:
                intpos_Current = int;
                bol_Exit = true;
                break;
        }
    }
    // Last char of URL is found
    bol_Exit = false;
    int_Begin = intpos_Current;
    // Search end of URL string
    for (int=intpos_Current;int<int_Limit && bol_Exit == false;int++) {
        switch (str_Web page_lower.charAt(int)) {
            case '"': case '<': case ' ': case '/':
                bol_Exit = true;
                intpos_Current = int;
                break;
            default:
                break;
        }
    }
    // Extract URL string
    str_URL = str_Web page_copy.substring(int_Begin, intpos_Current);
    // Same URL but lowercase in order to test later
    str_URL_aux = str_Web page_lower.substring(int_Begin, intpos_Current);
    // Test if it contains an anchor
    intpos_Anchor = str_URL.indexOf("#");
    if (intpos_Anchor!=-1) {
        // Discard anchor
        str_URL = str_URL.substring(0, intpos_Anchor);
        // Check if URL is correct and what type it is if correct
        if ((int_URL_type = Check_URL_Type(str_URL_aux)) != 0) {
            if (int_URL_type == 2) {
                // URL is relative
                if (bol_Base == true) {
                    // URL is relative to BASE href
                    Add_URL_Absolute(URL_Base, str_URL);
                } else {
                    // URL is relative to current location
                    Add_URL_Absolute(URL_app, str_URL);
                }
            } else {
                // int_URL_type == 1, URL is absolute
                Add_URL_Absolute(null, str_URL);
            }
        }
    }
    // Get next frame tag
    intpos_FRAME = str_Web page_lower.indexOf("<frame", intpos_Current);
}  // end-while
** Get_Base_HREF() **

* Searches for a BASE HREF tag in order to set the Base URL needed to form absolute URL from relative ones.
* Sets bol_base boolean value to true if a BASE HREF tag is found.
* In addition URL_Base is set to the BASE HREF URL value.
* Otherwise URL_Base still holds the same value as the URL of the Web page.

* @return int representing position where BASE HREF tag is found:
  * <ul>
  * <li> -1 not found
  * <li> >0 found
  * </ul>

```java
public int Get_Base_HREF()
{
    int inti; // Counter
    int int_Begin; // Beginning pos. BASE URL string extraction
    int intpos_BASE; // Pos. BASE HREF tag
    int intpos_Current; // Current pos.
    int intpos_End; // Pos. end of BASE HREF tag
    boolean bol_Found; // Indicates item found
    boolean bol_Exit; // Indicates loop exit
    String str_Base_URL; // Base URL string

    // test if there is a base tag for HREFs
    intpos_BASE = str_Web_page_lower.indexOf("<base",0);
    intpos_Current = -1;
    // If BASE HREF found
    if(intpos_BASE!=-1) {
        intpos_Current = str_Web_page_lower.indexOf("href", intpos_BASE + 5) + 4;
        intpos_End = str_Web_page_lower.indexOf(">", intpos_BASE + 5);
        bol_Found = false;
        // If href tag is found after closing tag
        if(intpos_Current > intpos_End)
            // This href does not belong to base href, base is not well constructed
            // do not enter in following loop
            bol_Exit = true;
        else
            bol_Exit = false;
        // BASE HREF well defined, continue parsing, search for = char
        for(inti=intpos_Current;inti<int_Limit && bol_Exit == false;inti++) {
            switch (str_Web_page_lower.charAt(inti)) {
                case ' ': continue;
                case '=':
                    bol_Found = true;
                    intpos_Current = inti + 1;
                    bol_Exit = true;
                    break;
                default:
                    bol_Exit = true;
                    break;
            }
        }
        // Well constructed, continue parsing
        if(bol_Found == true) {
            bol_Exit = false;
            // Filter spaces, ', and " chars
            for(inti=intpos_Current;inti<int_Limit && bol_Exit == false;inti++) {
                switch (str_Web_page_lower.charAt(inti)) {
                    case ' ': case "": case \\
                        continue;
                    default:
                        intpos_Current = inti;
                        bol_Exit = true;
                        break;
                }
            }
            // 1st char BASE URL string found, now search for end char
            bol_Exit = false;
            int_Begin = intpos_Current;
            return int_Begin;
        }
    }
    // Not found
    return -1;
}
```
for(int i=intpos_Current; i<int_limit && bol_exit == false; i++) {
    switch (str_web_page_lower.charAt(i)) {
        case ' ': case '"': case '\': case '>':
            bol_exit = true;
            intpos_current = i;
            break;
        default:
            break;
    }
    // Extract now BASE URL string
    str_base_url = str_web_page_copy.substring(int_begin, intpos_current);
    try {
        // Generate a new Base URL <> URL_app (Web page)
        URL_base = new URL(str_base_url);
        bol_base = true;
    } catch (MalformedURLException url_exception) {
        txta_urls_visited.append("nException: " + url_exception.toString() + "n\n";
        robot_stats_obj.set_value('E', 1);
        intpos_current = -1;
    }
}
} // end-bol_found
} // end-int_pos

