INSTRUCTION MANUAL

Relational Data Base Management Systems Lab

(BTCS-506)

Prepared by

Er. Poonamdeep Kaur
Assistant Professor (CSE)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA-141006

DECLARATION

This Manual of Relational Data Base Management Systems Lab (BTCS-506) has been prepared by me as per syllabus of Relational Data Base Management Systems Lab (BTCS-506).

Signature

Syllabus

BTCS-506RDBMS LAB

External Marks: 20
Internal Marks: 30
Total Marks: 50

L T P
- - 4

1 Introduction to SQL and installation of SQL Server / Oracle.

2. Data Types, Creating Tables, Retrieval of Rows using Select Statement, Conditional Retrieval of Rows, Alter and Drop Statements.

3. Working with Null Values, Matching a pattern from a Table, Ordering the Result of a query, Aggregate Functions, Grouping the Result of a query, Update and Delete Statements.

4. Set Operators, Nested Queries, Joins, Sequences.

5. Views, Indexes, Database Security and privileges: Grant and Revoke Commands, and Rollback Commands.

6. PL/SQL Architecture, Assignments and Expressions, Writing PL/SQL Code, Referencing Non SQL parameters.

7. Stored Procedures and Exception Handling.

8. Triggers and Cursor Management in PL/SQL

Suggested Tools - MySQL, DB2, Oracle, SQL Server2012, postgre SQL, SQL Lite

INDEX

Exercise-1

PRACTICAL :- Intoduction of Sql and Installation of SQL/Oracle

Intoduction to SQL

SQL, which is an abbreviation for **Structured Query Language**, is a language to request data from a database, to add, update, or remove data within a database, or to manipulate the metadata of the database.

SQL is generally pronounced as the three letters in the name, e.g. *ess-cue-ell*, or in some people's usage, as the word *sequel*.

SQL is a declarative language in which the expected result or operation is given without the specific details about how to accomplish the task. The steps required to execute SQL statements are handled transparently by the SQL database. Sometimes SQL is characterized as *non-procedural* because procedural languages generally require the details of the operations to be specified, such as opening and closing tables, loading and searching indexes, or flushing buffers and writing data to filesystems. Therefore, SQL is considered to be designed at a higher conceptual level of operation than procedural languages because the lower level logical and physical operations aren't specified and are determined by the SQL engine or server process that executes it.

Instructions are given in the form of statements, consisting of a specific SQL statement and additional parameters and operands that apply to that statement. SQL statements and their modifiers are based upon official SQL standards and certain extensions to that each database provider implements. Commonly used statements are grouped into the following categories:

Data Query Language (DQL)

- SELECT - Used to retrieve certain records from one or more tables.

Data Manipulation Language (DML)

- INSERT - Used to create a record.
- UPDATE - Used to change certain records.
- DELETE - Used to delete certain records.

Data Definition Language (DDL)

- CREATE - Used to create a new table, a view of a table, or other object in database.
- ALTER - Used to modify an existing database object, such as a table.
- DROP - Used to delete an entire table, a view of a table or other object in the database.

Data Control Language (DCL)

- GRANT - Used to give a privilege to someone.
- REVOKE - Used to take back privileges granted to someone

Installation of Oracle 10g

This describes how to quickly install Oracle Database on Windows systems. It includes information about the following:

1 Log In to the System with Administrator Privileges

Log on as a member of the Administrators group to the computer on which to install Oracle components. Log on as a member of the Domain Administrators group if you are installing on a Primary Domain Controller (PDC) or a Backup Domain Controller (BDC).

2 Hardware Requirements

Table 1 Hardware Requirements

Requirement	Minimum Value
Physical memory (RAM)	256 MB minimum, 512 MB recommended
Virtual memory	Double the amount of RAM
Temp disk space	100 MB
Hard disk space	1.5 GB
Video adapter	256 colors
Processor	200 MHz minimum

To ensure that the system meets these requirements, follow these steps:

- Determine the physical RAM size. For a computer using Windows 2000, for example, open **System** in the control panel and select the **General** tab. If the size of the physical RAM installed in the system is less than the required size, then you must install more memory before continuing.
- Determine the size of the configured swap space (also known as paging file size). For a computer using Windows 2000, for example, open **System** in the control panel, select the **Advanced** tab, and click **Performance Options**.

  If necessary, see your operating system documentation for information about how to configure additional swap space.
- Determine the amount of disk space available in the temp directory. This is equivalent to the total amount of free disk space, minus what will be needed for the Oracle software to be installed.

  If there is less than 100 MB of disk space available in the temp directory, then first delete all unnecessary files. If the temp disk space is still less than 100 MB, then set the TEMP or TMP environment variable to point to a different hard drive. For a computer using Windows 2000, for example, open the **System** control panel, select the **Advanced** tab, and click **Environment Variables**.
- Determine the amount of free disk space on the system. For a computer using Windows 2000, for example, open **My Computer**, right-click the drive where the Oracle software is to be installed, and choose **Properties**.

3 Software Requirements

Table 2 Software Requirements

Requirement	Value
System Architecture	32-bit **Note:**Oracle provides both 32-bit and 64-bit versions of Oracle Database for Windows. Currently, the 32-bit version of the database must run on the 32-bit version of the operating system. The 64-bit version of the database must run on the 64-bit version of the operating system.
Operating System	Oracle Database for Windows is supported on the following operating systems: • Windows NT Server 4.0, Windows NT Server Enterprise Edition 4.0, and Terminal Server Edition with service pack 6a or higher are supported. Windows NT Workstation is no longer supported. • Windows 2000 with service pack 1 or higher. All editions, including Terminal Services and Windows 2000 MultiLanguage Edition (MLE), are supported. • Windows Server 2003 • Windows XP Professional Windows Multilingual User Interface Pack is supported on Windows XP Professional and Windows Server 2003.
Compiler	Oracle C++ Call Interface supports the following compilers: Microsoft Visual C++ 6.0, Microsoft Visual C++ .NET 2002, and Microsoft Visual C++ .NET 2003 Oracle Call Interface supports the following compilers: Microsoft Visual C++ 6.0, Microsoft Visual C++ .NET 2002, and Microsoft Visual C++ .NET 2003 External callouts support the following compilers: Microsoft Visual C++ 6.0, Microsoft Visual C++ .NET 2002, and Microsoft Visual C++ .NET 2003 PL/SQL native compilation supports the following compilers: Microsoft Visual C++ 6.0, Microsoft Visual C++ .NET 2002, and Microsoft Visual C++

	.NET 2003 Pro*COBOL supports the Micro Focus Net Express compiler. Object Oriented COBOL (OOCOBOL) specifications are not supported. XDK supports the following compilers: Microsoft Visual C++ 6.0, Microsoft Visual C++ .NET 2002, and Microsoft Visual C++ .NET 2003
Network Protocol	The Oracle Net foundation layer uses Oracle protocol support to communicate with the following industry-standard network protocols: • TCP/IP • TCP/IP with SSL • Named Pipes

4.Install the Oracle Database Software

Start Oracle Universal Installer and install Oracle software, as follows:

- Insert the CD-ROM labeled Oracle Database 10*g* Release 1 (10.1) Disk 1 of 1 or navigate to the Oracle Database location on the Oracle Database 10*g* Release 1 (10.1) DVD-ROM.

  When installing from a hard disk, double-click setup.exe.

  When installing from the installation media, the Autorun screen automatically appears. If the Autorun screen does not appear, then:

  - Click **StartRun**.
  - Enter the following:

    DRIVE_LETTER:\autorun\autorun.exe

    The Autorun screen appears. Click **Install/Deinstall Products** from the Autorun screen. The Welcome screen appears.

- Use the following guidelines to complete the installation:
  - Table 3 describes the recommended action for each Oracle Universal Installer screen.
  - If you need more assistance, or if you want to choose an option that is not a default, click **Help** for additional information.

- If you encounter errors while installing or linking the software, see the *Oracle Database Installation Guide for Windows* for information about troubleshooting.

Oracle Universal Installer Screens

Screen	**Recommended Action**
Welcome	Select **Basic Installation**or **Advanced Installation**. Select **Basic Installation**installation method if you want to quickly install Oracle Database. This installation method requires minimal user input and you will not see many of the screens listed in this table. It installs the software and optionally creates a general-purpose database using the information that you specify on this screen. Click **Next**.
Specify File Locations	In the **Destination**section, accept the default values or enter the Oracle home name and directory path in which to install Oracle components. Click **Next**.
Select Installation Type	Select **Enterprise Edition**, **Standard Edition**, **Personal Edition**, or **Custom**. Click **Next**.
Select Database Configuration	Select the database configuration that best meets your needs. See the online help provided by either Oracle Universal Installer or Database Configuration Assistant for a description of these preconfigured database types. Click **Next**.
Specify Database Configuration Options	Specify the following information, then click **Next**: **Global Database Name** Specify a name for the database, followed by the domain name of the system: sales.*your_domain.com* The value that you specify, up to the first period, is also used for the **SID**value. **Select Database Character Set** Accept the default value, which is based on your system locale, or if you need to support more than one language, click **Help**for more information about the

	supported character sets. **Create database with example schemas** Choose this option to create the EXAMPLEtablespace that contains the Sample Schemas (optional, but recommended).
Select Database Management Option	Accept the default values, then click **Next**. **Note:**You can enable e-mail notifications after you have installed the software.
Specify Database File Storage Option	Select the **File System**option and specify the database file location, then click **Next**. **Specify database file location:** Accept the default location or specify a new file location.
Specify Backup and Recovery Options	Accept the default values, then click **Next**. **Note:**You can enable automated backups after you have installed the software.
Specify Database Schema Passwords	Enter and confirm passwords for all of the privileged database accounts, then click **Next**. **Note:**Oracle recommends that you specify a different password for each account. You must remember the passwords that you specify.
Summary	Review the information displayed, then click **Install**.
Install	The Install screen displays status information while the product is being installed.
Configuration Assistants	The Configuration Assistants screen displays status information for the configuration assistants that configure the software and create a database. After Database Configuration Assistant finishes, review the information on the screen. Make a note of the following information: • Enterprise Manager URL • Database creation logfiles location • Global Database Name

	• System Identifier (SID) • Server parameter filename and location Click **OK**to continue or click **Password Management**to unlock accounts and set passwords.
End of Installation	The configuration assistants configure several Web-based applications, including Oracle Enterprise Manager Database Control. This screen displays the URLs configured for these applications. Make a note of the URLs used. The port numbers used in these URLs are also recorded in the following file: *ORACLE_BASE**ORACLE_HOME*\install\portlist.ini To exit from Oracle Universal Installer, click **Exit**, then click **Yes**. Oracle Enterprise Manager Database Control displays in a Web browser.

