Jawaharlal Nehru Engineering College

Laboratory Manual

DATABASE MANAGEMENT SYSTEM

For

Third Year Students CSE
Dept: Computer Science & Engineering (NBA Accredited)

Author JNEC, Aurangabad
FOREWORD

It is my great pleasure to present this laboratory manual for THIRD YEAR COMPUTER SCIENCE engineering students for the subject of Database Management System keeping in view the vast coverage required for cryptography and encryption.

As a student, many of you may be wondering with some of the questions in your mind regarding the subject and exactly what has been tried is to answer through this manual.

As you may be aware that MGM has already been awarded with ISO 9001:2000 certification and it is our endure to technically equip our students taking the advantage of the procedural aspects of ISO 9001:2000 Certification.

Faculty members are also advised that covering these aspects in initial stage itself, will greatly relived them in future as much of the load will be taken care by the enthusiasm energies of the students once they are conceptually clear.

Dr. S.D.Deshmukh

Principal
LABORATORY MANUAL CONTENTS

This manual is intended for THIRD YEAR COMPUTER SECINCE students for the subject of **Database Management System**. This manual typically contains practical/Lab Sessions related Database Management System covering various aspects related the subject to enhanced understanding.

Students are advised to thoroughly go though this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions

Prof. D.S.Deshpande  
HOD, CSE

Ms. S.B.Jaiwal  
Lecturer, CSE Dept.
**DOs and DON'Ts in Laboratory:**

1. Make entry in the Log Book as soon as you enter the Laboratory.

2. All the students should sit according to their roll numbers starting from their left to right.

3. All the students are supposed to enter the terminal number in the log book.

4. Do not change the terminal on which you are working.

5. All the students are expected to get at least the algorithm of the program/concept to be implemented.

6. Strictly observe the instructions given by the teacher/Lab Instructor.

**Instruction for Laboratory Teachers:**

1. Submission related to whatever lab work has been completed should be done during the next lab session. The immediate arrangements for printouts related to submission on the day of practical assignments.

2. Students should be taught for taking the printouts under the observation of lab teacher.

3. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.
**SUBJECT INDEX**

1. Installing oracle 9i on windows xp.

2. Creating Entity-Relationship Diagram using case tools.

3. Writing SQL statements Using ORACLS /MYSQL:
   1. Writing basic SQL SELECT statements.
   2. Restricting and sorting data.
   3. Displaying data from multiple tables.
   4. Aggregating data using group function.
   5. Manipulating data.
   6. Creating and managing tables.

4. Normalization in ORACLE.

5. Creating cursor in oracle.

6. Creating procedure and functions in oracle.

7. Creating packages and triggers in oracle.

8. TEAM Project
1. Lab Exercise

Aim: Installation of Oracle 9i on WIN xp (Introduction to Database.)

S/w Requirement: CD of the setup Oracle 9i, 10g.

Theory:

Database is a collection of information in a structured way. We can say that it is a collection of a group of facts. Your personal address book is a database of names you like to keep track of, such as personal friends and members of your family. Fig.1 contains required details about each student. There are six pieces of information on each student. They are Roll No, Name, Address and Subjects. Each piece of information in database is called a Field. We can define field as the smallest unit in a database. Each field represents one and only one characteristic of an event or item. Thus there are four fields in this database.

<table>
<thead>
<tr>
<th>ROLL NO</th>
<th>NAME</th>
<th>ADDRESS</th>
<th>SUBJECTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>200207</td>
<td>Abc</td>
<td>n-6,Aurangabad</td>
<td>DBMS</td>
</tr>
<tr>
<td>200207</td>
<td>Pqr</td>
<td>n-5,Aurangabad</td>
<td>.NET</td>
</tr>
<tr>
<td>200207</td>
<td>Xyz</td>
<td>n-2,Aurangabad</td>
<td>C</td>
</tr>
<tr>
<td>200207</td>
<td>Stq</td>
<td>n-1,Aurangabad</td>
<td>VB.NET</td>
</tr>
<tr>
<td>200207</td>
<td>Plm</td>
<td>n-7,Aurangabad</td>
<td>CG</td>
</tr>
</tbody>
</table>

Fig. 1

If you take a close look at all these fields, they are not of the same
type. Name is character type Roll no is number type. In database there can be
five categories of fields. They are:

- Numeric
- Character
- Logic
- Memo
- Date

**Installation of oracle 9i:**

After inputting the cd we get this screen:

![Image of Oracle 9i Installation Screen](image)

Then click on INSTALL/DEINSTALL
Welcome

The Oracle Universal Installer guides you through the installation and configuration of your Oracle products.

Click "Installed Products..." to see all installed products.

Press Next
Confirm/modify the Destination Name and Path and press Next
Available Products

Select a product to install.

- **Oracle9i Database 9.2.0.1.0**
  - Installs an optional pre-configured starter database, product options, management tools, networking services, utilities and basic client software for an Oracle database server.

- **Oracle9i Management and Integration 9.2.0.1.0**
  - Installs the management server, management tools, Oracle Internet Directory, Oracle Integration Server, networking services, utilities and basic client software.

- **Oracle9i Client 9.2.0.1.0**
  - Installs enterprise management tools, networking services, utilities, development tools and precompilers and basic client software.

Select the Oracle 9i Database
Select Standard Edition and press Next

Alternatively, you could select Custom and deselect all Enterprise/Management/Web Server components.
Select General Purpose and press Next.
Oracle Services for Microsoft Transaction Server

Oracle MTS Recovery Service Configuration

The Oracle MTS Recovery Service is automatically installed with Oracle Services for Microsoft Transaction Server. The Oracle MTS Recovery Service accepts requests to resolve in-doubt MS DTC-coordinated transactions started on this computer. Enter the port number on which the Oracle MTS Recovery Service will listen for requests on this computer.

Port Number: 2030

Windows platforms only: Just press Next
Enter your Global Database Name (also called TNS Name). *You need to know this name for later steps!*

In this example, "compiere" is the database name or SID and "compiere.org" the company name

**Press Next.**
Confirm directory and press Next
In most cases, you could use the default character set, but we recommend using Unicode; press Next.

Note: The character set AL16UTF16 is more efficient than the default AL32UTF8. AL16UTF16 is also referred to as UTF16 or UCS2 and is a fixed 16 bit length code. AL32UTF8 is a variable length unicode implementation and a bit less efficient as with every access, the length/width needs to be calculated. So, you have to choose between speed (AL16UTF16) and space (AL32UTF8).
Check and press Install
You will see this for a while ... and then ...
Configuration Tools

The following tools will be automatically started for you:

These tools are optional. It is recommended, although not required, that these tools be run successfully.