// Position is -1 or end of BASE HREF tag pos.
return intpos_current;
} // end-get_base_href

/**
 * Tests if a given URL string is correct and what kind of URL it represents.
 * @param str_url URL string to be checked
 * @return int code indicating type of URL:
 * <ul>
 * <li>0 Incorrect
 * <li>1 Correct Absolute URL
 * <li>2 Correct Relative URL
 * </ul>
 *
 * @return
 */
public int check_url_type(String str_url) {
    // 0 incorrect, 1 Absolute, 2 relative
    // test if incorrect protocol
    if(str_url.startsWith("mailto:"))
        return 0;
    if(str_url.startsWith("telnet:"))
        return 0;
    if(str_url.startsWith("gopher:"))
        return 0;
    if(str_url.startsWith("news:"))
        return 0;
    if(str_url.startsWith("wais:"))
        return 0;
    // incorrect protocol
    return 0;
    // test if ftp points to incorrect file (not an htm,html)
    if(str_url.startsWith("ftp")) {
        if(!str_url.endsWith("htm") && !str_url.endsWith("html"))
            // ftp can only point to html files
            return 0;
        else
            // ftp points to Web pages, therefore URL is absolute
            return 1;
    }
    // dealing only with HTTP protocol
    if(!str_url.startsWith("http"))
        return 2;
    return 1;
}

/**
 * Tests if a given Absolute URL is correct. That is, it belongs to the university
* domain and points to only Web pages.
  *
  * @param URL_aux URL to be checked
  * @return int code indicating status of URL:
  * <ul>
  * <li>0 Incorrect
  * <li>1 Correct Absolute URL.
  * </ul>
  */

public int Check_Absolute_URL(URL url_aux)
{
  int intpos_Dot_Extension;
  int int_Limit;
  String str_Pat; // File part of URL
  String str_Ext; // Extension part of URL file
  String str_Absolute_URL; // Absolute URL string

  str_Absolute_URL = url_aux.toString();
  if(str_Absolute_URL.indexOf(str_Domain) == -1)
    // URL points to a server outside the domain
    return 0;
  // Get file part of URL, which includes directory and file or / (directory)
  str_Path_File = url_aux.getFile();
  if(str_Path_File.equals("/"))
    // no file, getFile returns a "/"
    // URL is absolute, with no file and correct
    return 1;
  else {
    // Has a file, now test if file has extension
    intpos_Dot_Extension = str_Path_File.indexOf(".");
    // If extension is found
    if(intpos_Dot_Extension != -1) {
      // test file extension
      int_Limit = str_Path_File.length();
      str_Ext = str_Path_File.substring(intpos_Dot_Extension + 1, int_Limit);
      if(!str_Ext.equals("html") && !str_Ext.equals("htm"))
        // incorrect extension, should only point to Web pages
        return 0;
    } /* If reaches here URL is absolute, a directory/Web page and correct */
    return 1;
  }
} //end-Test_URL_ext

/**
 * Displays Metadata Statistics:
 * <ul>
 * <li>No meta tag processed
 * <li>Instance type detected
 * <li>Status Code returned by the ILLUSTR ORDB
 * </ul>
 *
 * @param int_No_Meta_loc Number of UniGuide meta tag processed
 * @param str_Table_loc Instance type detected
 * @param str_Code_loc Status code returned by ILLUSTR
 *
 */

public void Get_MetaData_Statistics(int int_No_Meta_loc, String str_Table_loc,
                      String str_Code_loc)
{
  String str_Line;

  str_Line = String.valueOf(int_No_Meta_loc) + "\t\t" + str_Table_loc + "\t\t" + str_Code_loc;
  txta_Metadata.append(str_Line + "\n");
}
} // end-class