5. Install Products from the Oracle Database Companion CD

The Oracle Database Companion CD contains products that improve the performance of or complement Oracle Database. For most installations, Oracle recommends that you install Oracle Database 10*g* Products from the Companion CD.

Products Included on the Companion CD

The Companion CD includes two sets of products:

- **Oracle Database 10*g* Products**

  Includes Oracle Database Examples, natively compiled Java libraries for Oracle JVM and Oracle *inter*Media, Oracle Text supplied knowledge bases, and Legato Single Server Version (LSSV).

- **Oracle Database 10*g* Companion Products**

  Includes Oracle HTTP Server and Oracle HTML DB.

The following subsection describes how to install Oracle Database 10*g* Products. For more information about the products on the Companion CD, and for more detailed information about installing them, see the *Oracle Database Companion CD Installation Guide for Windows* which is located on the Companion CD.

6.Installing Oracle Database 10*g* Products

To install Oracle Database 10*g* Products, follow these steps:

- Insert the CD labeled Oracle Database Companion CD 10*g* Release 1 (10.1) Disk 1 of 1 in the disk drive.

  When installing from a hard disk, double-click setup.exe.

  When installing from the Oracle Database Companion CD, the Autorun screen automatically appears. If the Autorun screen does not appear, then:

  - Click **Start > Run**.
  - Enter the following:

    DRIVE_LETTER:\autorun\autorun.exe

The Autorun screen appears. Click **Install/Deinstall Products** from the Autorun screen.

- Use the following guidelines to complete the installation:
  - On the Specify File Locations screen, select the Oracle home name and path for the Oracle Database installation where you want to install the components.
  - On the Select a Product to Install Screen, select **Oracle Database 10g Products**.

Screen	**Recommended Action**
Welcome	Click **Next**.
Specify File Locations	In the **Destination**section, select the **Name**or **Path**value that specifies the Oracle home directory where you installed Oracle Database 10*g*, then click **Next**. The default Oracle home path is similar to the following: c:\oracle\product\10.1.0\db_1
Select a Product to Install	Select **Oracle Database 10g Products**, then click **Next**.
Summary	Review the information displayed, then click **Install**.
Install	The Install screen displays status information while the product is being installed.
End of Installation	To exit from Oracle Universal Installer, click **Exit**, then click **Yes**.

Exercise-2

PRACTICAL :-Data Types, Creating Tables, Retrieval of Rows using Select Statement, Conditional Retrieval of Rows, Alter and Drop Statements.

Oracle workgroup or server is the largest selling RDBMS product.it is estimated that the combined sales of both these oracle database product account for aroud 80% of the RDBMSsystems sold worldwide.
These products are constantly undergoing change and evolving.The natural language of this RDBMS product is ANSI SQL,PL/SQL a superset of ANSI SQL.oracle 8i and 9i also under stand SQLJ.

Oracle corp has also incorporated a full-fledged java virtual machine into its database engine.since both executable share the same memory space the JVM can communicate with the database engine with ease and has direct access to oracle tables and their data.

Data Types :- Oracle supports following types of data types.

• CHAR (SIZE) :- Fixed length character data of length SIZE bytes. The maximum length is 255 bytes in Oracle 7 and 2000 bytes in Oracle 8 onwards. Default and minimum size is 1 byte.
• VARCHAR2(SIZE) :- Variable length character string having maximum length SIZE bytes. The maximum length 2000 bytes in Oracle 7 and 4000 bytes in Oracle 8 onwards. The minimum size is 1
• NUMBER(L) :- Numeric data with number of digits L.
• NUMBER(L, D) :- Numeric data with total number of digits L and number of digits D after decimal point.
• DATE :- Valid date range. The date ranges from January 1, 4712 BC to December 31, 9999 AD.
• LONG :- Character data of variable length which stores upto 2 Gigabytes of data. (A bigger version the VARCHAR2 datatype).
• INTEGER :- Integer type of Data. It is actually a synonym for NUMBER(38)

Different types of commands in SQL:

A). **DDL commands:-**To create a database objects
B). **DML commands:-**To manipulate data of a database objects
C). **DQL command: -**To retrieve the data from a database.
D). **DCL/DTL commands:-**To control the data of a databas

DDL commands:

1. The Create Table Command:- it defines each column of the table uniquely. Each column has minimum of three attributes, a name , data type and size.

Syntax:
Create table<table name>(<col1><datatype>(<size>),<col2><datatype><size>));

Ex:
create table emp(empno number(4) primary key, ename char(10));

2. Modifying the structure of tables.
a)add new columns

Syntax:
Alter table<tablename> add(<new col><datatype(size),<new col>datatype(size));

Ex:
alter table emp add(sal number(7,2));

3.Dropping a column from a table.

Syntax:
Alter table <tablename> drop column <col>;

Ex:
alter table emp drop column sal;

4. Modifying existing columns.

Syntax:
Alter table <tablename> modify(<col><newdatatype>(<newsize>));

Ex:
alter table emp modify(ename varchar2(15));

5. Renaming the tables

Syntax:
Rename<oldtable> to <new table>;

Ex:
rename emp to emp1;

6.truncating the tables.

Syntax:
Truncate table<tablename>;

Ex:
trunc table emp1;

7. Destroying tables.

Syntax:
Drop table<tablename>;

Ex:
drop table emp;

DML commands:

8.Inserting Data into Tables:- once a table is created the most natural thing to do is load this table with data to be manipulated later.

Syntax:
insert into <tablename> (<col1>,<col2>) values(<exp>,<exp>);

9.Delete operations.

a)remove all rows
Syntax:
delete from <tablename>;

b)removal of a specified row/s
Syntax:
delete from <tablename> where <condition>;

10. Updating the contents of a table.

a)updating all rows
Syntax:
Update <tablename> set <col>=<exp>,<col>=<exp>;

b)updating seleted records.
Syntax:

Update <tablename> set <col>=<exp>,<col>=<exp> where <condition>;

11. Types of data constrains.

a)not null constraint at column level.
Syntax:
<col><datatype>(size)not null

b) unique constraint
Syntax:
Unique constraint at column level.
<col><datatype>(size)unique;

c)unique constraint at table level:
Syntax:
Create tabletablename(col=format,col=format,unique(<col1>,<col2>);

d)primary key constraint at column level
Syntax:
<col><datatype>(size)primary key;

e) primary key constraint at table level.
Syntax:
Create table tablename(col=format,col=format
primary key(col1>,<col2>);

f)foreign key constraint at column level.
Syntax:
<col><datatype>(size>) references <tablename>[<col>];

g) foreign key constraint at table level
Syntax:
foreign key(<col>[,<col>])references<tablename>[(<col>,<col>)

h)check constraint
check constraint constraint at column level.
Syntax: <col><datatype>(size) check(<logical expression>)

i) check constraint constraint at table level.
Syntax:check(<logical expression>)

DCL commands:

Oracle provides extensive feature in order to safeguard information stored in its tables from unauthoraised viewing and damage.The rights that allow the user of some or all oracle resources on the server are called privileges.

a)Grant privileges using the GRANT statement

The grant statement provides various types of access to database objects such as tables,views and sequences and so on.

Syntax:
GRANT <object privileges>
ON <objectname>
TO<username>
[WITH GRANT OPTION];

b)Revoke permissions using the REVOKE statement:

The REVOKE statement is used to deny the Grant given on an object.

Syntax:
REVOKE<object privilege>
ON
FROM<user name>;

Lab Assignments

Following tables (Relations) are considered for the lab purpose.
- SAILORS (SID:INTEGER, SNAME:STRING, RATING:INTEGER, AGE:REAL)
- BOATS (BID:INTEGER, BNAME:STRING, COLOR:STRING)
- RESERVES (SID:INTEGER, BID:INTEGER, DAY:DATE)

Creating Tables :-
The CREATE TABLE command is used to create the table (relation) in SQL.
CREATE TABLE TABLENAME (ATT_NAME1 DATATYPE, ATT_NAME2 DATATYPE, ATT_NAME3 DATATYPE, …..);
SQL> CREATE TABLE SAILORS (SID NUMBER (5), SNAME VARCHAR2(30), RATING NUMBER(5), AGE NUMBER(4,2));

Primary Key & Foreign Key :- Consider the Sailors relation and the constraint that no two sailors have the same SID. This type of constraint can be defined using **Primary Key**, which
gives uniqueness for the value of attribute defined (Eg. SID).
Similarly, a sailor can't reserve a boat unless he/she is a valid sailor i.e. the SID of Reserves relation must available in the Sailors relation. This type of constraint can be defined using **Foreign Key**, which gives the existence of the value of attribute in one relation is depends on value in another relation.
We can use Primary Key or/and Foreign Key constraint while creating table.

Creating tables with Primary Key
CREATE TABLE TABLENAME (ATT_NAME1 DATATYPE, ATT_NAME2 DATATYPE, ATT_NAME3 DATATYPE ….., **PRIMARY KEY(ATT_NAMES)**);
SQL> CREATE TABLE SAILORS (SID NUMBER(5), SNAME VARCHAR2(30), RATING NUMBER(5), AGE NUMBER(4,2), , **PRIMARY KEY(SID)**);

Creating tables with Foreign Key
CREATE TABLE TABLENAME (ATT_NAME1 DATATYPE, ATT_NAME2 DATATYPE, ATT_NAME3 DATATYPE ….., **FOREIGN KEY (ATT_NAME) REFERENCES TABLENAME2)**);
SQL> CREATE TABLE RESERVES (SID NUMBER(5), BID NUMBER(5), DAY DATE, **FOREIGN KEY (SID) REFERENCES (SAILORS)**);

The following example gives the complete definition to create Reserves table (Defines Primary Key as well as Foreign Keys).
SQL> CREATE TABLE RESERVES (SID NUMBER(5), BID NUMBER(5), DAY DATE, PRIMARY KEY (SID, BID, DAY), FOREIGN KEY (SID) REFERENCES (SAILORS) , FOREIGN KEY (BID) REFERENCES (BOATS));

Similar way we can create Sailors as well as Boats table using Primary Key constraint.

Creating table with some Constraint :- Suppose we want to add rule for rating of the sailors - "Rating should be between 1 to 10" while creating table then we can use following

command.
SQL> CREATE TABLE SAILORS (SID NUMBER(5), SNAME VARCHAR2(30), RATING NUMBER(5), AGE NUMBER(4,2), , PRIMARY KEY(SID), **CHECK (RATING >=1 AND RATING <=10)**);

Deleting Table :- The table along with its definition & data can be deleted using following command.
DROP TABLE <TABLENAME>;
SQL> DROP TABLE SAILORS;

Adding & Deleting the Attributes and Constraints to the Table :-

- To add the attribute to a existing relation we can use ALTER TABLE Command.

ALTER TABLE <TABLENAME> ADD COLUMN ATT_NAME DATATYPE;
SQL> ALTER TABLE SAILORS ADD COLUMN SALARY NUMBER(7,2);

- To remove the attribute from an existing relation we can use following Command.