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Net Configuration Assistant</td>
<td>succeeded</td>
</tr>
<tr>
<td>Oracle Database Configuration Assistant</td>
<td>in progress</td>
</tr>
<tr>
<td>Agent Configuration Assistant</td>
<td>pending</td>
</tr>
<tr>
<td>SH Agent Configuration Assistant</td>
<td></td>
</tr>
</tbody>
</table>

... at the end of the database install ...
Enter and remember your SYS and SYSTEM password and press OK.
End of Installation

The installation of Oracle9i Database was successful.

Please remember...

The following information is available in:
D:\oracle\ora92\Apache\Apache\setupinfo.txt

The HTTP Server can be accessed using the following URLs:

http://MAIN:7778
https://MAIN:4443

Compiler does not use the Oracle HTTP Server. You can disable the start in your operation system. Press Exit.

Done!!! Press Yes


2. **Lab Exercise**

**Aim:** Study of Entity-Relationship diagram.

**S/w Requirement:** Oracle 9i, 10g.

**Theory:**

An entity-relationship (ER) diagram is a specialized graphic that illustrates the interrelationships between entities in a database. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities. Diamonds are normally used to represent relationships and ovals are used to represent attributes.

1. **Entity**
   
   An *entity* is a person, place, concept, or thing about which the business needs data.

2. **Relationship**

   A *relationship* is an association between entity types.

**Example of ER Diagram**

we have identified three entity types (*Employee, Department, Division*) and two relationships among these entity types (*manages, contains*). Now we can begin to represent the problem in the language of ER modeling.

ER models are usually represented graphically. The language we are going to use represents entity types as rectangles and relationships as diamonds. Below is the representation of the situation we are working with.

```
Division  contains  Dept  manages  Employee
```

Notice that the contains relationship is drawn between the two entities that it is associated with. Similarly for the manages relationship. This (simplified) ER model tells us that:

- Division is related to department through a relationship called *contains*.
- Departments are related to employees through a relationship called *manages*.
- Employees are not directly related to divisions.

Certainly, we know more about the problem than this. Consider the relationship between divisions and departments. We know that divisions have multiple departments and departments can only be contained within one division. Or, for every *one* division there can be many departments. In the
language of ER modeling this is called a \textbf{1: M} (read: “one to many”) relationship.

3. Cardinality: There are 4 types of cardinality:

A. One to One  
B. One to Many  
C. Many to Many  
D. Many to One

\textbf{A. One-to-one}

One-to-one table relationships are a little more interesting and more underused than either of the other two types of relationships. The key indicator of a possible need for a one-to-one relationship is a table that contains fields that are only used for a certain subset of the records in that table.

Let’s take a look at building a Catalog table for the items that your store sells. Odds are that you need to store some information about the individual items like catalog numbers, weight, and other common data. But if you’re selling different kinds of items, books and CDs for example, you may want some item-specific information in the database. For example, you may want a page count, author, publish date, and ISBN for books, while you want playing time, number of tracks, artist, and label for the CDs. You could come up with some way to fit both sets of data into the same structure, but then when management decides you’re also selling pet supplies, your system will probably break!

A better solution would be a one-to-one relationship between the Item table and another table of item-specific data for each type of item. The resulting structure is essentially one "master" table (CatalogItems) with one or more "subtables" (CDs and Books in this example). You link the two subtables to the master table through the primary key of the master table.

\begin{verbatim}
Catalog Table

<table>
<thead>
<tr>
<th>CatalogID</th>
<th>Price</th>
<th>Description</th>
<th>QuantityOnHand</th>
</tr>
</thead>
</table>

CDs

<table>
<thead>
<tr>
<th>CatalogID</th>
<th>PlayingTime</th>
<th>NumOfTracks</th>
<th>Artist</th>
<th>Label</th>
</tr>
</thead>
</table>

Books

<table>
<thead>
<tr>
<th>CatalogID</th>
<th>PageCount</th>
<th>Author</th>
<th>PublishDate</th>
<th>ISBN</th>
</tr>
</thead>
</table>
\end{verbatim}
It may take a few minutes for this design to sink in. As a comparison, here is what the proposed database table would look like as a single monolithic table.

The one-to-one relationship has saved us from doubling the number of fields in the Catalog table and, more importantly, helped us break the database into more discrete entities. In this scenario, we can get all the general information about an item from the Catalog table and can use the primary key of that table to pull up the appropriate information from the sub table.

B. One-to-many

The one-to-many relationship is the workhorse of relational databases as well as being the easiest relationship to understand. Let us say you need to build a shopping cart application for an e-commerce site. Your first draft of the database has columns for Item1, Item2, and Item3 with the corresponding Quantity1, Quantity2, and Quantity3 fields.

<table>
<thead>
<tr>
<th>OrderNum</th>
<th>ShippingInfo</th>
<th>Item1</th>
<th>Quantity1</th>
<th>Item2</th>
<th>Quantity2</th>
<th>Item3</th>
<th>Quantity3</th>
</tr>
</thead>
</table>

Of course, this immediately starts to break down with more than three orders! Any time you find yourself designing a database and adding similar fields like this to the same table, you need to break the table into two (or more!) related tables using a one-to-many relationship.

A one-to-many relationship allows records in Table 1 to be connected to an arbitrary number of records in Table 2 without the limitations imposed by resorting to redundant or limited numbers of fields in a single table. This reduces the size of the database and greatly increases the flexibility and performance of queries operating on that data. We can take our shopping cart example and break it into an Order table and an Item table quite simply.

Order Table

<table>
<thead>
<tr>
<th>OrderID</th>
<th>ShippingInfo</th>
</tr>
</thead>
</table>

OrderItem Table

<table>
<thead>
<tr>
<th>OrderItemID</th>
<th>OrderID</th>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
</table>

The two tables are linked together using the OrderID field. The contents of any order in the Order table can easily be found by finding all the items with that value in the OrderID field. There is also the added advantage that the two pieces of data are independent and can easily be modified. If we now want to add an ItemNumber to the OrderItem table, we add a single column; in our
original monolithic data table, we'd be adding ItemNumber1, ItemNumber2, etc.

C. Many-to-many

Finally, there is the many-to-many table. This relationship is a little more complex than the one-to-many because, in addition to the two tables of data, we need another table to join the two tables of interest together. That's right, we're adding a table to the database -- but it is a simple table and saves us lots of effort down the road. As an example, let's say you want to add the ability to search for CDs by the musicians on any given song. From the musician side, you have one musician related to many songs.

Musician Table

<table>
<thead>
<tr>
<th>MusicianID</th>
<th>MusicianName</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>Paul McCartney</td>
</tr>
</tbody>
</table>

Song Table

<table>
<thead>
<tr>
<th>SongID</th>
<th>MusicianID</th>
<th>SongName</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>44</td>
<td>Sgt. Pepper's Lonely Heart's Club Band</td>
</tr>
<tr>
<td>201</td>
<td>44</td>
<td>Ebony and Ivory</td>
</tr>
</tbody>
</table>

But from the song side, you potentially have a song related to many musicians. The following visual represents that situation.

