Chapter 7
Bibliographical Database Management System Integration

7.1 Preamble.

The basics of the BDMS model design and system integration is studied through this chapter. Experience in designing and writing application programs with object-oriented essence reveals problems in connecting application programs to Bibliographical Database Management System services. This is due to the difference between the declarative description of data assumed in a BDMS and the behavioral, encapsulated format in programming language. To overcome this problem, the integration of BDMS services programming language must be improved. A method is proposed to achieve seamless integration of BDMS functionality with application code using inheritance. The language must provide i) multiple inheritance, allowing the objects to be stored in the database to inherit the necessary methods, and ii) General Class, in order to document what operations each object to be stored in the database and must export for use by the BDMS (e.g. access operations). Most current object-oriented languages do not provide both features and thus make it difficult to add a DBMS package. Using multiple inheritance and general classes, a BDMS package can be written in the same language as the application program, and can be added to an object-oriented programming system.

During my study I use Oracle8i enterprise edition as the database engine. I use version 8i because when I started the research in the yare 2002, 8i was the latest version of oracle, although the developed codes are also be compatible with 9i and later version. I also use the significant features for online transaction processing (OLTP) and data warehouse applications. The performance, scalability, and availability necessary to support very large database (VLDB) and mission-critical applications are also ensured and capability of dealing with all types of data through developed tools. Oracle8i is much more than just a simple relational data store. The advances "enabling"
technologies including Clint/server, data warehousing, and online analytical processing of the database engine are used to facilitate retrieving a data tuple related to a student/staff registered to the library as borrower.

For my study I use Windows 2000 (with Window NT flavor) to integrate with Oracle8i enterprise edition. It provides full native integration with Microsoft Transaction Server (MTS) in the Windows NT environment. Application development is made simpler by the oracle Application for Web enable client who provides developers with a GUI tool. The codes are written for framework including object classes for connectivity and data access from databases.

Finally, Oracle8i includes Oracle enterprise manager which is the comprehensive management framework for administrating/managing the database and application environment. The enterprise manager presents an easy-to-use centralized console, a rich set of management tools, and the extensibility to detect and solve any problems that may arise. It also includes several administrative applications for performing day-to-day tasks for databases and applications such as scheduling backup routines. Oracle8i provides the lowest cost platform for developing and deploying applications on the Internet.

Figure 7.1 - Oracle Enterprise Manager.
The integration of a DBMS as application code is not a trivial problem. Solutions reported use object-based, but not object-oriented languages \(^1\) and a relational data model with a global schema on the level of the application. This contradicts the object-oriented view of data encapsulated in modules. My own efforts to integrate develop end BDMS in an object-oriented programming essence with an understanding how to combine separate object definitions with database functions, the problems are handled by using a language that provides multiple inheritance. In the proposed design WebDB is used to design the front end GUI.

Figure 7.2 - WebDB GUI interface.

Figure 7.3 - Behavioral view of WebDB.
7.2 BDMS Architecture.

Physically, in its simplest form, the BDM database is nothing more than a set of files somewhere on disk. The physical location of these files is irrelevant to the function (although important for the performance) of the database. The files are binary files those can only access using the kernel software. Querying data in the database files is typically done with procedural queries[^4].

Logically, the database is divided into a set of user accounts (schemas) identified (Figure 7.7) by a user name and password unique to that database. Tables and other objects are owned by one of these users and access to the data is only available by logging in to the database using an username and password. Without a valid username and password for the database access to database[^10] will be shorn off.

7.3 Data Dictionary.

A comprehensive set of tables and views created and updated by the database server which contains administrative information about users, data storage and privileges to track information e.g. (who can log into the database and what files are required to run the database). It is installed when oracle is initially installed and is a central source of information for the database server itself and for all users of BDMS. The tables are automatically maintained by system. It is sometimes referred to as the catalog[^21].

Database Files: There are three major sets of files on disk that compose a database:

- Database files
- Control files
- Redo logs

The database files hold the actual data and are typically the largest in size (from a few megabytes to many gigabytes). The other files (control files and redo logs) support the
rest of the architecture. The database files are fixed in size and never grow bigger than the size at which they were created.

7.4 Structured Query Language (SQL)

An attempt is made to classify and hierarchically arrange the different SQL queries used for development the BDM system. The procedural language used as DDL statement to define data object like create, alter or drop of tables and to define maintain and drop schema objects when they are no longer needed. DDL statements also include statements that allows DBA to grant other users the privileges or rights to access the database and specific objects within the database.