ALTER TABLE <TABLENAME> DROP COLUMN ATT_NAME;
SQL> ALTER TABLE SAILORS DROP COLUMN SALARY;

- To add the constraint to existing relation we can use ALTER TABLE Command.

ALTER TABLE <TABLENAME> ADD CONSTRAINT <CON_NAME>
<CON_DEFINITION>;
SQL> ALTER TABLE SAILORS ADD CONSTRAINT RATE CHECK (RATING >= 1 AND
RATING <=10);

Similarly we can add primary key or foreign key constraint.

- To delete the constraint to existing relation we can use following Command.

DROP CONSTRAINT <CON_NAME>;
SQL> DROP CONSTRAINT RATE;

Similarly we can drop primary key or foreign key constraint.

Adding data to the Table :- We can add data to table by using INSERT INTO command. While adding the data to the table we must remember the order of attributes as well

as their data types as defined while creating table. The syntax is as follows.

INSERT INTO <TABLENAME> VALUES (VALUE1, VALUE2, VALUE3, …..);
SQL> INSERT INTO SAILORS VALUES (1, 'Rajesh', 10, 30);

But sometimes while adding data we may not remember the exact order or sometimes we want to insert few values then we can use following format to add data to a table.

INSERT INTO <TABLENAME> (ATT_NAME1, ATT_NAME2, ATT_NAME3, …..)
VALUES (VALUE1, VALUE2, VALUE3, …..);
SQL> INSERT INTO SAILORS (SNAME, SID, AGE, RATING) VALUES ('Rajesh', 1, 30, 10);

By using one of these methods we can add records or data to Sailors, Boats as well as Reserves Table.

To see the records :- To view all records present in the table.
SELECT * FROM <TABLENAME>
SQL> SELECT * FROM SAILORS;

To delete the record(s) :- To delete all records from table or a single/multiple records which matches the given condition, we can use DELETE FROM command as follows.
DELETE FROM <TABLENAME> WHERE <CONDITION>;
SQL> DELETE FROM SAILORS WHERE SNAME = 'Rajesh';
To delete all records from the table
DELETE FROM <TABLENAME>;
SQL> DELETE FROM SAILORS;

To change particular value :- We can modify the column values in an existing row using the UPDATE command.
UPDATE <TABLENAME> SET ATT_NAME = NEW_VALUE WHERE CONDITION;
SQL> UPDATE SAILORS SET RATING = 9 WHERE SID = 1;

To update all records without any condition.
SQL> UPDATE SAILORS SET RATING = RATING + 1;

Simple Queries on the Tables :- The basic form of an SQL query is:
SELECT <SELECT_LIST>FROM <TABLE_LIST> WHERE <CONDITION>;

Q1) Display names & ages of all sailors.

SQL> SELECT SNMAE, AGE FROM SAILORS;

Write queries for

1) Find all sailors with a rating above 7.
2) Display all the names & colors of the boats.
3) Find all the boats with Red color.

Queries on multiple tables
Q2) Find the names of sailors who have reserved boat number 123.
SQL> SELECT SNAME FROM SAILORS S, RESERVES R WHERE S.SID = R.SID AND R.BID = 123;

Write queries for
1) Find SIDs of sailors who have reserved Pink Boat;
2) Find the color of the boats reserved by Rajesh.
The ALTER TABLE Statement
The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

ALTER TABLE COMMAND

- To add a column in a table, use the following syntax:

ALTER TABLE table_name
ADD column_name datatype

- To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE table_name
DROP COLUMN column_name

ALTER TABLE Example
Look at the "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn ,10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to add a column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

ALTER TABLE Persons
ADD DateOfBirth date

Notice that the new column, "DateOfBirth", is of type date and is going to hold a date. The data type specifies what type of data the column can hold. For a complete reference of all the

data types available in MS Access, MySQL, and SQL Server, go to our complete Data Types reference.

The "Persons" table will now like this:

P_Id	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

QL ALTER TABLE Example
Look at the "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to add a column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
ALTER TABLE Persons
ADD DateOfBirth date
```

Notice that the new column, "DateOfBirth", is of type date and is going to hold a date. The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MS Access, MySQL, and SQL Server, go to our complete Data Types reference.

The "Persons" table will now like this:

P_Id	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

DROP COLUMN Example
Next, we want to delete the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
ALTER TABLE Persons
DROP COLUMN DateOfBirth
```

The "Persons" table will now like this:

P_Id	LastName	FirstName	Address	City

1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The DROP TABLE Statement

The DROP TABLE statement is used to delete a table.

DROP TABLE table_name

Viva-Voce:

Q1. What is SQL?
Ans: Structured Query Language

2. What is database?

A database is a logically coherent collection of data with some inherent meaning, representing some aspect of real world and which is designed, built and populated with data for a specific purpose.

3. What is DBMS?

It is a collection of programs that enables user to create and maintain a database. In other words it is general-purpose software that provides the users with the processes of defining, constructing and manipulating the database for various applications.

4. What is a Database system?

The database and DBMS software together is called as Database system.

5. Advantages of DBMS?

- Redundancy is controlled.
- Unauthorised access is restricted.
- Providing multiple user interfaces.
- Enforcing integrity constraints.
- Providing backup and recovery.

6. Disadvantage in File Processing System?

- Data redundancy & inconsistency.
- Difficult in accessing data.
- Data isolation.
- Data integrity.
- Concurrent access is not possible.
- Security Problems.

Exercise-3

PRACTICAL :- . Working with Null Values, Matching a pattern from a Table, Ordering the Result of a query, Aggregate Functions, Grouping the Result of a query, Update and Delete Statements.

(a) Working with Null Values NULL

- values represent missing unknown data.
- By default, a table column can hold NULL values.
- This exercise will explain the IS NULL and IS NOT NULL operators.

SQL NULL Values

If a column in a table is optional, we can insert a new record or update an existing record without adding a value to this column. This means that the field will be saved with a NULL value.

- ✓ NULL values are treated differently from other values.
- ✓ NULL is used as a placeholder for unknown or inapplicable values.

SQL Working with NULL Values

Look at the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola		Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari		Stavanger

Suppose that the "Address" column in the "Persons" table is optional. This means that if we insert a record with no value for the "Address" column, the "Address" column will be saved with a NULL value.

How can we test for NULL values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

SQL IS NULL

How do we select only the records with NULL values in the "Address" column?

We will have to use the IS NULL operator:

SELECT LastName,FirstName,Address FROM Persons

WHERE Address IS NULL

The result-set will look like this:

LastName	FirstName	Address
Hansen	Ola	
Pettersen	Kari	

SQL IS NOT NULL

How do we select only the records with no NULL values in the "Address" column?

We will have to use the IS NOT NULL operator:

SELECT LastName,FirstName,Address FROM Persons

WHERE Address IS NOT NULL

The result-set will look like this:

LastName	FirstName	Address
Svendson	Tove	Borgvn 23

(b) Pattern Matching

Pattern matching applies to the use of the LIKE keyword in a WHERE clause. This feature enables the finding of text strings with particular characteristics such as all starting with the same character.

Patterns are described by the use of special characters together with ordinary characters which are to be matched in the string being searched.

Characters in a pattern are taken literally unless they are one of the special pattern matching characters described below. For example,

```
WHERE  ADDRESS  LIKE  'Ave'
```

finds all values of ADDRESS containing the string "Ave". The string "Ave" may appear anywhere. This is different to the EQ relational operator, as in:

```
WHERE  ADDRESS  EQ  'Ave'
```

This condition will only be true when ADDRESS is exactly equal to the string "Ave".

For example, to find the name of everyone whose first name starts with "B" and second name starts with "L":

```
SELECT ID  NAME  CURRPOS  -

FROM   EMPLOYEE  -

WHERE  NAME  LIKE '%B?*L'
```

searches for and finds all rows in the EMPLOYEE table in which NAME starts with the letter "B" followed by zero, one, or more intervening characters followed by the letter "L".

Trim and Upper

Pattern matching is affected by system parameters CMPTRIM and CMPUPPER. CMPTRIM causes trailing blanks of strings to be trimmed before they are compared. CMPUPPER maps all strings to upper case before the comparison takes place. Generally, when using the LIKE function, SET CMPTRIM and CLEAR CMPUPPER. By default, both parameters are SET.

Special Characters

%	beginning of line
$	end of line
?	match any single character

[start a character class

- range of characters

] end a character class

! negate a character class

* closure, zero or more occurrences

+ closure, one or more occurrences

Beginning of the Line %

The % character specifies searching for patterns at the beginning of a string variable.

To find all rows that have a string variable which begin with the word PROCESS, use:

```
... WHERE variable LIKE '%PROCESS'
```

This returns only those rows that begin with the string "PROCESS". It does not return rows containing "PROCESS" in the middle of the variable such as "END PROCESS" or "EXIT PROCESS".

End of the Line $

The $ character specifies searching for patterns at the end of a string variable.

To search for all records in which the NAME column ends in "smith" :

```
SELECT  ID  NAME   FROM  EMPLOYEE  -
      WHERE   NAME  LIKE 'smith$'
```

Match Anything Character ?

The character ? matches any single character. For example,

... WHERE NAME LIKE 'A?e'

finds names containing strings such as:

Aae Abe Ace ..... Axe Aye Aze

and also:

A+e A-e A*e A/e A,e A.e A(e A)e A'e A"e

The match anything character can appear more than once in a pattern. The next example, selects all records in which the customer identifier begins with the letters AC followed by any three characters followed by a 9. Notice the use of two special characters, the % and the ?.

```
SELECT CUSTID CUSTNAME ADDRESS PHONE -
    FROM CUSTFILE -
    WHERE CUSTID LIKE '%AC???9'
```

Classes of Characters [...]

Search for a class or set of characters by enclosing them in square brackets. Some examples of character classes are:

[12] match all instances of "1" or "2" or both

[123] match all possible combinations of "1", "2", and "3"

[a-z] match lowercase letters

[A-Z] match uppercase letters

[0-9] match decimal digits

[J-Q] match uppercase letters "J" through "Q"

[A-Z,a-z] match uppercase and lowercase letters

For example, to locate information on 1984 accounts. The account identifiers for 1984 begin with an uppercase letter followed by the string "1984". The Where clause might be:

```
... WHERE ACCTID LIKE '%[A-Z]1984'
```

Negated Character Class [!...]

To match all lines except those containing the members of a character class, place the negation character ! at the beginning of the class inside the square brackets. For examples:

[!12] match all characters except "1" and "2"

[!a-z] match all characters except lower case letters

[!A-Z] match all characters except upper case letters

[!0-9] match all characters except decimal digits

[!J-Q] match all characters except upper case letters "J" through "Q"

[!A-Za-z] match all characters except upper and lower case letters

For example, to search and delete all rows in which the value of DEPTNUM is not composed entirely of digits.

```
DELETE  FROM  TCOMPANY.TAB1  -

      WHERE   DEPTNUM  LIKE '[!0-9]'
```

Closure Character (Zero or More Occurrences) *

To search for strings or patterns of characters that occur an indefinite number of times (known as a closure) specify the closure character "*" after the required pattern.