Song Table

<table>
<thead>
<tr>
<th>SongID</th>
<th>SongName</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Sgt. Pepper's Lonely Heart's Club Band</td>
</tr>
</tbody>
</table>

Musician Table

<table>
<thead>
<tr>
<th>MusicianID</th>
<th>SongID</th>
<th>MusicianName</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>200</td>
<td>John Lennon</td>
</tr>
<tr>
<td>44</td>
<td>200</td>
<td>Paul McCartney</td>
</tr>
</tbody>
</table>

These two tables work individually, but when you try to put them together, you end up with this mish-mash table.
This has saved us nothing -- in fact, it has complicated the structure by introducing lots of redundant data to manage. The way to handle this situation is to create two one-to-many relationships involving a linking table which we'll call Song_Musician, since it links those tables. We create a one-to-many from Song to Song_Musician since one song will have 0-N musicians and then another one-to-many from Musician to Song_Musician since any one musician will be in one or more songs. The results look like the following:

**Musician Table**

<table>
<thead>
<tr>
<th>MusicianID</th>
<th>MusicianName</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>John Lennon</td>
</tr>
<tr>
<td>44</td>
<td>Paul McCartney</td>
</tr>
</tbody>
</table>

**Song_Musician Table**

<table>
<thead>
<tr>
<th>SongID</th>
<th>MusicianID</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>43</td>
</tr>
<tr>
<td>200</td>
<td>44</td>
</tr>
<tr>
<td>201</td>
<td>44</td>
</tr>
</tbody>
</table>

**Song Table**

<table>
<thead>
<tr>
<th>SongID</th>
<th>SongName</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Sgt. Pepper's Lonely Heart's Club Band</td>
</tr>
<tr>
<td>201</td>
<td>Ebony and Ivory</td>
</tr>
</tbody>
</table>
This time around, all of the redundant data is in the Song_Musician table, which are only two columns of integers. Any changes to the structure of the Song or Musician table remain independent of their relationship, which is precisely what we’re after.

**D. Many to One**

An employee can work in only one department; this relationship is single-valued for employees. On the other hand, one department can have many employees; this relationship is multi-valued for departments. The relationship between employees (single-valued) and departments (multi-valued) is a one-to-many relationship.

To define tables for each one-to-many and each many-to-one relationship:

1. Group all the relationships for which the "many" side of the relationship is the same entity.
2. Define a single table for all the relationships in the group.

In the following example, the "many" side of the first and second relationships is "employees" so an EMPLOYEE table is defined.

<table>
<thead>
<tr>
<th>Table 3. Many-to-One Relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entity</strong></td>
</tr>
<tr>
<td>Employees</td>
</tr>
<tr>
<td>Employees</td>
</tr>
<tr>
<td>Departments</td>
</tr>
</tbody>
</table>

In the third relationship, "departments" is on the "many" side, so a department table, DEPARTMENT, is defined.

The following tables show these different relationships.

The EMPLOYEE table:

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>WORKDEPT</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>000010</td>
<td>A00</td>
<td>President</td>
</tr>
<tr>
<td>000020</td>
<td>B01</td>
<td>Manager</td>
</tr>
</tbody>
</table>
### EMPLOYEE Table:

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>WORKDEPT</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>000120</td>
<td>A00</td>
<td>Clerk</td>
</tr>
<tr>
<td>000130</td>
<td>C01</td>
<td>Analyst</td>
</tr>
<tr>
<td>000030</td>
<td>C01</td>
<td>Manager</td>
</tr>
<tr>
<td>000140</td>
<td>C01</td>
<td>Analyst</td>
</tr>
<tr>
<td>000170</td>
<td>D11</td>
<td>Designer</td>
</tr>
</tbody>
</table>

### DEPARTMENT Table:

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>ADMRDEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>C01</td>
<td>A00</td>
</tr>
<tr>
<td>D01</td>
<td>A00</td>
</tr>
<tr>
<td>D11</td>
<td>D01</td>
</tr>
</tbody>
</table>
3. **Lab Exercise**

**Aim:** Writing SQL statements Using ORACLS / MYSQL

**S/w Requirement:** Oracle 9i, 10g.

**Theory:**

1. **DDL statements [Data Definition Language].**

Statements are used to define the database structure or schema. Some examples:

A. **CREATE** - to create objects in the database

B. **ALTER** - alters the structure of the database

C. **DROP** - delete objects from the database

D. **TRUNCATE** - remove all records from a table, including all spaces allocated for the records are removed

**SYNTAX:**

A. `create table <table_name> (column definition 1, column definition 2,...);`

B. `alter table <table_name> modify (column definition....);`

C. `alter table <table_name> add (column definition....);`

D. `drop table <table_name>;`

E. `truncate table <table_name>;`

2. **DML Statements [Data Manipulation Language]**

Statements are used for managing data within schema objects. Some examples:

A. **SELECT** - retrieve data from the a database

B. **INSERT** - insert data into a table

C. **UPDATE** - updates existing data within a table

D. **DELETE** - deletes all records from a table, the space for the records remain

**SYNTAX:**

A. `select <column_name> from <table_name>;`
B. insert into <table_name> values(alist of data values);
C. update <table_name> set <column_name>;
D. delete from <table_name> where <condition>;

3. **DCL Statements [Data Control Language]**

A. GRANT - gives user's access privileges to database
B. REVOKE - withdraw access privileges given with the GRANT command

**SYNTAX:**

A. grant privileges on <object_name> to <user_name>;
B. revoke privileges on <object_name> to <user_name>;

4. **Transaction Control [TCL]**

Transaction Control (TCL) statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

COMMIT - save work done
ROLLBACK - restore database to original since the last COMMIT

**SYNTAX:**

A. commit;
B. rollback;

5. **Aggregating data using group function:**

A group function returns a result based on group of rows. Some are purely mathematical function.

A. Avg function:

It returns average of values of the column specified in the arguments in the column.
**SYNTAX:**

Select `avg(column_name whose avg to find)` from `<table_name>` where condition;

B.min function:

This function will give the least value of the column present in the argument.

Select `min(column_name whose min to find)` from `<table_name>` where condition;

C.max function:

This function will give the maximum value of the column present in the argument.

Select `max(column_name whose max to find)` from `<table_name>` where condition;

D.sum:

This function will give the sum of value of the column present in the argument.

Select `sum(column_name whose sum to find)` from `<table_name>` where condition;

E.count:

This function is used to count the number of rows in function.

Select `count (*)` from `<table_name>` ;
4. Lab Exercise

Aim: Normalization in ORACLE.

S/w Requirement: Oracle 9i, 10g.