The meta-object protocol for handling access to objects is used which makes it possible to specify specific algorithms for object caching and synchronization of...
The persistence of objects/object types and associative object the two important features of OODB are used during the integration. Persistence means that every object or instance of a user-defined type that is created must remain beyond volatile memory and be stored permanently on disk until the user declares that it to be deleted. The definitions of user-defined data types are also be stored in files and be read and interpreted automatically by the database management system.

**Transaction Control Statements:** Transaction control statements are used to ensured the change made by DML statements. It allow the user or application developer to group changes into logical transactions. e.g include COMMIT, ROLLBACK, and SAVEPOINT.

**Session Control Statements:** Session control statements are used to allow the user to control the properties of his current session including enabling and disabling roles and changing language settings. e.g ALTER SESSION and SET ROLE.

**System Control Statements:** System control statements are used to change the properties of the database server instance. e.g ALTER SYSTEM.

### 7.4.1 Embedded SQL Statement.

Embedded SQL statements incorporate DDL, DML, and transaction control statements in a procedural language program. i.e. OPEN, CLOSE, FETCH, and EXECUTE.

**Transactions:** A transaction is a logical unit of work that comprises one or more SQL statements executed by a single user. According to the ANSI/ISO SQL standard, with which oracle is compatible, a transaction begins with the user’s first executable SQL statement. A transaction ends when it is explicitly committed or rolled back by that user.\(^\text{42}\)
Let consider developed BDMS database, when library user borrowed a book from the library the transaction might consist of three separate operations: check the User etitle, increase the issue counter, and record the transaction in the user_issue schema. Database Server must guarantee that all three SQL statements are performed to maintain the accounts in proper balance. When something prevents one of the statements in the transaction from executing (such as a hardware failure), the other statements of the transaction must be undone which is called rolling back. If an error occurs in making either of the updates then neither update is made.

Classification of Queries

![Diagram of Data Definition Language (DDL)]

**Figure 7.5- Data Definition Language.**

SQL databases are relational databases this means simply that data is stored in a set of simple relations. A database can have one or more tables. Each table has columns and rows. A table that has an u_code field for example a column called U_name and each row in that column would be an U_code’s U_Name.
During the design table data is manipulated with SQL statements. Data definition language (DDL) statements are used to create/alter database and tables. DDL statements include statements for creating and altering databases and tables. The user of the database having the roll privilege could update, delete, or retrieve data in a table with data manipulation language (DML). DML statements include statements to alter and fetch data. The most common SQL statement is the SELECT statement which allows retrieving data from the database. In addition to SQL statements, NODAL (New Object Oriented Database Language) for programming SQL statement to control the flow of a SQL is proposed. It allows controlling the flow of a SQL program to use variables, and to write error-handling procedures.

**Commit and Rollback:** Database changes are not saved until the user explicitly decides that the insert, update, and delete statements should be made permanent. Till that point the changes are in a pending status and any failures such as a machine crash will reserve the changes. A transaction is an atomic unit of work comprising one or more SQL statements; it begins when the user first connects to the database and ends when a COMMIT or ROLLBACK statement is issued. Upon a COMMIT or ROLLBACK, the next transaction automatically begins. Committing a transaction makes changes permanent in the entire transaction to the database and once committed the changes cannot then be reserved. Rolling back reserves all the inserts, updates, deletes in the transaction; again once rolled back those changes cannot then be committed. The
transaction or a unit of concurrency or work nothing smaller or less then a transaction can be occurred. That is no one can halfway change a piece of data. All transaction must be atomic in that each individual transaction either complete or does not complete. The transaction is smaller unit of concurrence like an atom in physics. It is al-or-nothing. A transaction that couplet is said to be committed and one that does not is rolled-backed 24,32.

```
SQL> Create rollback segment r05
    Tablespace RBS
    Storage ( initial 10M
    Next 10M
    Minextents 2
    Maxextents 100);
```

**Data Integrity:** Data integrity is about enforcing data validation rules such as checking that a percentage amount is between 0 and 100 to ensure that invalid data does not get into the tables. Historically, these rules are enforced by the application programs themselves (and the same rules were checked repeatedly in different programs). Database server, however, enables the DBA to define and store these rules against the database related objects so that DBA needs to code them only once. This checking takes the form of integrity constraints and database triggers.