Some examples of closure patterns are:

a* zero or more occurrences of lowercase a

[A-Z]* zero or more uppercase letters

[Q3x]* zero or more occurrences of "Q" or "3" or "x"

[a-z,A-Z]* zero or more letters, upper or lower case

this pattern matches a word of text or a null string

For example, to search for all text that appears inside a pair of parentheses :

```
... WHERE STRING LIKE '(?*)'
```

The pattern requests all lines that contain "(" followed by zero or more occurrences of any character followed by ")".

Similarly, to search for all Illinois accounts. These are identified in the ACCTID when characters 3 and 4 are "IL" and the last two (verification) digits are "13". The ACCTID may be 10 to 17 characters long and therefore the last two digits may appear in positions 9-10, 10-11, ..., 16-17. The closure character takes care of this problem.

```
SELECT * FROM ACCOUNTS -

    WHERE  ACCTID LIKE '%??IL?*13$'
```

Note the beginning and end of line characters. The % character followed by ??IL makes sure that "IL" appears in position 3 and 4. The $ character preceded by 13 makes sure that "13" appears as the last two digits. The ?* notation means that any number of characters can appear between "IL" and "13".

Closure Character (One or More Occurrences)+

This specifies a search for one or more occurrences of a pattern instead of zero or more occurrences. For example, the following command:

```
... WHERE STRING LIKE ' [aehrt]+ '
```

searches for complete words made up of the letters a,e,h,r,t.

The search pattern requests strings containing a blank followed by one or more occurrences of the letters a,e,h,r,t followed by another blank. The lines listed contain words such as "a", "are", "at", "here", "rather", "that", "the", "there", "three", etc.

Escape Character @

The special characters represent instructions sent to the pattern matching routine. Occurrences of these special characters cannot be searched for in the normal way. A search for question marks in a field cannot be specified as:

```
... WHERE string LIKE '?'
```

because this command will match every character.

The escape character @ is provided to handle this situation. Precede any special character with @, and the character is treated literally. Thus to search for question marks enter:

.... WHERE string LIKE '@?'

In addition, special characters lose their meaning (i.e. the escape character should not be used) when they appear out of context as follows:

% when not at the beginning of the pattern

$ when not at the end of the pattern

* at the beginning of the pattern

+ at the beginning of the pattern

! not at the beginning of a character class

- at the beginning or end of a character class

Special characters do not apply in the specification of a character class except for:

! at the beginning of the character class

- in the middle of the character class

@@ anywhere in the character class

(c) Ordering the result of a Query

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

Syntax:

The basic syntax of ORDER BY clause which would be used to sort result in ascending or descending order is as follows:

SELECT column-list

FROM table_name

[WHERE condition]

[ORDER BY column1, column2, .. columnN] [ASC | DESC];

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

Example:

Consider CUSTOMERS table is having following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example which would sort the result in ascending order by NAME and SALARY

```
SQL> SELECT * FROM CUSTOMERS
    ORDER BY NAME, SALARY;
```

This would produce following result:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
```

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

+----+----------+-----+-----------+----------+

Following is an example which would sort the result in descending order by NAME:

SQL> SELECT * FROM CUSTOMERS

ORDER BY NAME DESC;

This would produce following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

+----+----------+-----+-----------+----------+

To fetch the rows with own preferred order, the SELECT query would as follows:

```
SQL> SELECT * FROM CUSTOMERS
   ORDER BY (CASE ADDRESS
   WHEN 'DELHI'       THEN 1
   WHEN 'BHOPAL'   THEN 2
   WHEN 'KOTA'        THEN 3
   WHEN 'AHMADABAD' THEN 4
   WHEN 'MP'  THEN 5
   ELSE 100 END) ASC, ADDRESS DESC;
```

This would produce following result:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
+----+----------+-----+-----------+----------+
```

This will sort customers by ADDRESS in your ownoOrder of preference first and in a natural order for the remaining addresses. Also remaining Addresses will be sorted in the reverse alpha order.

(d) Aggregate Functions

AGGREGATE Functions :- In addition to simply retrieving data, we often want toperform some computation or summarization. We now consider a powerful class of constructs for computing aggregate values such as MIN and SUM. These features represent a significantextension of relational algebra. SQL supports five aggregate operations, which can be applied onany column, say A, of a relation:

1. COUNT (A) :- The number of values in the A column.
Or COUNT (DISTINCT A): The number of unique values in the A column.
Ex:- 1) To count number SIDs of sailors in Sailors table
SQL> SELECT **COUNT** (SID) FROM SAILORS;
2) To count numbers of boats booked in Reserves table.
SQL> SELECT **COUNT** (DISTINCT BID) FROM RESERVES;
3) To count number of Boats in Boats table.
SQL> SELECT **COUNT** (*) FROM BOATS;

2. SUM (A) :- The sum of all values in the A column.
Or SUM (DISTINCT A): The sum of all unique values in the A column.
Ex:- 1) To find sum of rating from Sailors
SQL> SELECT **SUM** (RATING) FROM SAILORS;
2) To find sum of distinct age of Sailors (Duplicate ages are eliminated).
SQL> SELECT **SUM** (DISTINCT AGE) FROM SAILORS;

3. AVG (A) :- The average of all values in the A column.
Or AVG (DISTINCT A): The average of all unique values in the A column.
Ex:- 1) To display average age of Sailors.
SQL> SELECT **AVG** (AGE) FROM SAILORS;
2) To find average of distinct age of Sailors (Duplicate ages are eliminated).
SQL> SELECT **AVG** (DISTINCT AGE) FROM SAILORS;

4. MAX (A) :- The maximum value in the A column.
Ex:- To find age of Oldest Sailor.
SQL> SELECT **MAX** (AGE) FROM SAILORS;

5. MIN (A) :- The minimum value in the A column.
Ex:- To find age of Youngest Sailor.
SQL> SELECT **MIN** (AGE) FROM SAILORS;

ORDER BY Clause :- The ORDER BY keyword is used to sort the result-set by a specified column. The ORDER BY keyword sorts the records in ascending order by default (we can even
use ASC keyword). If we want to sort the records in a descending order, we can use the DESC keyword. The general syntax is

SELECT ATT_LIST FROM TABLE_LIST ORDER BY ATT_NAMES [ASC | DESC];
Ex:-
1) Display all the sailors according to their ages.
SQL> SELECT * FROM SAILORS ORDER BY AGE;

2) Display all the sailors according to their ratings (topper first).
SQL> SELECT * FROM SAILORS ORDER BY RATING DESC;

3) Displays all the sailors according to rating, if rating is same then sort according to age.
SQL> SELECT * FROM SAILORS ORDER BY RATING, AGE;
Write the query
1) To display names of sailors according to alphabetical order.
2) Displays all the sailors according to rating (Topper First), if rating is same then sort according to age (Older First).
3) Displays all the sailors according to rating (Topper First), if rating is same then sort according to age (Younger First).

(e)Grouping the Result of a Query

GROUP BY and HAVING Clauses :- Thus far, we have applied aggregate operations toall (qualifying) rows in a relation. Often we want to apply aggregate operations to each of anumber of groups of rows in a relation, where the number of groups depends on the relationinstance. For this purpose we can use Group by clause.

GROUP BY:- Group by is used to make each a number of groups of rows in a relation, wherethe number of groups depends on the relation instances. The general syntax is

SELECT [DISTINCT] ATT_LIST FROM TABLE_LIST WHERE CONDITION **GROUP BY**
GROUPING_LIST;

Ex:- Find the age of the youngest sailor for each rating level.
SQL> SELECT S.RATING, MIN (S.AGE) FROM SAILORS S **GROUP BY** S.RATING;

HAVING :- The extension of GROUP BY is HAVING clause which can be used to specify the
qualification over group. The general syntax is

SELECT [DISTINCT] ATT_LIST FROM TABLE_LIST WHERE CONDITION GROUP BY GROUPING_LIST **HAVING** GROUP_CONDITIION;

Ex :- Find the age of youngest sailor with age >= 18 for each rating with at least 2 such sailors.
SQL> SELECT S.RATING, MIN (S.AGE) AS MINAGE FROM SAILORS S WHERE S.AGE
>= 18 GROUP BY S.RATING **HAVING** COUNT (*) > 1;

Write following queries in SQL.
1) For each red boat; find the number of reservations for this boat.
2) Find the average age of sailors for each rating level that has at least two sailors.
3) Find those ratings for which the average age of sailors is the minimum over all ratings.

(f)Update and Delete Statements :

(i) The SQL UPDATE Statement

The UPDATE statement is used to update existing records in a table.

SQL UPDATE Syntax

```
UPDATE table_name
SET column1=value1,column2=value2,...
WHERE some_column=some_value;
```

The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución 2222	México		
3	Antonio Moreno	Antonio	Mataderos	México	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover	London	WA1 1DP	UK
5	Berglunds	Christina Berglund	Berguvsvägen	S-98 22		Sweden

SQL UPDATE Example

Assume we wish to update the customer "Alfreds Futterkiste" with a new contact person and city.

We use the following SQL statement:

```
UPDATE Customers
SET ContactName='Alfred Schmidt', City='Hamburg'
WHERE CustomerName='Alfreds Futterkiste';
```

The selection from the "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkist	Alfred Schmidt	Obere Str. 57	Hamburg	12209	Germany

(ii) The SQL DELETE Statement

The DELETE statement is used to delete rows in a table.