Theory:

A. Trivial functional dependency
   A trivial functional dependency is a functional dependency of an attribute on a superset of itself. \{Employee ID, Employee Address\} → \{Employee Address\} is trivial, as is \{Employee Address\} → \{Employee Address\}.

B. Full functional dependency
   An attribute is fully functionally dependent on a set of attributes X if it is
   - Functionally dependent on X, and
   - not functionally dependent on any proper subset of X. \{Employee Address\} has a functional dependency on \{Employee ID, Skill\}, but not a full functional dependency, because it is also dependent on \{Employee ID\}.

C. Transitive dependency
   A transitive dependency is an indirect functional dependency, one in which X→Z only by virtue of X→Y and Y→Z.

D. Multivalued dependency
   A multivalued dependency is a constraint according to which the presence of certain rows in a table implies the presence of certain other rows.

E. Join dependency
   A table T is subject to a join dependency if T can always be recreated by joining multiple tables each having a subset of the attributes of T.

F. Superkey
   A superkey is an attribute or set of attributes that uniquely identifies
rows within a table; in other words, two distinct rows are always
guaranteed to have distinct superkeys. \{Employee ID, Employee Address,
Skill\} would be a superkey for the "Employees' Skills" table; \{Employee
ID, Skill\} would also be a superkey.

G. Candidate key

A candidate key is a minimal superkey, that is, a superkey for which we
can say that no proper subset of it is also a superkey. \{Employee Id, Skill\}
would be a candidate key for the "Employees' Skills" table.

H. Non-prime attribute

A non-prime attribute is an attribute that does not occur in any
candidate key. Employee Address would be a non-prime attribute in the
"Employees' Skills" table.

F. Primary key

Most DBMSs require a table to be defined as having a single unique key,
rather than a number of possible unique keys. A primary key is a key
which the database designer has designated for this purpose.

<table>
<thead>
<tr>
<th>Normal form</th>
<th>Defined by</th>
<th>Brief definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>First normal form (1NF)</td>
<td>Two versions: E.F. Codd (1970), C.J. Date (2003)</td>
<td>Table faithfully represents a relation and has no &quot;repeating groups&quot;</td>
</tr>
<tr>
<td>Second normal form (2NF)</td>
<td>E.F. Codd (1971)</td>
<td>No non-prime attribute in the table is functionally dependent on a part (proper subset) of a candidate key</td>
</tr>
<tr>
<td>Third normal form (3NF)</td>
<td>E.F. Codd (1971); see also Carlo Zaniolo's equivalent</td>
<td>Every non-prime attribute is non-transitively dependent on every key</td>
</tr>
</tbody>
</table>
Now we ready to come to grips with the ideas of normalization. The following table, containing information about some students at Enormous State University, is a table that is in 1st Normal Form, 1NF.

You can easily verify for yourself that this table satisfies the definition of 1NF: viz., it has no duplicated rows; each cell is single-valued (i.e., there are no repeating groups or arrays); and all the entries in a given column are of the same kind.
In Table 1 we can see that the key, SSN, functionally determines the other attributes; i.e., a given Social Security Number implies (determines) a particular value for each of the attributes FirstName, LastName, and Major (assuming, at least for the moment, that a student is allowed to have only one major). In the arrow notation: SSN → FirstName, SSN → LastName, and SSN → Major.

A key attribute will, by the definition of key, uniquely determine the values of the other attributes in a table; i.e., all non-key attributes in a table will be functionally dependent on the key. However, there may be non-key attributes in a table that determine other attributes in that table. Consider the following table:

<table>
<thead>
<tr>
<th>SocialSecurity Number</th>
<th>FirstName</th>
<th>LastName</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>Jack</td>
<td>Jones</td>
<td>Library and Information Science</td>
</tr>
<tr>
<td>222-33-4444</td>
<td>Lynn</td>
<td>Lee</td>
<td>Library and Information Science</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>Mary</td>
<td>Ruiz</td>
<td>Pre-Medicine</td>
</tr>
<tr>
<td>123-54-3210</td>
<td>Lynn</td>
<td>Smith</td>
<td>Pre-Law</td>
</tr>
<tr>
<td>111-33-5555</td>
<td>Jane</td>
<td>Jones</td>
<td>Library and Information Science</td>
</tr>
</tbody>
</table>
Table 2

In Table 2 the Level attribute can be said to be functionally dependent on the Major attribute. Thus, we have an example of an attribute that is functionally dependent on a non-key attribute. This statement is true in the table *per se*, and that is all that the definition of functional dependence requires; but the statement also reflects the real-world fact that Library and Information Science is a major that is open only to graduate students and that Pre-Medicine and Pre-Law are majors that are open only to undergraduate students.

The 2nd Normal Form (2NF)

Table 2 has another interesting aspect. Its key is a composite key, consisting of the paired attributes, FirstName and LastName. The Level attribute is functionally dependent on this composite key, of course; but, in addition, Level can be seen to be dependent on only the attribute LastName. (This is true because each value of Level is paired with a distinct value of LastName. In contrast, there are two occurrences of the value Lynn for the attribute FirstName, and the two Lynns are paired with different values of Level, so Level is not functionally dependent on FirstName.) Thus this table fails to qualify as a 2nd Normal Form table, since the definition of 2NF requires that all non-key attributes be dependent on the entire key. (Admittedly, this example of a partial dependency is artificially contrived, but nevertheless it illustrates the problem of partial dependency.)

We can turn Table 2 into a table in 2NF in an easy way, by adding a column for the Social Security Number, which will then be the natural thing to use as the key.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FirstName</strong></td>
<td><strong>LastName</strong></td>
<td><strong>Major</strong></td>
<td><strong>Level</strong></td>
</tr>
<tr>
<td>Jack</td>
<td>Jones</td>
<td>LIS</td>
<td>Graduate</td>
</tr>
<tr>
<td>Lynn</td>
<td>Lee</td>
<td>LIS</td>
<td>Graduate</td>
</tr>
<tr>
<td>Mary</td>
<td>Ruiz</td>
<td>Pre-Medicine</td>
<td>Undergraduate</td>
</tr>
<tr>
<td>Lynn</td>
<td>Smith</td>
<td>Pre-Law</td>
<td>Undergraduate</td>
</tr>
<tr>
<td>Jane</td>
<td>Jones</td>
<td>LIS</td>
<td>Graduate</td>
</tr>
</tbody>
</table>
Table 3

With the SSN defined as the key, Table 3 is in 2NF, as you can easily verify. This illustrates the fact that any table that is in 1NF and has a single-attribute (i.e., a non-composite) key is automatically also in 2NF.

Table 3 still exhibits some problems, however. For example, it contains some repeated information about the LIS-Graduate pairing.

Anomalies and Normalization

At this point it is appropriate to note that the main thrust behind the idea of normalizing databases is the avoidance of insertion and deletion anomalies in databases.