7. 5 NODAL Stored procedures.

A NODAL stored procedure, function, or package is used as a procedure language program unit that:

- Has a name.
- Can take parameters and can return values.
- Is stored in the data dictionary.
- Can be called by many users.

When defining exceptions, I declare them in the declaration part of a block and define them in the exception part of the block. An example follows:
create or replace procedure orderrpt(vcode in varchar2 default null, ono in varchar2 default null)
as
cursor vcur is select ven_name, ven_address from vendors
where ven_code = vcode or ven_code = upper(vcode);
cursor bcur is select title, author, edition, quantity, pub_code, price, curr_code from bkreq
where cntrl_no = any(select cntrl_no from book_accounts where order_no = ono);
cursor deptcur is select a.title, a.dept_code, b.dept_name from book_accounts where
order_no = ono and a.dept_code = b.dept_code order by a.dept_code;
number := 0;
p varchar2(50);
v varchar2(10);
currency varchar2(3) default null;
deptcode varchar2(10) default 'rds';
begin
select unique ven_code into v from bk where order_no = vcode;
htp.htmlopen;
htp.headopen;
htp.title('Order Form');
htp.headclose;
htp.bodyopen;
htp.p('<center>CENTRAL LIBRARY<BR>INDIAN INSTITUTE OF
TECHNOLOGY,KHARAGPUR<BR><BR>
<U>ORDER FORM</U></CENTER><br><br>
Ref No. :IIT/LIB/A/2005-06/XXXX/||vcode||'||ono||' Date
||sysdate||'"</PRE>');</htp.tableopen;
for vrec in vcur
loop
htp.tableRowOpen;
htp.tableData(vrec.ven_name);
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData(vrec.ven_address);
htp.tablerowclose;
end loop;
htp.tableclose;
htp.br;
htp.p('<pre>Dear Sirs, Please supply the following publication(s) to the extent shown below. (The terms & conditions are attached herewith).</pre>');
htp.p('<hr>');
htp.tableopen;
htp.tablerowopen;
htp.tabledata('S.No.');
htp.tabledata('Title/Publisher');
htp.tabledata('Author');
htp.tabledata('Edition');
htp.tabledata('Quantity');
htp.tabledata('Price per Copy');
htp.tabledata('');
htp.tablerowclose;
htp.tableopen;
htp.tablerowopen;
htp.p('<td colspan="7"><hr></td>');
htp.tablerowclose;
for brec in bcur
loop
select decode(brec.curr_code,null,'INR',brec.curr_code)into currency from
dual;
if brec.pub_code is not null then
select pub_name into p from publishers where pub_code=brec.pub_code;
else
  p:=null;
end if;
x:=x+1;
htp.tablerowopen;
htp.tabledata('<center>'||x'||'</center>');</p>
htp.tabledata(brec.title||'/');</p>
htp.tabledata('<center>'||brec.author'||'</center>');</p>
htp.tabledata('<center>'||brec.edition'||'</center>');</p>
  htp.tabledata('<center>'||brec.quantity'||'</center>');</p>
  htp.tabledata('<center>'||brec.price||'(,||currency||)'</center>');</p>
htp.tablerowclose;htp.tablerowopen;
  htp.tabledata('');</p>
htp.tabledata(p);
htp.tablerowclose;
end loop;
htp.tableclose;
htp.p('<br><hr><br>');
htp.p('<pre> Yours Sincerely,

Asstt. Librarian/Chairman,Central Library

Copy Forwarded to:

1.Asstt.Registrar (A/cs.)
2.Vendors Copy
3.Deptt.Library Representative
4.Order File
5.Master File

<br></pre>');
Department Wise Books are as follows:

<table>
<thead>
<tr>
<th>Dept. Name</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>-------------</td>
<td>-------</td>
</tr>
</tbody>
</table>

Database server enabled the capability to store Declarative Language (DL) blocks as database objects in the form of stored procedures, functions and database packages. Now portions of the application logic especially that requiring database access can reside where they are processed on the server. Using stored procedures significantly increases the efficiency of a client/server system for several reasons as it generates minimal network traffic, it provides a convenient and effective security mechanism, and the need to parse and compile the DL at run time is alleviated.