SQL DELETE Syntax

```
DELETE FROM table_name
WHERE some_column=some_value;
```

To Delete All Data

It is possible to delete all rows in a table without deleting the table. This means that
the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name;
or
DELETE * FROM table_name
```

Exercise-4

PRACTICAL :- Set Operators, Nested Queries, Joins, Sequences

(a) Set Operators/Nested Queries :-

DISTINCT Keyword :- The DISTINCT keyword eliminates the duplicate tuples from the result records set.

Ex:- Find the Names and Ages of all sailors.
SQL> SELECT DISTINCT S.SNAME, S.AGE FROM SAILORS S;
The answer is a set of rows, each of which is a pair (sname, age). If two or more sailors have the
same name and age, the answer still contains just one pair with that name and age.

UNION, INTERSECT, EXCEPT (MINUS) :- SQL provides three set-manipulationconstructs that extend the basic query form. Since the answer to a query is a multiset of rows, itis natural to consider the use of operations such as union, intersection, and difference.SQL supports these operations under the names UNION, INTERSECT and MINUS.

UNION :- It is a set operator used as alternative to **OR** query.
Here is an example of Query using **OR.**

Ex:- Find the names of sailors who have reserved a red or a green boat.
SQL> SELECT S.SNAME FROM SAILORS S, RESERVES R, BOATS B WHERE S.SID = R.SID AND R.BID = B.BID AND (B.COLOR = 'RED' **OR** B.COLOR = 'GREEN');

Same query can be written using **UNION** as follows.
SQL> SELECT S.SNAME FROM SAILORS S, RESERVES R, BOATS B WHERE S.SID = R.SID AND R.BID = B.BID AND B.COLOR = 'RED' **UNION** SELECT S2.SNAME FROM SAILORS S2, BOATS B2, RESERVES R2 WHERE S2.SID = R2.SID AND R2.BID = B2.BID
AND B2.COLOR = 'GREEN';

This query says that we want the union of the set of sailors who have reserved red boatsand the set of sailors who have reserved green boats.

INTERSECT :- It is a set operator used as alternative to **AND** query. Here is an example of Query using **AND.**

Ex:- Find the names of sailor's who have reserved both a red and a green boat.

SQL> SELECT S.SNAME FROM SAILORS S, RESERVES R1, BOATS B1, RESERVES R2,
BOATS B2 WHERE S.SID = R1.SID AND R1.BID = B1.BID AND S.SID = R2.SID AND R2.BID = B2.BID AND B1.COLOR='RED' AND B2.COLOR = 'GREEN';

Same query can be written using **INTERSECT** as follows.
SQL> SELECT S.SNAME FROM SAILORS S, RESERVES R, BOATS B WHERE S.SID = R.SID AND R.BID = B.BID AND B.COLOR = ‘RED’ INTERSECT **SELECT** S2.SNAME FROM SAILORS S2, BOATS B2, RESERVES R2 WHERE S2.SID = R2.SID AND R2.BID =
B2.BID AND B2.COLOR = 'GREEN';

EXCEPT (MINUS) :- It is a set operator used as set-difference. Our next query illustrates the set-difference operation.

Ex:- Find the sids of all sailor's who have reserved red boats but not green boats.
SQL> SELECT S.SID FROM SAILORS S, RESERVES R, BOATS B WHERE S.SID = R.SID
AND R.BID = B.BID AND B.COLOR = 'RED' **MINUS** SELECT S2.SID FROM SAILORS S2,
RESERVES R2, BOATS B2 WHERE S2.SID = R2.SID AND R2.BID = B2.BID AND B2.COLOR = 'GREEN’;

Same query can be written as follows. Since the Reserves relation contains sid information, there
is no need to look at the Sailors relation, and we can use the following simpler query
SQL> SELECT R.SID FROM BOATS B, RESERVES R WHERE R.BID = B.BID AND B.COLOR = 'RED' **MINUS** SELECT R2.SID FROM BOATS B2, RESERVES R2 WHERE R2.BID = B2.BID AND B2.COLOR = ‘GREEN’;

NESTED QUERIES:- For retrieving data from the tables we have seen the simple & basicqueries. These queries extract the data from one or more tables. Here we are going to see somecomplex & powerful queries that enables us to retrieve the data in desired manner. One of themost powerful features of SQL is nested queries. A nested query is a query that has anotherquery embedded within it; the embedded query is called a subquery.

IN Operator :- The IN operator allows us to test whether a value is in a given set of elements; an SQL query is used to generate the set to be tested.
Ex:- Find the names of sailors who have reserved boat 103.
SQL> SELECT S.SNAME FROM SAILORS S WHERE S.SID **IN** (SELECT R.SID FROM RESERVES R WHERE R.BID = 103);

NOT IN Operator :- The NOT IN is used in a opposite manner to IN.
Ex:- Find the names of sailors who have not reserved boat 103.
SQL> SELECT S.SNAME FROM SAILORS S WHERE S.SID **NOT IN** (SELECT R.SID

FROM RESERVES R WHERE R.BID = 103);

EXISTS Operator :- This is a Correlated Nested Queries operator. The EXISTS operator is another set comparison operator, such as IN. It allows us to test whether a set is nonempty, an implicit comparison with the empty set.

Ex:- Find the names of sailors who have reserved boat number 103.
SQL> SELECT S.SNAME FROM SAILORS S WHERE **EXISTS** (SELECT * FROM RESERVES R WHERE R.BID = 103 AND R.SID = S.SID);

NOT EXISTS Operator :- The NOT EXISTS is used in a opposite manner to EXISTS.
Ex:- Find the names of sailors who have not reserved boat number 103.
SQL> SELECT S.SNAME FROM SAILORS S WHERE **NOT EXISTS** (SELECT * FROM RESERVES R WHERE R.BID = 103 AND R.SID = S.SID);

Set-Comparison Operators:- We have already seen the set-comparison operators EXISTS, IN along with their negated versions. SQL also supports **op ANY** and **op ALL**, where **op** is one
of the arithmetic comparison operators {<, <=, =, <>, >=, >}. Following are the example which illustrates the use of these Set-Comparison Operators.

op ANY Operator :- It is a comparison operator. It is used to compare a value with any of element in a given set.
Ex:- Find sailors whose rating is better than some sailor called Rajesh.
SQL> SELECT S.SID FROM SAILORS S WHERE S.RATING > **ANY** (SELECT S2.RATING
FROM SAILORS S2 WHERE S2.SNAME = ' RAJESH ');
Note that IN and NOT IN are equivalent to = ANY and <> ALL, respectively.

op ALL Operator :- It is a comparison operator. It is used to compare a value with all the elements in a given set.
Ex:- Find the sailor's with the highest rating using ALL.
SQL> SELECT S.SID FROM SAILORS S WHERE S.RATING >= **ALL** (SELECT S2.RATING FROM SAILORS S2)

(b) SQL JOINS:-

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

The most common type of join is: SQL INNER JOIN (simple join). An SQL INNER JOIN return all rows from multiple tables where the join condition is met.

Let's look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, have a look at a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo	Ana Trujillo	Mexico
3	Antonio Moreno	Antonio Moreno	Mexico

Notice that the "CustomerID" column in the "Orders" table refers to the customer in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, if we run the following SQL statement (that contains an INNER JOIN):

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

Different SQL JOINs

INNER JOIN: Returns all rows when there is at least one match in BOTH tables
LEFT JOIN: Return all rows from the left table, and the matched rows from the right table
RIGHT JOIN: Return all rows from the right table, and the matched rows from the left table
FULL JOIN: Return all rows when there is a match in ONE of the tables.

(c)Sequences:-

A sequence is a database object that generates numbers in sequential order. Applications most often use these numbers when they require a unique value in a table such as primary key values. Some database management systems use an "auto number" concept or "auto increment" setting on numeric column types. Both the auto numbering columns and sequences provide a unique number in sequence used for a unique identifier. The following list describes the characteristics of sequences:

- Sequences are available to all users of the database
- Sequences are created using SQL statements (see below)

- Sequences have a minimum and maximum value (the defaults are minimum=0 and maximum=2^{63}-1); they can be dropped, but not reset
- Once a sequence returns a value, the sequence can never return that same value
- While sequence values are not tied to any particular table, a sequence is usually used to generate values for only one table
- Sequences increment by an amount specified when created (the default is 1)

Creating a Sequence

To create sequences, execute a CREATE SEQUENCE statement in the same way as an UPDATE or INSERT statement. The sequence information is stored in a data dictionary file in the same location as the rest of the data dictionary files for the database. If the data dictionary file does not exist, the SQL engine creates the file when it creates the first sequence. In legacy dictionaries, the new file name is SEQUENCE.DD. The format of this file remains proprietary and subject to change, so do not depend on the record layout or format of the data. In Journaled Filesystem databases, this information is also proprietary and subject to change.

The format for a CREATE SEQUENCE statement is as follows:

CREATE SEQUENCE sequence_name

[INCREMENT BY #]

[START WITH #]

[MAXVALUE # | NOMAXVALUE]

[MINVALUE # | NOMINVALUE]

[CYCLE | NOCYCLE]

Variable	Description
INCREMENT BY	The increment value. This can be a positive or negative number.
START WITH	The start value for the sequence.
MAXVALUE	The maximum value that the sequence can generate. If specifying NOMAXVALUE, the maximum value is 2^{63}-1.
MINVALUE	The minimum value that the sequence can generate. If specifying NOMINVALUE, the minimum value is -2^{63}.
CYCLE	Specify CYCLE to indicate that when the maximum value is reached the sequence starts over again at the start value. Specify NOCYCLE to generate an error upon reaching the maximum value.

Dropping a Sequence

To drop a sequence, execute a DROP SEQUENCE statement. Use this function when a sequence is no longer useful, or to reset a sequence to an older number. To reset a sequence, first drop the sequence and then recreate it.

Drop a sequence following this format:

DROP SEQUENCE my_sequence

Using a Sequence

Use sequences when an application requires a unique identifier. INSERT statements, and occasionally UPDATE statements, are the most common places to use sequences. Two "functions" are available on sequences:

NEXTVAL: Returns the next value from the sequence.

CURVAL: Returns the value from the last call to NEXTVAL by the current user during the current connection. For example, if User A calls NEXTVAL and it returns 124, and User B immediately calls NEXTVAL getting 125, User A will get 124 when calling CURVAL, while User B will get 125 while calling CURVAL. It is important to understand the connection between the sequence value and a particular connection to the database. The user cannot call CURVAL until making a call to NEXTVAL at least once on the connection. CURVAL returns the current value returned from the sequence on the current connection, not the current value of the sequence.

Examples

- To create the sequence:
  CREATE SEQUENCE customer_seq INCREMENT BY 1 START WITH 100

- To use the sequence to enter a record into the database:
  INSERT INTO customer (cust_num, name, address)
  VALUES (customer_seq.NEXTVAL, 'John Doe', '123 Main St.')

- To find the value just entered into the database:
  SELECT customer_seq.CURVAL AS LAST_CUST_NUM

Exercise- 5

PRACTICAL :- Views, Indexes, Databases Security, and Privileges: Grant and Revoke Commands and Rollback Commands.

VIEWS :- A view is a table whose rows are not explicitly stored in the database but are computed as needed from a view definition. The views are created using **CREATE VIEW** command.

Ex :- Create a view for Expert Sailors (A sailor is a Expert Sailor if his rating is more than 7).
SQL> CREATE VIEW EXPERTSAILOR AS SELECT SID, SNAME, RATING FROM SAILORS WHERE RATING > 7;
Now on this view we can use normal SQL statements as we are using on Base tables.
Eg:- Find average age of Expert sailors.

SQL> SELECT AVG (AGE) FROM EXPERTSAILOR;

Write the following queries on Expert Sailor View.
1) Find the Sailors with age > 25 and rating equal to 10.
2) Find the total number of Sailors in Expert Sailor view.
3) Find the number of Sailors at each rating level (8, 9, 10).
4) Find the sum of rating of Sailors.
5) Find the age of Oldest as well as Youngest Expert Sailor.

If we decide that we no longer need a view and want to destroy it (i.e. removing the definition of
view) we can drop the view. A view can be dropped using the DROP VIEW command. To drop the ExpertSailor view.

SQL> DROP VIEW EXPERTSAILOR;

INDEXES:-

The CREATE INDEX statement is used to create indexes in tables.

Indexes allow the database application to find data fast; without reading the whole table.

An index can be created in a table to find data more quickly and efficiently.The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So you should only create indexes on columns (and tables) that will be frequently searched against.

SQL CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index_name

ON table_name (column_name)

SQL CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index_name

ON table_name (column_name)

Note: The syntax for creating indexes varies amongst different databases. Therefore: Check the syntax for creating indexes in your database.

CREATE INDEX Example

The SQL statement below creates an index named "PIndex" on the "LastName" column in the "Persons" table:

CREATE INDEX PIndex

ON Persons (LastName)

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

CREATE INDEX PIndex

ON Persons (LastName, FirstName)

Database Security and Privileges: Grant , Revoke and Roll Back Commands

SQL GRANT Command

SQL GRANT is a command used to provide access or privileges on the database objects to the users.

The Syntax for the GRANT command is:

GRANT privilege_name

ON object_name

TO {user_name |PUBLIC |role_name}

[WITH GRANT OPTION];

privilege_name is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.

object_name is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.

user_name is the name of the user to whom an access right is being granted.

user_name is the name of the user to whom an access right is being granted.

PUBLIC is used to grant access rights to all users.

ROLES are a set of privileges grouped together.

WITH GRANT OPTION - allows a user to grant access rights to other users.

For Example: GRANT SELECT ON employee TO user1;This command grants a SELECT permission on employee table to user1.You should use the WITH GRANT option carefully because for example if you GRANT SELECT privilege on employee table to user1 using the WITH GRANT option, then user1 can GRANT SELECT privilege on employee table to another user, such as user2 etc. Later, if you REVOKE the SELECT privilege on employee from user1, still user2 will have SELECT privilege on employee table.

SQL REVOKE Command:

The REVOKE command removes user access rights or privileges to the database objects.

The Syntax for the REVOKE command is:

REVOKE privilege_name

ON object_name

FROM {user_name |PUBLIC |role_name}

For Example: REVOKE SELECT ON employee FROM user1;This command will REVOKE a SELECT privilege on employee table from user1.When you REVOKE

SELECT privilege on a table from a user, the user will not be able to SELECT data from that table anymore. However, if the user has received SELECT privileges on that table from more than one users, he/she can SELECT from that table until everyone who granted the permission revokes it. You cannot REVOKE privileges if they were not initially granted by you.

The ROLLBACK Command:

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.

The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for ROLLBACK command is as follows:

ROLLBACK;

Exercise-6

PRACTICAL :- PL/SQL Architecture,Assignments and Expressions, Writing PL/SQL Code, Referencing Non- SQL Parameters.

PL/SQL Architecture

PL/SQL, the Oracle procedural extension of SQL, is a completely portable, high-performance transaction-processing language. PL/SQL combines the data-manipulating power of SQL with the processing power of procedural languages.

When a problem can be solved using SQL, you can issue SQL statements from your PL/SQL programs, without learning new APIs.

Like other procedural programming languages, PL/SQL lets you declare constants and variables, control program flow, define subprograms, and trap run-time errors.

Main Features of PL/SQL

- PL/SQL Blocks
- PL/SQL Error Handling
- PL/SQL Input and Output
- PL/SQL Variables and Constants
- PL/SQL Data Abstraction
- PL/SQL Control Structures
- PL/SQL Subprograms
- PL/SQL Packages (APIs Written in PL/SQL)
- Conditional Compilation
- Embedded SQL Statements

Advantages of PL/SQL

- Tight Integration with SQL
- High Performance
- High Productivity
- Full Portability
- Tight Security
- Access to Predefined Packages
- Support for Object-Oriented Programming
- Support for Developing Web Applications and Server Pages

Architecture of PL/SQL

PL/SQL Engine

This is a separator, which splits the SQL and PL/SQL statements.PL/SQL block comprises of SQL &PL commands are submitted as one request.PL/SQL engine split the block and sends appropriated commands to the corresponding units. SQL evaluator will handle SQL statements(available on the server side).PL Evaluator handles PL commands. Engine can reside either in client side or in the server side. The PL/SQL compilation and run-time system is an engine that compiles and executes PL/SQL units. The engine can be installed in the database or in an application development tool, such as Oracle Forms.

In either environment, the PL/SQL engine accepts as input any valid PL/SQL unit. The engine executes procedural statements, but sends SQL statements to the SQL engine in the database.

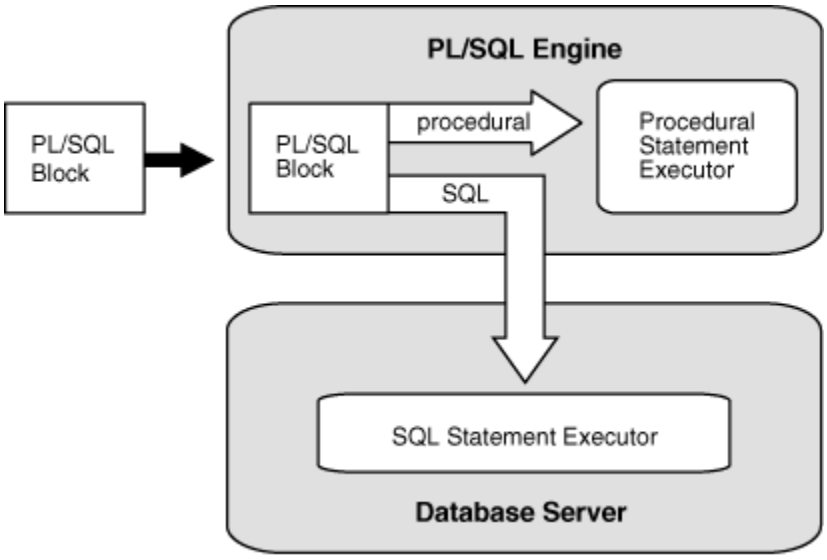


Fig. PL/SQL Engine processing an anonymous block.

Typically, the database processes PL/SQL units. When an application development tool processes PL/SQL units, it passes them to its local PL/SQL engine. If a PL/SQL unit contains no SQL statements, the local engine processes the entire PL/SQL unit. This is useful if the application development tool can benefit from conditional and iterative control.

For example, Oracle Forms applications frequently use SQL statements to test the values of field entries and do simple computations. By using PL/SQL instead of SQL, these applications can avoid calls to the database.

PL/SQL PROGRAMMING
Procedural Language/Structured Query Language (PL/SQL) is an extension of SQL.

Basic Syntax of PL/SQL