To illustrate the idea of anomalies, consider what would happen to our knowledge (at least, as explicitly contained in a table) of the level of the major, Pre-Medicine, if Mary Ruiz left Enormous State University. With the deletion of the row for Ms. Ruiz, we would lose the information that Pre-Medicine is an Undergraduate major. This is an example of a deletion anomaly. We may possess the real-world information that Pre-Medicine is an Undergraduate major, but no such information is explicitly contained in a table in our database.

As an example of an insertion anomaly, we can suppose that a new student wants to enroll in ESU: e.g., suppose Jane Doe wants to major in Public Affairs. From the information in Table 3 we cannot tell whether Public Affairs is
an Undergraduate or a Graduate major; in fact, we do not even know whether Public Affairs is an established major at ESU. We do not know whether it is permissible to insert the value, Public Affairs, as a value of the attribute, Major, or what to insert for the attribute, Level, if we were to assume that Public Affairs is a valid value for Major. The point is that while we may possess real-world information about whether Public Affairs is a major at ESU and what its level is, this information is not explicitly contained in any table that we have thus far mentioned as part of our database.

A database-management system, a DBMS, can work only with the information that we put explicitly into its tables for a given database and into its rules for working with those tables, where such rules are appropriate and possible.

How do anomalies relate to normalization? The simple answer is that by arranging that the tables in a database are sufficiently normalized (in practice, this typically means to at least the 4th level of normalization), we can ensure that anomalies will not arise in our database. Anomalies are difficult to avoid directly, because with databases of typical complexity (i.e., several tables) the database designer can easily overlook possible problems. Normalization offers a rigorous way of avoiding unrecognized anomalies.

Normalization may look like a difficult process when one views it from the standpoint of the formal definitions of the various normal forms, as presented in Section 2 of this handout. But in practice, you can easily attain sufficient normalization in your database by simply ensuring that the tables in your database are what we can call "single-theme" tables. This idea will be illustrated as we proceed through the rest of the discussion in this handout.

Turning a Table with Anomalies (Table 3) into Single-Theme Tables

Although Table 3 is in 2NF, it is still open to the problems of insertion and deletion anomalies, as the discussion in the preceding section shows. The reason is that Table 3 deals with more than a single theme. What can we do to turn it into a set of tables that are, or at least come closer to being, single-theme tables?

A reasonable way to proceed is to note that Table 3 deals with both information about students (their names and SSNs) and information about majors and levels. This should strike you as two different themes. Presented below is one possible set of single-theme tables dealing with the information in Table 3. (To save space, the following tables also contain some information that is not in Table 3, and the discussion will deal with this added information.)

<table>
<thead>
<tr>
<th>SSN</th>
<th>FirstName</th>
<th>LastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>Jack</td>
<td>Jones</td>
</tr>
<tr>
<td>SSN</td>
<td>Major</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>123-45-6789</td>
<td>LIS</td>
<td></td>
</tr>
<tr>
<td>222-33-4444</td>
<td>LIS</td>
<td></td>
</tr>
</tbody>
</table>

Table 5
The three preceding tables should strike you as providing a better arrangement of the information in Table 3. For one thing, this arrangement puts the information about the students into a smaller table, Table 4, which happily fails to contain redundant information about the LIS-Graduate pairing. For another thing, this arrangement permits us to enter information about students (e.g., Newton Gingpoor) who have not yet identified themselves as pursuing a particular major. For still another thing, it puts the information about the Major-Level pairings into a separate table, Table 5, which can easily be expanded to include information (e.g., that the Public Affairs major is at the Graduate level) about majors for which, at the moment, there may be no students registered. Finally, Table 6 provides the needed link between individual students and their majors (note that Newton Gingpoor's SSN is not in this Table 6, which tells us that he has not yet selected a major).

Tables 4 - 6 are single-theme tables and are in 2NF, as you can easily verify. (In fact, they are in DKNF, but we are not yet ready to discuss the latter level in detail.)

The 3rd Normal Form (3NF)

In order to discuss the 3rd Normal Form, we need to begin by discussing the idea of transitive dependencies.

In mathematics and logic, a transitive relationship is a relationship of the following form: "If A implies B, and if also B implies C, then A implies C." An example is: "If John Doe is a human, and if every human is a primate, then John Doe must be a primate." Another way of putting it is this: "If A functionally governs B, and if B functionally governs C, then A functionally governs C." In the arrow notation, we have:

\[(A \implies B) \land (B \implies C) \implies (A \implies C)\]

The following table, Table 7, provides an example of how transitive dependencies can occur in a table in a relational database.
By examining Table 7 we can infer that books dealing with history, cognitive psychology, and folksong are assigned to the PCL General Stacks collection; that books dealing with legal procedures are assigned to the Law Library; that books dealing with Greek literature are assigned to the Classics Library; that books dealing with library biography are assigned to the Library and Information Science Collection (LISC); and that books dealing with music literature are assigned to the Fine Arts Library.

Further, we can infer that the PCL General Stacks collection and the LISC are both housed in the Perry-Castañeda Library (PCL) building; that the Classics Library is housed in Waggener Hall; and that the Law Library and Fine Arts Library are housed, respectively, in Townes Hall and the Fine Arts Building.

Thus we see that there is a transitive dependency in Table 7: any book that deals with history, cognitive psychology, or library biography will be physically housed in the PCL building (unless it is temporarily checked out to a borrower); any book dealing with legal procedures will be housed in Townes Hall; and so on. In short, if we know what subject a book deals with, we also know not only what library or collection it will be assigned to but also what building it is physically housed in.

Table 7

<table>
<thead>
<tr>
<th>Name</th>
<th>Name</th>
<th>Subject</th>
<th>Library</th>
<th>Building</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berdahl</td>
<td>Robert</td>
<td>The Politics of the Prussian Nobility</td>
<td>History</td>
<td>PCL General Stacks</td>
</tr>
<tr>
<td>Yudof</td>
<td>Mark</td>
<td>Child Abuse and Neglect</td>
<td>Legal Procedures</td>
<td>Law Library</td>
</tr>
<tr>
<td>Harmon</td>
<td>Glynn</td>
<td>Human Memory and Knowledge</td>
<td>Cognitive Psychology</td>
<td>PCL General Stacks</td>
</tr>
<tr>
<td>Graves</td>
<td>Robert</td>
<td>The Golden Fleece</td>
<td>Greek Literature</td>
<td>Classics Library</td>
</tr>
<tr>
<td>Miksa</td>
<td>Francis</td>
<td>Charles Ammi Cutter</td>
<td>Library Biography</td>
<td>Library and Information Science Collection</td>
</tr>
<tr>
<td>Hunter</td>
<td>David</td>
<td>Music Publishing and Collecting</td>
<td>Music Literature</td>
<td>Fine Arts Library</td>
</tr>
<tr>
<td>Graves</td>
<td>Robert</td>
<td>English and Scottish Ballads</td>
<td>Folksong</td>
<td>PCL General Stacks</td>
</tr>
</tbody>
</table>

41
What is wrong with having a transitive dependency or dependencies in a table? For one thing, there is duplicated information: from three different rows we can see that the PCL General Stacks are in the PCL building. For another thing, we have possible deletion anomalies: if the Yudof book were lost and its row removed from Table 7, we would lose the information that books on legal procedures are assigned to the Law Library and also the information the Law Library is in Townes Hall. As a third problem, we have possible insertion anomalies: if we wanted to add a chemistry book to the table, we would find that Table 7 nowhere contains the fact that the Chemistry Library is in Robert A.Welch Hall. As a fourth problem, we have the chance of making errors in updating: a careless data-entry clerk might add a book to the LISC but mistakenly enter Townes Hall in the building column.