### 7.6 Database Triggers.

A database trigger is a DL block defined to automatically insert, update, and delete statements against a table. Database triggers resemble stored procedures in that they are database resident DL blocks; the difference between the two is that triggers are fired automatically by the RDBMS kernel in response to a commit time event (such as...
Triggers are used to enforce complex integrity checking, perform complex auditing and security functions, and implement application alerts and monitors. Like stored procedures, database triggers greatly reduce the amount of code and processing that is necessary in the client portion of an application. Supports both statement level (those which affect multiple rows) and row-level triggers to be used in conjunction with one another. Unlike database triggers, since procedures on the database are stored in a compiled form, capable to put the longer code segments into a procedure and then call the procedure from the database trigger. Triggers are similar to stored procedures. A trigger stored in the database can include SQL, NODAL/PL/SQL or Java statements to execute as a unit and invoke stored procedures. However, procedures and triggers differ in the way that they are invoked. A procedure is explicitly executed by a user, application, or trigger. Triggers are implicitly fired by Oracle when a triggering event occurs, no matter which user is connected or which application is being used. When defining a trigger, the trigger timing whether the trigger is specified for an action to be executed before or after the triggering statement. BEFORE and AFTER apply to both statement and row triggers.

BEFORE and AFTER triggers fired by DML statements is defined only on tables, not on views. However, triggers on the base tables of a view are fired if an INSERT, UPDATE, or DELETE statement is issued against the view. BEFORE and AFTER triggers fired by DDL statements are defined only on the database or a schema not on particular tables.

Figure 7.6 - Triggers.
Example:
create trigger y before insert on reqbk
for each row
when (new.a is null)
begin
select x.nextval into :new.a from dual;
end;

7.6.1 Large objects (LOBs).

To store non-traditional data types like images. Database server is sound enough to store multiple LOB columns in each table. The LOB (large object) datatypes BFILE, BLOB, CLOB, and NCLOB let store blocks of unstructured data (such as text, graphic images, video clips, and sound waveforms) up to four gigabytes in size. And they allow efficient, random, piece-wise access to the data.

The Use of LOBs: As applications evolve to encompass increasingly richer semantics they encounter the need to deal with various kinds of data -- simple structured data, complex structured data, semi-structured data and unstructured data. Traditionally, the relational model had been very successful at dealing with simple structured data -- the kind which can be fit into simple tables. Oracle has added Object-Relational features so that applications can deal with complex structured data -- collections, references, user-defined types and so on. The queuing technologies deal with messages and other semi-structured data. LOBs are designed to support the last piece - unstructured data.

Unstructured data cannot be decomposed into standard components. Data about an employee can be 'structured' into a Name (probably a character string), an Id (likely a number), an employeecode and so on. In case a picture, the data really consists of a long stream of 0s and 1s. These 0s and 1s are used to switch pixels on or off so that it is observed the picture on a display but they can't be broken down into any finer structure in terms of database storage.
7.6.2 Unstructured Data.

Also interesting is that unstructured data such as text, graphic images, still video clips, full motion video, and sound waveforms tend to be large -- a typical employee record may be a few hundred bytes but even small amounts of multimedia data can be thousands times larger.

**Unstructured Data in System Files Need Accessing from the Database.** Finally, some multimedia data may reside on operating system files and it is desirable to access them from the database.

7.6.3 LOB Data type Helps Support Internet Applications.

Lately, with the growth of the internet and content-rich applications, it has become imperative that the database support a datatype to fulfill the following:

- Can store unstructured data
- Is optimized for large amounts of such data
- Provides a uniform way of accessing large unstructured data within the database or outside.

7.7 Object Oriented Programming (OOP).

Support of object structures was offered as an option to Oracle8i programmers to create user defined data types, complete with their own methods and attributes. Object Views facilitate the use of object oriented programs to use relational data stored in the database. Objects can be stored as varying arrays (VARRAYs), nested tables, or index organized tables (IOTs).
7.8 3rd Generation Languages.

Programmers can interact with the oracle databases from C, C++, java, COBOL, or FORTRAN applications by embedding SQL in those applications. Prior to compiling these applications using a platform's native compilers, one must run the SQL code through a compiler that replaces SQL statements with library calls the native compiler can accept.

Call Interface (CI): to define SQL statements within host language character strings and then explicitly parse the statements, bind variables for the statements and execute them which are used through the study. National Language Support (NLS) which provides character sets and associated functions such as date and numeric formats for a variety of languages, this aspect also have taken care.