```
DECLARE
/* Variables can be declared here */
BEGIN
/* Executable statements can be written here */
EXCEPTION
/* Error handlers can be written here. */
END;
```

Steps to Write & Execute PL/SQL

- As we want output of PL/SQL Program on screen, before Starting writing anything type (Only Once per session)

```
SQL> SET SERVEROUTPUT ON
```

- To write program, use Notepad through Oracle using ED command.

```
SQL> ED ProName
```

Type the program Save & Exit.

- To Run the program

```
SQL> @ProName
```

Ex :- PL/SQL to find addition of two numbers

```
DECLARE
A INTEGER := &A;
B INTEGER := &B;
C INTEGER;
BEGIN
C := A + B;
DBMS_OUTPUT.PUT_LINE('THE SUM IS '||C);
END;
/
```

Decision making with IF statement :- The general syntax for the using IF--ELSE statement is

```
IF(TEST_CONDITION) THEN
```

SET OF STATEMENTS
ELSE
SET OF STATEMENTS
END IF;

For Nested IF—ELSE Statement we can use IF--ELSIF—ELSE as follows

```
IF(TEST_CONDITION) THEN
SET OF STATEMENTS
ELSIF (CONDITION)
SET OF STATEMENTS
END IF;
```

Ex:- Largest of three numbers.

This program can be written in number of ways, here are the two different ways to write the program.

1)

```
DECLARE
A NUMBER := &A;
B NUMBER := &B;
C NUMBER := &C;
BIG NUMBER;
BEGIN
IF (A > B) THEN
BIG := A;
ELSE
BIG := B;
END IF;
IF(BIG < C ) THEN
DBMS_OUTPUT.PUT_LINE('BIGGEST OF A, B AND C IS ' || C);
ELSE
DBMS_OUTPUT.PUT_LINE('BIGGEST OF A, B AND C IS ' || BIG);
END IF;
END;
/
```

2)

```
DECLARE
A NUMBER := &A;
B NUMBER := &B;
C NUMBER := &C;
BEGIN
IF (A > B AND A > C) THEN
DBMS_OUTPUT.PUT_LINE('BIGGEST IS ' || A);
ELSIF (B > C) THEN
DBMS_OUTPUT.PUT_LINE('BIGGEST IS ' || B);
ELSE
DBMS_OUTPUT.PUT_LINE('BIGGEST IS ' || C);
END IF;
```

END;
/

LOOPING STATEMENTS:- For executing the set of statements repeatedly we can use loops. The oracle supports number of looping statements like GOTO, FOR, WHILE & LOOP. Here is the syntax of these all the types of looping statements.

• **GOTO STATEMENTS**
<<LABEL>>
SET OF STATEMENTS
GOTO LABEL;
• **FOR LOOP**
FOR <VAR> IN [REVERSE] <INI_VALUE>**..**<END_VALUE>
SET OF STATEMENTS
END LOOP;
• **WHILE LOOP**
WHILE (CONDITION) LOOP
SET OF STATEMENTS
END LOOP;
• **LOOP STATEMENT**
LOOP
SET OF STATEMENTS
IF (CONDITION) THEN
EXIT
SET OF STATEMENTS
END LOOP;
While using LOOP statement, we have take care of EXIT condition, otherwise it may go into infinite loop.
Example :- Here are the example for all these types of looping statement where each program prints numbers 1 to 10.
GOTO EXAMPLE
DECLARE
I INTEGER := 1;
BEGIN
<<OUTPUT>>
DBMS_OUTPUT.PUT_LINE(I);
I := I + 1;
IF I<=10 THEN
GOTO OUTPUT;
END IF;
END;
/
FOR LOOP EXAMPLE
BEGIN
FOR I IN 1..10 LOOP

```
DBMS_OUTPUT.PUT_LINE(I);
END LOOP;
END;
/
```

WHILE EXAMPLE

```
DECLARE
I INTEGER := 1;
BEGIN
WHILE(I<=10) LOOP
DBMS_OUTPUT.PUT_LINE(I);
I := I + 1;
END LOOP;
END;
/
```

LOOP EXAMPLE

```
DECLARE
I INTEGER := 1;
BEGIN
LOOP
DBMS_OUTPUT.PUT_LINE(I);
I := I + 1;
EXIT WHEN I=11;
END LOOP;
END;
/
```

DATA TYPES

Already we know following data types.
NUMBER, INTEGER, VARCHAR2, DATE, BOOLEAN, etc. Now let's see few more data types that are useful for writing PL/SQL programs in Oracle.

• **%TYPE :-** %TYPE is used to give data type of predefined variable or database column.
Eg:- itemcode Number(10);
icode itemcode%Type;
The database column can be used as
id Sailors.sid%type

• **%ROWTYPE :-** %rowtype is used to provide record datatype to a variable. The variable can store row of the table or row fetched from the cursor.
• **Eg:-** If we want to store a row of table Sailors then we can declare variable as
Sailors %Rowtype

Comments :- In Oracle we can have two types of comments i.e Single Line & Multiline comments.
• Single line comment :- It starts with --.