The solution to the problem is, once again, to place the information in Table 7 into appropriate single-theme tables. Here is one such possible arrangement:

<table>
<thead>
<tr>
<th>Author Last Name</th>
<th>Author First Name</th>
<th>Book Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berdahl</td>
<td>Robert</td>
<td>The Politics of the Prussian Nobility</td>
</tr>
<tr>
<td>Yudof</td>
<td>Mark</td>
<td>Child Abuse and Neglect</td>
</tr>
<tr>
<td>Harmon</td>
<td>Glynn</td>
<td>Human Memory and Knowledge</td>
</tr>
<tr>
<td>Graves</td>
<td>Robert</td>
<td>The Golden Fleece</td>
</tr>
<tr>
<td>Miksa</td>
<td>Francis</td>
<td>Charles Ammi Cutter</td>
</tr>
<tr>
<td>Hunter</td>
<td>David</td>
<td>Music Publishing and Collecting</td>
</tr>
<tr>
<td>Graves</td>
<td>Robert</td>
<td>English and Scottish Ballads</td>
</tr>
</tbody>
</table>

Table 8

<table>
<thead>
<tr>
<th>Book Title</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Politics of the Prussian Nobility</td>
<td>History</td>
</tr>
<tr>
<td>Child Abuse and Neglect</td>
<td>Legal Procedures</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Human Memory and Knowledge</td>
<td>Cognitive Psychology</td>
</tr>
<tr>
<td>The Golden Fleece</td>
<td>Greek Literature</td>
</tr>
<tr>
<td>Charles Ammi Cutter</td>
<td>Library Biography</td>
</tr>
<tr>
<td>Music Publishing and Collecting</td>
<td>Music Literature</td>
</tr>
<tr>
<td>English and Scottish Ballads</td>
<td>Folksong</td>
</tr>
</tbody>
</table>

Table 9

<table>
<thead>
<tr>
<th>Subject</th>
<th>Collection or Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>History</td>
<td>PCL General Stacks</td>
</tr>
<tr>
<td>Legal Procedures</td>
<td>Law Library</td>
</tr>
<tr>
<td>Cognitive Psychology</td>
<td>PCL General Stacks</td>
</tr>
<tr>
<td>Greek Literature</td>
<td>Classics Library</td>
</tr>
<tr>
<td>Library Biography</td>
<td>Library and Information Science Collection</td>
</tr>
<tr>
<td>Music Literature</td>
<td>Fine Arts Library</td>
</tr>
<tr>
<td>Folksong</td>
<td>PCL General Stacks</td>
</tr>
</tbody>
</table>

Table 10

<table>
<thead>
<tr>
<th>Collection or Library</th>
<th>Building</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCL General Stacks</td>
<td>Perry-Castañeda Library</td>
</tr>
</tbody>
</table>
You can verify for yourself that none of these tables contains a transitive dependency; hence, all of them are in 3NF (and, in fact, in DKNF).

We can note in passing that the fact that Table 8 contains the first and last names of Robert Graves in two different rows suggests that it might be worthwhile to replace it with two further tables, along the lines of:

<table>
<thead>
<tr>
<th>Author Last Name</th>
<th>Author First Name</th>
<th>Author Identification Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berdahl</td>
<td>Robert</td>
<td>001</td>
</tr>
<tr>
<td>Yudof</td>
<td>Mark</td>
<td>002</td>
</tr>
<tr>
<td>Harmon</td>
<td>Glynn</td>
<td>003</td>
</tr>
<tr>
<td>Graves</td>
<td>Robert</td>
<td>004</td>
</tr>
<tr>
<td>Miksa</td>
<td>Francis</td>
<td>005</td>
</tr>
<tr>
<td>Hunter</td>
<td>David</td>
<td>006</td>
</tr>
</tbody>
</table>

Table 11

Table 12
<table>
<thead>
<tr>
<th>Author Identification Number</th>
<th>Book Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>The Politics of the Prussian Nobility</td>
</tr>
<tr>
<td>002</td>
<td>Child Abuse and Neglect</td>
</tr>
<tr>
<td>003</td>
<td>Human Memory and Knowledge</td>
</tr>
<tr>
<td>004</td>
<td>The Golden Fleece</td>
</tr>
<tr>
<td>005</td>
<td>Charles Ammi Cutter</td>
</tr>
<tr>
<td>006</td>
<td>Music Publishing and Collecting</td>
</tr>
<tr>
<td>004</td>
<td>English and Scottish Ballads</td>
</tr>
</tbody>
</table>

Table 13

Though Tables 12 and 13 together take a little more space than Table 8, it is easy to see that given a much larger collection, in which there would be many more authors with multiple works to their credit, Tables 12 and 13 would be more economical of storage space than Table 8. Furthermore, the structure of Tables 12 and 13 lessens the chance of making updating errors (e.g., typing Grave instead of Graves, or Miska instead of Miksa).

The Boyce-Codd Normal Form (BCNF)

The Boyce-Codd Normal Form (BCNF) deals with the anomalies that can occur when a table fails to have the property that every determinant is a candidate key. Here is an example, Table 14, that fails to have this property. (In Table 14 the SSNs are to be interpreted as those of students with the stated majors and advisers. Note that each of students 123-45-6789 and 987-65-4321 has two majors, with a different adviser for each major.)
We begin by showing that Table 14 lacks the required property, viz., that every determinant be a candidate key.

What are the determinants in Table 14? One determinant is the pair of attributes, SSN and Major. Each distinct pair of values of SSN and Major determines a unique value for the attribute, Adviser. Another determinant is the pair, SSN and Adviser, which determines unique values of the attribute, Major. Still another determinant is the attribute, Adviser, for each different value of Adviser determines a unique value of the attribute, Major. (These observations about Table 14 correspond to the real-world facts that each student has a single adviser for each of his or her majors, and each adviser advises in just one major.)