To ensure the complete solution for building, deploying, and proactively monitoring Web database applications and content-driven Web sites WebDB is used.

7.9 Implementation the BDMS

Starting Database Application

![Starting Database Application](image)

**Figure 7.7- Starting Database Application.**

1. The following information is required to installed WebDB. From any client machine with the privilege of DBA, listeners of the database can be operated.
2. Start Web browser.

3. In the Address or Location field of browser, type the URL of the WebDB home page and press the Enter or Return key.

The following tables prepared for the database for BDMS with embedded SQL command. I want to mention that the PL/SQL command is also used through WebDB for this purpose.

![Figure 7.8 - PL/SQL command through WebDB.](image-url)
### SQL> create table ven

1. `ven_code` VARCHAR2(4) NOT NULL,
2. `ven_name` VARCHAR2(25),
3. `ven_address` VARCHAR2(200),
4. `ven_phone` VARCHAR2(25),
5. `ven_fax` VARCHAR2(25),
6. `ven_email` VARCHAR2(45),
7. `ven_status` VARCHAR2(12),
8. `ven_site` VARCHAR2(40).

### SQL> describe ven;

<table>
<thead>
<tr>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NOT NULL</td>
</tr>
<tr>
<td>UEN_CODE</td>
<td>VARCHAR2(4)</td>
</tr>
<tr>
<td>UEN_NAME</td>
<td>VARCHAR2(25)</td>
</tr>
<tr>
<td>UEN_ADDRESS</td>
<td>VARCHAR2(200)</td>
</tr>
<tr>
<td>UEN_PHONE</td>
<td>VARCHAR2(25)</td>
</tr>
<tr>
<td>UEN_FAX</td>
<td>VARCHAR2(25)</td>
</tr>
<tr>
<td>UEN_EMAIL</td>
<td>VARCHAR2(45)</td>
</tr>
<tr>
<td>UEN_STATUS</td>
<td>VARCHAR2(12)</td>
</tr>
<tr>
<td>UEN_SITE</td>
<td>VARCHAR2(40)</td>
</tr>
</tbody>
</table>

### SQL> describe bk;

<table>
<thead>
<tr>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NOT NULL</td>
</tr>
<tr>
<td>ACC_NO</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>TITLE</td>
<td>VARCHAR2(150)</td>
</tr>
<tr>
<td>STMU_OR_RES</td>
<td>VARCHAR2(100)</td>
</tr>
<tr>
<td>MAIN_ENTRY</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>ADDED_ENTRY</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>EDITION</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>EDITOR</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>PUB_PLACE</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>PUBLISHER</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>PUB_DATE</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>CALL_NO</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>VUL_NO</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>PLACE</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>PAGES</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>SERIES</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>ISBN</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>CURR_CODE</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>ORDER_NO</td>
<td>NUMBER(15,2)</td>
</tr>
<tr>
<td>DISCOUNT</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>YEAR</td>
<td>NUMBER(5,2)</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>VARCHAR2(70)</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>VARCHAR2(100)</td>
</tr>
</tbody>
</table>