```
-- Comment here
```

• Multiline comment is same as C/C++/JAVA comments where comments are present in the pair of /* & */.
/* Comment here */

Inserting values to table :- Here is the example for inserting the values into a database through PL/SQL Program. Remember that we have to follow all the rules of SQL like Primary Key Constraints, Foreign Key Constraints, Check Constraints, etc.
Ex:- Insert the record into Sailors table by reading the values from the Keyboard.

```
DECLARE
SID NUMBER (5):=&SID;
SNAME VARCHAR2(30):='&SNAME';
RATING NUMBER(5):=&RATING;
AGE NUMBER(4,2):=&AGE;
BEGIN
INSERT INTO SAILORS VALUES(SID, SNAME, RATING, AGE);
END;
/
```

Reading from table

```
DECLARE
SID VARCHAR2(10); -- or can be defined SID Sailors.SID%Type
SNAME VARCHAR2(30);
RATING NUMBER(5);
AGE NUMBER(4,2);
BEGIN
SELECT SID, SNAME, RATING, AGE INTO SID, SNAME, RATING, AGE FROM
SAILORS WHERE SID='&SID';
DBMS_OUTPUT.PUT_LINE(SID || ' '|| SNAME || ' '|| RATING ||' '|| AGE );
END;
/
```

Some Points regarding SELECT --- INTO

We have to ensure that the SELECT….INTO statement should return one & only one row. If no row is selected then exception NO_DATA_FOUND is raised. If more than one row is
selected then exception TOO_MANY_ROWS is raised.
To handle the situation where no rows selected or so many rows selected we can use Exceptions. We have two types of exception, User-Defined and Pre-Defined Exceptions.

Program with User-Defined Exception

```
DECLARE
N INTEGER:=&N;
A EXCEPTION;
B EXCEPTION;
BEGIN
IF MOD(N,2)=0 THEN
RAISE A;
ELSE
```

```
RAISE B;
END IF;
EXCEPTION
WHEN A THEN
DBMS_OUTPUT.PUT_LINE('THE INPUT IS EVEN.....');
WHEN B THEN
DBMS_OUTPUT.PUT_LINE('THE INPUT IS ODD.....');
END;
/
```

Program with Pre-Defined Exception

```
DECLARE
SID VARCHAR2(10);
BEGIN
SELECT SID INTO SID FROM SAILORS WHERE SNAME='&SNAME';
DBMS_OUTPUT.PUT_LINE(SID);
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('No Sailors with given SID found');
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE('More than one Sailors with same name found');
END;
/
```

Example Programs:-

1. Write a program to generate all prime numbers below 100.

PRACTICAL : to generate all prime numbers below 100.

PlSQL

```
Declare
        I number;
        J number;
        C number;
Begin
While(i<=100)
Loop
```

```
        C:=0;
        J:=1;

        While(j<=i)

        Loop

                If(floor(i%j)=0) then

                        C:= C+1;

                End if;

        J:=j+1;

        End loop;

If(c=2) then

        Dbms_output.put_line(i);

End if;

Endloop;
End;
```

Valid Test Data

OUTPUT:

2

3

5

7

11

.

.

99

2. Program to check whether given number is Armstrong or not.

PRACTICAL : to check whether given number is Armstrong or not.

Pl-SQL

Algorithm:

Step 1: Declare the variable N, S, D and DUP.
Step 2: Store the value in var. N and var. DUP..
Step 3: check for the value of N, which is not equal to 0.
Step 4: divide value stored in N by 10 and store it var. D. (D=n%10).
Step 5: the reminder will be multiply 3 times and store it in Var. S.
Step 6: The coefficient will be calculated using FLOOR function. And store it in var. N.
Step 7: repeat the Steps 3, 4, 5, and 6 till loop will be terminated.
Step 8: Check whether the stored value and calculated values are same
Step 9: if both the values are same, then display "The given number is Armstrong"
Step 10: Otherwise display "it is not Armstrong" and terminate the loop.

```
Declare

        N number;
        S number;
        D number;

Begin

        N:=&n;

        S:=0;

While(n!=0)

Loop

        D=n%10;
        S:=s+(D*D*D);
        N:=floor(n/10);

End loop;

If (DUP=S) then

        DBMS_output.put_line('number is armstrong');
```

```
Else
    DBMS_output.put_line('number is not armstrong');
End if;
End;
```

Test Valid Data Set:

Enter value of n

153

OUTPUT:

number is Armstrong.

---

3). Write a program to find largest number from the given three numbers.

PRACTICAL : To find largest number from the given three numbers.

PlSQL

Algorithm:

Step 1: Declare the variable A, B, and C.
Step 2: Store the valid data.
Step 3: Compare variable A with B and A with C
Step 4: If the value stored in variable A is big, it displays "A is Big". (IF conditional statement should be used)
Step 5: Compare variable B with C
Step 6: If the value stored in variable B is big, it displays "B is Big".
Step 7: other wise it displays "C is Big"

```
Declare
    A number;
    B number;
    C number;
Begin
```

```
        A:=&a;
        B:=&b;
        C:=&c;
If a > b && a> c then
        Dbms_output.put_line(‘ A is big ‘);
Else
If( b>c && b> a ) then
        Dbms_output.put_line(‘ B is big ‘);
Else
        Dbms_output.put_line(‘ C is big ‘);
End if;
End if;
End;
```

Valid Data Sets:

```
Enter the value of a:
1
Enter the value of b:
2
Enter the value of c:
3
```

OUTPUT:

C is big

Exercise-7

PRACTICAL :- Stored Procedures and Exception Handling.

Stored Procedures:- A **stored procedure** or in simple a **proc** is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages.

A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block.

A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

We can pass parameters to procedures in three ways.

1) IN-parameters
2) OUT-parameters
3) IN OUT-parameters

A procedure may or may not return any value.

General Syntax to create a procedure is:

```
CREATE [OR REPLACE] PROCEDURE proc_name [list of parameters]
IS
   Declaration section
BEGIN
   Execution section
EXCEPTION
  Exception section
END;
```

IS - marks the beginning of the body of the procedure and is similar to DECLARE in anonymous PL/SQL Blocks. The code between IS and BEGIN forms the Declaration section.

The syntax within the brackets [ ] indicate they are optional. By using CREATE OR REPLACE together the procedure is created if no other procedure with the same name exists or the existing procedure is replaced with the current code.

The below example creates a procedure ‘employer_details’ which gives the details of the employee.

```
1> CREATE OR REPLACE PROCEDURE employer_details
2> IS
3>  CURSOR emp_cur IS
4>  SELECT first_name, last_name, salary FROM emp_tbl;
5>  emp_rec emp_cur%rowtype;
6> BEGIN
7>  FOR emp_rec in sales_cur
8>  LOOP
9>  dbms_output.put_line(emp_cur.first_name || ' ' ||emp_cur.last_name
10>    || ' ' ||emp_cur.salary);
11> END LOOP;
12>END;
13> /
```

How to execute a Stored Procedure?
There are two ways to execute a procedure.
1) From the SQL prompt.
EXECUTE [or EXEC] procedure_name;
2) Within another procedure – simply use the procedure name.
procedure_name;

Exercise

1)Create Stored procedure to print out a "Hello World" via DBMS_OUTPUT.

```
CREATEORREPLACEPROCEDURE procPrintHelloWorld
IS
BEGIN

  DBMS_OUTPUT.PUT_LINE('Hello World!');

END;
/
```

Run it

```
EXEC procPrintHelloWorld;
```

Output
Hello World!
2) Create simple stored procedure in oracle with passing parameter and creating select statement with where condition and with if..else clause.

```
SQL>CREATE OR REPLACE PROCEDURE AddNewEmployee (
p_FirstName employee.first_name%TYPE,
p_LastName employee.last_name%TYPE,
p_Salary employee.salary%TYPE) AS
BEGIN INSERT INTO employee (ID, first_name, last_name,
salary)
VALUES (student_sequence.nextval, p_FirstName, p_LastName,
p_Salary);
COMMIT;
END AddNewEmployee;
/
```

3) Create Oracle stored procedure for updating two tables in a single go.

```
Createor replace Procedure update_select AS
BEGIN
update
(select a.col_to_update as c1, b.col_to_update as c2, c.value_to_get v1
from a join b on(join conditions)
join c on(join conditions)
where conditions)
set
  c1 = v1, c2 = v2;
END;
```

(b) Introduction of Exception handling and Example

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly.
When an exception occurs messages which explains its cause is received. PL/SQL Exception message consists of three parts.

1) Type of Exception
2) An Error Code
3) A message
By Handling the exceptions we can ensure a PL/SQL block does not exit abruptly.
1) Structure of Exception Handling.
General Syntax for coding the exception section

```
DECLARE
  Declaration section
BEGIN
  Exception section
EXCEPTION
WHEN ex_name1 THEN
   -Error handling statements
WHEN ex_name2 THEN
   -Error handling statements
WHEN Others THEN
  -Error handling statements
END;
```

General PL/SQL statments can be used in the Exception Block.
When an exception is raised, Oracle searches for an appropriate exception handler in the exception section. For example in the above example, if the error raised is 'ex_name1 ', then the error is handled according to the statements under it. Since, it is not possible to determine all the possible runtime errors during testing fo the code, the 'WHEN Others' exception is used to manage the exceptions that are not explicitly handled. Only one exception can be raised in a Block and the control does not return to the Execution Section after the error is handled.
If there are nested PL/SQL blocks like this.