Now we need to examine these three determinants with respect to the question of whether they are candidate keys. The answer is that the pair, SSN and Major, is a candidate key, for each such pair uniquely identifies a row in Table 14. In similar fashion, the pair, SSN and Adviser, is a candidate key. But the determinant, Adviser, is not a candidate key, because the value Dewey occurs in two rows of the Adviser column. So Table 14 fails to meet the condition that every determinant in it be a candidate key.

It is easy to check on the anomalies in Table 14. For example, if student 987-65-4321 were to leave Enormous State University, the table would lose the

<table>
<thead>
<tr>
<th>SSN</th>
<th>Major</th>
<th>Adviser</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>Library and Information Science</td>
<td>Dewey</td>
</tr>
<tr>
<td>123-45-6789</td>
<td>Public Affairs</td>
<td>Roosevelt</td>
</tr>
<tr>
<td>222-33-4444</td>
<td>Library and Information Science</td>
<td>Putnam</td>
</tr>
<tr>
<td>555-12-1212</td>
<td>Library and Information Science</td>
<td>Dewey</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>Pre-Medicine</td>
<td>Semmelweis</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>Biochemistry</td>
<td>Pasteur</td>
</tr>
<tr>
<td>123-54-3210</td>
<td>Pre-Law</td>
<td>Hammurabi</td>
</tr>
</tbody>
</table>

Table 14
information that Semmelweis is an adviser for the Pre-Medicine major. As another example, Table 14 has no information about advisers for students majoring in history.

As usual, the solution lies in constructing single-theme tables containing the information in Table 14. Here are two tables that will do the job.

<table>
<thead>
<tr>
<th>SSN</th>
<th>Adviser</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>Dewey</td>
</tr>
<tr>
<td>123-45-6789</td>
<td>Roosevelt</td>
</tr>
<tr>
<td>222-33-4444</td>
<td>Putnam</td>
</tr>
<tr>
<td>555-12-1212</td>
<td>Dewey</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>Semmelweis</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>Pasteur</td>
</tr>
<tr>
<td>123-54-3210</td>
<td>Hammurabi</td>
</tr>
</tbody>
</table>

Table 15

<table>
<thead>
<tr>
<th>Major</th>
<th>Adviser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library and Information Science</td>
<td>Dewey</td>
</tr>
<tr>
<td>Public Affairs</td>
<td>Roosevelt</td>
</tr>
<tr>
<td>Library and Information Science</td>
<td>Putnam</td>
</tr>
<tr>
<td>Pre-Medicine</td>
<td>Semmelweis</td>
</tr>
<tr>
<td>Biochemistry</td>
<td>Pasteur</td>
</tr>
</tbody>
</table>
By way of an example of the value of separating Table 14 into single-theme tables, Table 16 includes information about at least one faculty member at ESU who could be the adviser of a student who wanted to major in history.

Tables 15 and 16 are in BCNF (in fact, they are in DKNF), since every determinant in them is also a candidate key. You can easily verify this statement if you note that the key in Table 15 is a composite key, SSN and Adviser.
**Aim:** Creating Cursor in Oracle.

**S/w Requirement:** Oracle 9i, 10g.

**Theory:**

There are two types of cursor
A. Explicit cursor
B. Implicit cursor

**A. Explicit cursor:**

An explicit cursor is one in which cursor name is explicitly assigned to select statement. An implicit cursor is used for all other sql statements. Processing of an explicit cursor involves four steps. Processing of an implicit cursor is taken care by PL/SQL .the declaration of the cursor is done in the declarative part of the block.

A cursor variable is a reference type. A reference type is similar to pointer.

Explicit cursor:
The set of rows return by query can contain zero or multiple rows depending upon the query defined. The rows are called active set. The cursor will point to the current row in the active set.

After declaring a cursor, we can use the following commands to control the cursor.

1. Open
2. Fetch
3. Close

```
declare
c_rollno stud.rollno%type;
cursor st is select rollno from stud where rollno=1;
begin
    open st;
    loop
        fetch st into c_rollno;
        update stud set lastname='barde' where rollno=c_rollno;
        exit when st%NOTFOUND;
    end loop;
    dbms_output.put_line('table updated');
close st;
end;
/```

outout:table updated
In the above cursor we are going to update the table. Name of table is stud and its attributes are rollno which is number, firstname which is of datatype varchar2, lastname which is of datatype varchar2.

In the Declare section we declare variable that match to the attribute in the table %TYPE matches the datatype of that variable. In the above procedure c_rollno is the variable that we declare in the procedure to access the column rollno from table stud because of %type it directly provide matching of data. In the begin section we open the cursor and fetch the values. Error handles by the exception section. Close the cursor. And end;

**B. Implicit cursor:**
These are inbuilt cursor in oracle.

---

6 **Lab Exercise**
**Aim:** Creating Procedure And Functions in Oracle

**S/w Requirement:** Oracle 9i, 10g.

**Theory:**

PL/SQL supports the two type of programming:
1. Procedure.
2. Function.

Procedures are usually used to perform any specific task and functions are used to compute a value.

**PROCEDURE:**
The basic syntax for the creating procedure is:

```
CREATE OR REPLACE PROCEDURE procedure_name (arguments) AS IS
```

```
procedure body;
```

The body of procedure is block of statements with declarative executable and exception sections.

The declarative section is located between the IS /AS keyword and BEGIN keyword.

The executable section is located between BEGIN and EXCEPTION keywords or between the BEGIN and END keywords if there is no EXCEPTION handling section.

If EXCEPTION handling is present, it is located between exception and END keywords.

**Steps for Creating Procedure:**

```
CREATE OR REPLACE PROCEDURE procedure_name (parameter list) AS IS
```

(Declarative section )

```
BEGIN
```

(Executable section )

```
EXCEPTION
```

(Error handling or exception section).
To execute the procedure we have to write a block of statement:
Begin
Procedurename(data);
End;
/

eg:
create or replace procedure addnewstud(
p_rollno stud.rollno%type,
p_firstname stud.firstname%type,
p_lastname stud.firstname%type) as
begin
insert into stud(rollno,firstname,lastname)values(p_rollno ,p_firstname,p_lastname);
end addnewstud;
/
output:Procedure created.

begin
addnewstud(2,'rohini','narwade');
end;
/

In the above procedure we are inserting data into the table. Here procedure name is addnewstud and take the variables like p_rollno, p_firstname, p_lastname having attribute in the table rollno, firstname, lastname respectively.in the begin section begin executable code is written. In the above procedure we are inserting the value so write the query to insert the data then end procedure by the end procedure then /.

FUNCTION:

Steps for Creating function:

CREATE OR REPLACE FUNCTION function _name (parameter list)AS/IS
Return datatype is/as
(local declaration)
BEGIN
A function has two parts, namely function specification and function body. The function specification begins with the keyword function and end with return clause. The function bodies begins with the keyword is/as and end with keyword end.

create or replace function studentn(
p_rollno stud.rollno%type)
return boolean as
v_firstname varchar2(20);
v_return boolean;
begin
select firstname into v_firstname from stud where rollno=p_rollno;
if (firstname='seema')then
v_return =true;
else
v_return =false;
end if;
end studentn;
/

7. Lab Exercise
**Aim:** Writing packages and triggers in oracle.

**S/w Requirement:** Oracle 9i, 10g.

**Theory:**
1. Package.
2. Triggers

**PACKAGES:**

A package is a database object, which is an encapsulation of related PL/SQL types, subprogram, cursor, exception, variables and constants. It consists of two parts: a specification and body. In a package specification, we can declare types, subprogram, cursor, exception, variables and constants. A package body implements the subprogram, cursor that are declared in package specification.

Package can be created with the following commands:

1. `create package` commands.
2. `create package body` commands.

1. `create package` commands