### SQL> describe bk_accounts

<table>
<thead>
<tr>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NOT NULL</td>
</tr>
<tr>
<td>INVOICE_NO</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>FUND_BOOKED_DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>PAY_REG_DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>DD_NO</td>
<td>VARCHAR2(15)</td>
</tr>
<tr>
<td>SQL&gt; describe bk_order</td>
<td>Null?</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>ORDER_NO</td>
<td>YES</td>
</tr>
<tr>
<td>ORDER_DATE</td>
<td>YES</td>
</tr>
<tr>
<td>VENDOR_CODE</td>
<td>YES</td>
</tr>
<tr>
<td>INVOICE_NO</td>
<td>YES</td>
</tr>
<tr>
<td>INVOICE_DATE</td>
<td>YES</td>
</tr>
<tr>
<td>INVOICE_AMOUNT</td>
<td>YES</td>
</tr>
<tr>
<td>USER_CODE</td>
<td>YES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SQL&gt; describe book_overdue</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U_CODE</td>
<td>YES</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>ACC_NO</td>
<td>YES</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>ISSUE_DATE</td>
<td>YES</td>
<td>DATE</td>
</tr>
<tr>
<td>DUE_DATE</td>
<td>YES</td>
<td>DATE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SQL&gt; describe user_entitle</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U_CATEGORY</td>
<td>YES</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>U_ENTITLE</td>
<td>YES</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>LOAN_PERIOD</td>
<td>YES</td>
<td>NUMBER(2)</td>
</tr>
<tr>
<td>U_FINE_DAY</td>
<td>YES</td>
<td>NUMBER(3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SQL&gt; describe reqbk</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNTRL_NO</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>TITLE</td>
<td></td>
<td>VARCHAR2(150)</td>
</tr>
<tr>
<td>F_AUTHOR</td>
<td></td>
<td>VARCHAR2(25)</td>
</tr>
<tr>
<td>S_AUTHOR</td>
<td></td>
<td>VARCHAR2(25)</td>
</tr>
<tr>
<td>EDITION</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>PUBLISHER</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>PUB_YEAR</td>
<td></td>
<td>NUMBER(4)</td>
</tr>
<tr>
<td>ISBN</td>
<td></td>
<td>VARCHAR2(13)</td>
</tr>
<tr>
<td>PRICE</td>
<td></td>
<td>NUMBER(15,2)</td>
</tr>
<tr>
<td>QUANTITY</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>VEN_CODE</td>
<td></td>
<td>VARCHAR2(3)</td>
</tr>
<tr>
<td>CURR_CODE</td>
<td></td>
<td>VARCHAR2(3)</td>
</tr>
<tr>
<td>DEP_CODE</td>
<td></td>
<td>VARCHAR2(20)</td>
</tr>
<tr>
<td>BUD_CODE</td>
<td></td>
<td>VARCHAR2(3)</td>
</tr>
<tr>
<td>USER_ID</td>
<td></td>
<td>VARCHAR2(10)</td>
</tr>
</tbody>
</table>
### SQL Queries for the Bibliographical Database

<table>
<thead>
<tr>
<th>Table</th>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_entitle</td>
<td>U_CATAGORY</td>
<td>NOT NULL</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td></td>
<td>U_ENTITLE</td>
<td>NOT NULL</td>
<td>NUMBER(2)</td>
</tr>
<tr>
<td></td>
<td>LOAN_PERIOD</td>
<td>NOT NULL</td>
<td>NUMBER(3)</td>
</tr>
<tr>
<td></td>
<td>U_FINE_DAY</td>
<td>NOT NULL</td>
<td>NUMBER(3)</td>
</tr>
<tr>
<td>user_issue</td>
<td>U_CODE</td>
<td>NOT NULL</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td></td>
<td>ACC_NO</td>
<td>NOT NULL</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td></td>
<td>ISSUE_DATE</td>
<td>NOT NULL</td>
<td>DATE</td>
</tr>
<tr>
<td></td>
<td>RETURN_DATE</td>
<td>NOT NULL</td>
<td>DATE</td>
</tr>
<tr>
<td>user_reserve</td>
<td>U_CODE</td>
<td>NOT NULL</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td></td>
<td>CALL_NO</td>
<td>NOT NULL</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td></td>
<td>RESERU_DATE</td>
<td>NOT NULL</td>
<td>DATE</td>
</tr>
<tr>
<td></td>
<td>CANCEL_DATE</td>
<td>NOT NULL</td>
<td>DATE</td>
</tr>
<tr>
<td></td>
<td>PRIORITY_NO</td>
<td>NOT NULL</td>
<td>NUMBER(2)</td>
</tr>
<tr>
<td>users</td>
<td>U_CODE</td>
<td>NOT NULL</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td></td>
<td>U_NAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td></td>
<td>U_ADDRESS</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td></td>
<td>U_EMAIL</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td></td>
<td>U_CATAGORY</td>
<td>NOT NULL</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td></td>
<td>REG_DATE</td>
<td>NOT NULL</td>
<td>DATE</td>
</tr>
<tr>
<td></td>
<td>EXP_DATE</td>
<td>NOT NULL</td>
<td>DATE</td>
</tr>
</tbody>
</table>

### 7.10 Queries execution on the Bibliographical Database

Data definition in SQL is via the CREATE statement. The statement is used to create a table, index, or view. To create table, the create statement specifies the name of the table and the names and data types of each column of the table. Its format is
Create table<relation> (<attribute list>)
Where the <attribute list> is specified as:
<attribute list> ::= <attribute name> (<data type>) [not null][, <attribute list>]

The data types supported by SQL depend on the particular implementation. However, the following data types are generally include: integer, decimal, real (i.e. floating point values) and character strings, both of fixed size and varying length.