```
CREATE PACKAGE package_name IS declaration
BEGIN
  (executable statements)
END package_name;
```

The procedure and cursor declared in the 'create package' command is fully defined and implemented by the package body, which can be achieved by using the following syntax:

```
CREATE PACKAGE BODY package_name IS declaration
BEGIN
  (executable statements)
END package_name;
```

In the `create package` and `create package body` commands, the keyword `public` and `private` denote the usage of object declaration in the
package variables declare in the package body can be termed private to restrict their use in the package only. On the other hand public variables can be used in the package as well as outside the package.

Package specification:

The package specification contains public objects and types it also include subprogram. The specification contains the package resources required for our application. If a package specification declares only types, constants, variables, and exception, then they need not to include package body, because all information required for usage of types, constants, variables, and exception are specified in the specification.

Package body:
The package body contains the definition of every cursor and subprogram declare in the package specification and implements them private declaration can also be included in a package body.

Example:

1. create package commands:

create or replace package pack_me is
  procedure order_proc(orno varchar2);
  function order_fun (ornos varchar2 )
    return varchar2;
end pack_me;
/

2. create package body commands:

create or replace package body pack_me as
  procedure order_proc(orno varchar2)is
    stat char (1);
    begin
      select ostatus into stat from order_master
      where orderno=orno;
      if stat= ‘p’ then
        dbms_output.put_line(‘pending order’);
      else
        dbms_output.put_line(‘completed order’);
      end if;
    end order_proc;
function order_fun (ornos varchar 2) 
return varchar2 is 
icode varchar 2(5); 
ocode varchar 2 (5); 
qtyord number; 
qtydeld number; 
begi 
select qty_ord ,qty_deld, itemcode,orderno into 
qtyord,qtydeld,icode,ocode from order_detail 
where order=ornos; 
if qtyord<qtydeld then 
return ocode; 
else 
return icode; 
end if; 
end order_fun; 
end pack_me; 
/

to execute the procedure in the package: 
exec pack_me.order_no('001');

to execute the function in the package: 
declare 
a varchar2(5); 
b varchar2(5); 
b:=pack_me.order_fun('202'); 
dbms_output.put_line('the value is'||b); 
end; 
/

Triggers: 
Types of triggers 
A. before 
B. after 
C. for each row 
D. for each statement.

SYNTAX FOR TRIGGERS: 
CREATE OR REPLACE TRIGGER trigger_name 
BEFORE /AFTER INSERT/UPADTE/DELETE ON table_name 
REFERENCING {OLD AS OLD/NEW AS NEW} 
FOR EACH STATEMENT/ FOR EACH ROW when condition
Before/After options:

The before/after options can be used to specify when the trigger body should be fired with respect to the triggering statement. If the user include a before option, then Oracle fires the triggers before executing the triggering statement. On the other hand if AFTER is used then, oracle fires the trigger after executing the triggering statement.

For each row/ statement:

For each row/ statement option included in the ‘create trigger ‘ syntax specifies that the triggers fires once per row . By defaults, database triggers a database triggers fires for each statement .

```plsql
create or replace trigger orders
    before insert on order_detail for each row
declare
    orno order_detail.orderno%type
begin
    select orno into orno from order_detail
    where qty_ord <qyt_deld;
    if orno =’001’ then
        raise_application_error(-200001,’enter some other name’);
    end if;
end;
/
```

In this trigger before inserting order table it check the condition orno (order number) equal to 001 and condition is satisfied then the value updated else error is occurred.

```plsql
create or replace trigger up_trig
    before update order_detail for each row
begin
    update order_detail set itemcode =’i200’ where qty_ord=50;
end;
```

8. **Lab Exercise**

**Aim:** TEAM Project
A team should contain 2-3 students from same batch. Project report should be as per IEEE format. All the phases of SDLC should be followed in project. No project topic should repeat in the class.

Mini Project should follow the steps below:
1) Define problem with specification.
2) Define the functionality.
3) Design solution for project.
4) Implement the solution.
5) Present and Evaluate the project.

**Viva-voce Questions:**

What is database?
What is DBMS?
Advantage of database
What is data model?
What is object oriented model?
What is an entity?
What is an entity type?
What is an entity set?
What is weak entity set?
What is relationship?
What is DDL?
What is DML?
What is normalization?
What is functional dependency?
What is 1st NF?
What is 2nd NF?
What is 3rd NF?
What is BCNF?
What is fully functional dependency?
What do you mean of aggregation, atomicity?
What are different phases of transaction?
What do you mean of flat file database?
What is query?
What is the name of buffer where all commands are stored?
Are the resulting of relation PRODUCT and JOIN operation is same?

9. Quiz on the subject:
Quiz should be conducted on tips in the laboratory, recent trends and subject knowledge of the subject. The quiz questions should be formulated such that
questions are normally are from the scope outside of the books. However twisted questions and self formulated questions by the faculty can be asked but correctness of it is necessarily to be thoroughly checked before the conduction of the quiz.

10. Conduction of Viva-Voce Examinations:

Teacher should oral exams of the students with full preparation. Normally, the objective questions with guess are to be avoided. To make it meaningful, the questions should be such that depth of the students in the subject is tested. Oral examinations are to be conducted in co-cordial environment amongst the teachers taking the examination. Teachers taking such examinations should not have ill thoughts about each other and courtesies should be offered to each other in case of difference of opinion, which should be critically suppressed in front of the students.

11. Evaluation and marking system:

Basic honesty in the evaluation and marking system is absolutely essential and in the process impartial nature of the evaluator is required in the examination system to become popular amongst the students. It is a wrong approach or concept to award the students by way of easy marking to get cheap popularity among the students to which they do not deserve. It is a primary responsibility of the teacher that right students who are really putting up lot of hard work with right kind of intelligence are correctly awarded.

The marking patterns should be justifiable to the students without any ambiguity and teacher should see that students are faced with unjust circumstances.

The assessment is done according to the directives of the Principal/ Vice-Principal/ Dean Academics.