An vendor relation for the library database is defined using the create table statement given below,

```sql
SQL> /
SP2-0103: Nothing in SQL buffer to run.
SQL> @uen
Table created.
SQL> 
```

```sql
create table uen
1  ( uen_code Uarchar2 (**) not null,
2  uen_name Uarchar2 (25),
3  uen_address Uarchar2 (200),
4  uen_phone Uarchar2 (25),
5  uen_fax Uarchar2 (25),
6  uen_email Uarchar2 (45),
7  uen_status Uarchar2 (12),
8  uen_site Uarchar2 (40))
```

```sql
SQL> describe uen;
Name ___________________________Type ___________________________
---------- ___________________________
VEN_CODE  VARCHAR2(4) NOT NULL
VEN_NAME  VARCHAR2(25)
VEN_ADDRESS VARCHAR2(200)
VEN_PHONE  VARCHAR2(25)
VEN_FAX  VARCHAR2(45)
VEN_EMAIL  VARCHAR2(12)
VEN_STATUS  VARCHAR2(40)
VEN_SITE
```

```sql
SQL>
```

The definition of an existing relation is modified using the alter statement.

```sql
Alter table (existing-table-name)
Add(column-name data-type)
```

The CREATE INDEX statement allows the creation of an index for an already existing relation.

```sql
Create [unique] index name-of-index
On existing-table-name
(column-name [aascending or descending]
[,column-name[order]...])
```

129
Create index venindex on Ven (Ven_code, ven_name, ven_status desc);

Through DML, the basic SELECT statement to retrieve data from the underlying table or tables. Various operators use in a select statement to perform various selected type of functions. It contains the UPDATE statement which uses to modify the data in a table. It also contains the Delete and Insert statements to delete a tuple or insert a new tuple.

```
Select [distinct], <target list>
From <relation list>
[where <predicate>]
```

The distinct option is used in the Select statement to eliminate duplicate tuples in the result. Without the distinct option duplicate tuples may appear in the result. The `<target list>` is a method of specifying a projection operation of the result relation. It takes the form: `<target list>::=<attribute name>[,<target list>]`.

The form clause specifies the relations to be used in the evaluation of the statement. It includes a relation list:

```
<relation list>::=<relation name>[<tuple variable>][,<relation list>]
```

Select Ven_Name
from Ven

The result of this select operation is a projection of the Ven relation on the attribute Ven_Name. Unlike the theoretical version of projection, this projection contains duplicate tuples. The reason for not eliminating these duplicates is the large amount of processing time required to do so. If the theoretical equivalent is desired, however, the distinct clause is `Select distinct Ven_Name
from Ven` added to the select statement.

Update statement: update statement is used to modify one or more records in a specified relation. The records to be modified are specified by a predicate in a where clause and the new value of the column(s) to be modified is specified by a set clause.
Update <relation> set <target-value-list>
[where.predicate.]
Where the target value list. Is of the form;
<target value list>::=<attribute name>=<value expression> [,<target value list>]
Update ven
Set ven_email = international@yahoo.com
Where v_name = 'International Book Supplier'

Delete Statement: The delete statement is used to delete one or more records from a
relation. The records to be deleted are specified by the predicate in the where clause.

Delete <relation>[where <predicate>
Delete ven
Where v_name = 'International Book Supplier'
If the where clause is left out all the tuples in the relation are deleted. In this case, the
relation is still known to the database although it is an empty relation. A relation along
with its tuples could be deleted by the drop statement.

Insert Statement: Insert statement to insert a new tuple into a specified relation is
used. The value of each field of the record to be inserted is either specified by an
expression or could come from select records of existing relations.

Insert into <relation>
Values (<values list>
Where the <value list> takes the form:
<value list>::=<value expression>[,<value list>]
Insert into ven
Values ('ACA', 'academia', 'Technology Market', '0322282447', '0322282431',
'aca@vsnl.net.in', 'A', 'http://www.aca.vsnl.org', 'ACA'),
Condition specification

To define a query language for BDMS whose models are described in the chapter 5.
The syntax of each construct, its typing constraint, and its semantic as the effect on a
BDMS. The following operations on records are used, and oid's: \[ \ell_1 = \nu_1, \ldots, \ell = \nu_n \] define a record with \( \nu_i \) for the value of each field \( \ell_i \), as seen in the examples. \( X, \ell \) selects the value contained in the field \( \ell \) of \( X \). If \( o \) denotes an oid, then \( \text{value}(o) \) returns the value associated with the oid. As a convention, I usually omit value if it is combined with a field selection operation. That is, \( o. \text{field} \) is an abbreviation for \( \text{value}(o) \text{field} \). I also assume the availability of standard primitive operations on atomic types such as integer arithmetic operations.

The constructs of the language falls into five categories: class creation, set operations, identity preserving transformation, oid operation, and reduction operation.

### 7.11 Class Creation.

The query language provides two ways to define classes of objects. The first is to define a set of mutually dependent classes by enumerating all the objects in each class using the following construct:

\[
\text{Class} \\
\text{User} = \{ \text{user} = [U\_name"G.Mukherjee"U\_code = '88032'.\ldots\ldots]\} \\
\text{and} \\
\text{Dept} = \text{lov} \{Aerospace = 'AE', Centrallibrary = 'CL'.\ldots\ldots\} \\
\text{End.} \\
\text{Class} \\
\text{User}_1 = \{o_{1,1} = O_{1,1}, \ldots, o_{1,n(1)}\} \\
\text{User}_2 = \{o_{2,1} = O_{2,1}, \ldots, o_{2,n(2)}\} \\
\text{User}_m = \{o_{m,1} = O_{m,1}, \ldots, o_{m,n(m)} = o_{m,n(m)}\} \\
\text{end} \\
\text{Name}_1 = \{o_{1,1} = O_{1,1}, \ldots, o_{1,n(1)}\} \\
\text{end}
\]
where each field name is a program variable in the language bound to a class of objects, \( O_{i,j} \)'s are object value \( O_{ij} \)'s and are oid descriptors for the corresponding objects.

The oid descriptors can appear in any object \( O_{i,j} \)'s to refer to the corresponding oid, allowing mutually dependent object definitions.

![Welcome to LIPS - Microsoft Internet Explorer](https://image.wikiwand.com/en/...)

**Figure 7.9 - Class Creation schema.**

The declaration works as an operation that defines \( m \) new classes and registers them in the database. It extends the schema \( S \) with the set of type equations \( \{t_1 = \tau_1, \ldots, t_m = \tau_m\} \) provided that the following typing relation holds \( O_{i,j} : \tau_i \) for any \( i,j \) under the assumption that \( o_{k,l} = t_k \) hold for all \( k,l \), where each \( t_k \) is a fresh type variable introduced for each \( Name_k \). In accordance with the extension of the schema, the schema instance is extended as follows: First, for every \( i (1 \leq i \leq m) \), a set of new oid's \( \{o_{ij} \} \) is created and the schema instance \( o \) is expanded with the set of oid equations \( \{o_{ij} = o_{ij} \mid 1 \leq i \leq m, 1 \leq j \leq n(i)\} \) where each \( o_{ij} \) is the value denoted by \( o_{i,j} \). Next, program variable \( Name_k \) is bound to the set of oid's \( \{o_{ij} \} \).

The source code for procedure of Acquisition schema, Acquisition report, WPAC search and the source codes for some relation in PL/SQL are include in appendix-1 and appendix-2 respectively at the end of chapter 9.
7.12 Reference.


4. BERNSTEIN, PHILIP A., IOANNIDIS, YANNIS E., RAMAKRISHNAN, RAGHU., PAPADIAS, DIMITRIS. The VLDB Endowment. VLDB '02: Proceedings of the 28th International Conference on Very Large Databases, 2002, pp. xii


17. HOBBs, LILIAN., HILLSON, SUSAN., LAWANDE, SHILPA., and SMITH, PETE., Architecture of a Data Warehouse. In *Oracle 10g Data Warehousing, 2005, pp. 73-111.*


21. KATSIRIKOU, ANTHI., And SEFERTZI, ELENA. Innovation in the every day life of libraries. In Technovation, Volume 20, Issue 12, December 2000, pp.705-709


43. YARGER, RANDY JAY. REESE, GEORGE and KING, TIM. *mysql and mSQL*. O'Relly Inc. 1999.