```
DELCARE
  Declaration section
BEGIN
   DECLARE
     Declaration section
   BEGIN
     Execution section
   EXCEPTION
     Exception section
   END;
EXCEPTION
```

Exception section
END;

In the above case, if the exception is raised in the inner block it should be handled in the exception block of the inner PL/SQL block else the control moves to the Exception block of the next upper PL/SQL Block. If none of the blocks handle the exception the program ends abruptly with an error.

2) Types of Exception.

There are 3 types of Exceptions.

a) Named System Exceptions

b) Unnamed System Exceptions

c) User-defined Exceptions

a) Named System Exceptions

System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule. There are some system exceptions which are raised frequently, so they are pre-defined and given a name in Oracle which are known as Named System Exceptions.

For example: NO_DATA_FOUND and ZERO_DIVIDE are called Named System exceptions.

Named system exceptions are:

1) Not Declared explicitly,

2) Raised implicitly when a predefined Oracle error occurs,

3) caught by referencing the standard name within an exception-handling routine.

Exception Name	**Reason**	**Error Number**
CURSOR_ALREADY_OPEN	When you open a cursor that is already open.	ORA-06511
INVALID_CURSOR	When you perform an invalid operation on a cursor like closing a cursor, fetch data from a cursor that is not opened.	ORA-01001
NO_DATA_FOUND	When a SELECT...INTO clause does not return any row from a table.	ORA-01403
TOO_MANY_ROWS	When you SELECT or fetch more than one row into a record or variable.	ORA-01422
ZERO_DIVIDE	When you attempt to divide a number by zero.	ORA-01476

For Example: Suppose a NO_DATA_FOUND exception is raised in a proc, we can write a code to handle the exception as given below.

```
BEGIN
  Execution section
EXCEPTION
WHEN NO_DATA_FOUND THEN
 dbms_output.put_line ('A SELECT...INTO did not return any row.');
 END;
```

b) Unnamed System Exceptions

Those system exception for which oracle does not provide a name is known as unamed system exception. These exception do not occur frequently. These Exceptions have a code and an associated message.

There are two ways to handle unnamed sysyem exceptions:

1. By using the WHEN OTHERS exception handler, or
2. By associating the exception code to a name and using it as a named exception.

We can assign a name to unnamed system exceptions using a **Pragma** called **EXCEPTION_INIT.**

EXCEPTION_INIT will associate a predefined Oracle error number to a programmer_defined exception name.

Steps to be followed to use unnamed system exceptions are

• They are raised implicitly.

• If they are not handled in WHEN Others they must be handled explicity.

• To handle the exception explicity, they must be declared using Pragma EXCEPTION_INIT as given above and handled referecing the user-defined exception name in the exception section.

The general syntax to declare unnamed system exception using EXCEPTION_INIT is:

```
DECLARE
   exception_name EXCEPTION;
   PRAGMA
   EXCEPTION_INIT (exception_name, Err_code);
BEGIN
Execution section
EXCEPTION
  WHEN exception_name THEN
     handle the exception
END;
```

For Example: Lets consider the product table and order_items table from sql joins.

Here product_id is a primary key in product table and a foreign key in order_items table. If we try to delete a product_id from the product table when it has child records in order_id table an exception will be thrown with oracle code number -2292.

We can provide a name to this exception and handle it in the exception section as given below.

```
 DECLARE
  Child_rec_exception EXCEPTION;
  PRAGMA
   EXCEPTION_INIT (Child_rec_exception, -2292);
BEGIN
  Delete FROM product where product_id= 104;
EXCEPTION
   WHEN Child_rec_exception
   THEN Dbms_output.put_line('Child records are present for this product_id.');
END;
/
```

c) User-defined Exceptions

Apart from sytem exceptions we can explicity define exceptions based on business rules. These are known as user-defined exceptions.

Steps to be followed to use user-defined exceptions:

- They should be explicitly declared in the declaration section.
- They should be explicitly raised in the Execution Section.
- They should be handled by referencing the user-defined exception name in the exception section.

For Example: Lets consider the product table and order_items table from sql joins to explain user-defined exception. Lets create a business rule that if the total no of units of any particular product sold is more than 20, then it is a huge quantity and a special discount should be provided.

```
DECLARE
  huge_quantity EXCEPTION;
  CURSOR product_quantity is
  SELECT p.product_name as name, sum(o.total_units) as units
  FROM order_tems o, product p
  WHERE o.product_id = p.product_id;
  quantity order_tems.total_units%type;
  up_limit CONSTANT order_tems.total_units%type := 20;
  message VARCHAR2(50);
BEGIN
  FOR product_rec in product_quantity LOOP
    quantity := product_rec.units;
     IF quantity > up_limit THEN
      message := 'The number of units of product ' || product_rec.name ||
                ' is more than 20. Special discounts should be provided.
                     Rest of the records are skipped. '
     RAISE huge_quantity;
     ELSIF quantity < up_limit THEN
      v_message:= 'The number of unit is below the discount limit.';
     END IF;
     dbms_output.put_line (message);
  END LOOP;
 EXCEPTION
   WHEN huge_quantity THEN
     dbms_output.put_line (message);
 END;
/
```

RAISE_APPLICATION_ERROR ( )

RAISE_APPLICATION_ERROR is a built-in procedure in oracle which is used to display the user-defined error messages along with the error number whose range is in between -20000 and -20999.

Whenever a message is displayed using RAISE_APPLICATION_ERROR, all previous transactions which are not committed within the PL/SQL Block are rolled back automatically (i.e. change due to INSERT, UPDATE, or DELETE statements).

RAISE_APPLICATION_ERROR raises an exception but does not handle it.
RAISE_APPLICATION_ERROR is used for the following reasons,
a) to create a unique id for an user-defined exception.
b) to make the user-defined exception look like an Oracle error.
The General Syntax to use this procedure is:
RAISE_APPLICATION_ERROR (error_number, error_message);

• The Error number must be between -20000 and -20999
• The Error_message is the message you want to display when the error occurs.
Steps to be folowed to use RAISE_APPLICATION_ERROR procedure:
1. Declare a user-defined exception in the declaration section.
2. Raise the user-defined exception based on a specific business rule in the execution section.
3. Finally, catch the exception and link the exception to a user-defined error number in RAISE_APPLICATION_ERROR.
Using the above example we can display a error message using RAISE_APPLICATION_ERROR.

```
DECLARE
  huge_quantity EXCEPTION;
  CURSOR product_quantity is
  SELECT p.product_name as name, sum(o.total_units) as units
  FROM order_tems o, product p
  WHERE o.product_id = p.product_id;
  quantity order_tems.total_units%type;
  up_limit CONSTANT order_tems.total_units%type := 20;
  message VARCHAR2(50);
BEGIN
  FOR product_rec in product_quantity LOOP
    quantity := product_rec.units;
     IF quantity > up_limit THEN
        RAISE huge_quantity;
     ELSIF quantity < up_limit THEN
      v_message:= 'The number of unit is below the discount limit.';
     END IF;
     Dbms_output.put_line (message);
  END LOOP;
 EXCEPTION
   WHEN huge_quantity THEN
         raise_application_error(-2100, 'The number of unit is above the discount limit.');
 END;
/
```

Example:-Write a program to demonstrate predefined exceptions

PRACTICAL : to demonstrate predefined exceptions

PlSQL

```
Declare
        A number
        B number;
        C number;
Begin
        A:=&a;
        B:=&b;
        C:=a/b;
        Dbms_output.put_line('division is ' || C);
Exception
        If (ZERO_DIVIDE) then
        Dbms_output.put_line('b could not be zero');
        End if;
End;
```

Valid Test Data:

Enter the value for a:

10

Enter the value for b:

0

OUTPUT:

b could not be zero

Example:-Write a program to demonstrate user defined exceptions

PRACTICAL : to demonstrate user defined exceptions

```
PlSQL
Declare
        A number
        B number;
        C number;
        Mydivide_zero EXCEPTION;
Begin
        A:=&a;
        B:=&b;
        If(B=0) then
        Raise Mydivide_zero;
        else
        C:=a/b;
        Dbms_output.put_line('division is ' || C);
        End if;
Exception
        If (mydivide_zero) then
        Dbms_output.put_line('b could not be zero');
        End if;
End;
```

Valid Test Data:

Enter the value for a:

10

Enter the value for b:

0

OUTPUT:

b could not be zero

Exercise-8

PRACTICAL :- Triggers and Cursor Management in PL/SQL.

Triggers:- A trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

Syntax for Creating a Trigger

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
BEGIN
  --- sql statements
END;
```

- *CREATE [OR REPLACE] TRIGGER trigger_name* - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
- *{BEFORE | AFTER | INSTEAD OF }* - This clause indicates at what time should the trigger get fired. i.e for example: before or after updating a table. INSTEAD OF is used to create a trigger on a view. before and after cannot be used to create a trigger on a view.
- *{INSERT [OR] | UPDATE [OR] | DELETE}* - This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.
- *[OF col_name]* - This clause is used with update triggers. This clause is used when you want to trigger an event only when a specific column is updated.
- *CREATE [OR REPLACE] TRIGGER trigger_name* - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
- *[ON table_name]* - This clause identifies the name of the table or view to which the trigger is associated.
- *[REFERENCING OLD AS o NEW AS n]* - This clause is used to reference the old and new values of the data being changed. By default, you reference the values as :old.column_name or :new.column_name. The reference names can also be changed from old (or new) to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.

- *[FOR EACH ROW]* - This clause is used to determine whether a trigger must fire when each row gets affected (i.e. a Row Level Trigger) or just once when the entire sql statement is executed(i.e.statement level Trigger).
- *WHEN (condition)* - This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

For Example:

The price of a product changes constantly. It is important to maintain the history of the prices of the products.

We can create a trigger to update the 'product_price_history' table when the price of the product is updated in the 'product' table.

1) Create the 'product' table and 'product_price_history' table

```
CREATE TABLE product_price_history
(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2) );

CREATE TABLE product
(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2) );
```

2) Create the price_history_trigger and execute it.

```
CREATE or REPLACE TRIGGER price_history_trigger
BEFORE UPDATE OF unit_price
ON product
FOR EACH ROW
BEGIN
INSERT INTO product_price_history
VALUES
(:old.product_id,
 :old.product_name,
 :old.supplier_name,
 :old.unit_price);
END;
```

3) Lets update the price of a product.

```
UPDATE PRODUCT SET unit_price = 800 WHERE product_id = 100
```

Once the above update query is executed, the trigger fires and updates the 'product_price_history' table.

4)If you ROLLBACK the transaction before committing to the database, the data inserted to the table is also rolled back.

There are two types of triggers based on which level it is triggered.

1) Row level trigger - An event is triggered for each row updated, inserted or deleted.

2) Statement level trigger - An event is triggered for each sql statement executed.

PL/SQL Trigger Execution Hierarchy

The following hierarchy is followed when a trigger is fired.
1) BEFORE statement trigger fires first.
2) Next BEFORE row level trigger fires, once for each row affected.
3) Then AFTER row level trigger fires once for each affected row. These events will alternate between BEFORE and AFTER row level triggers.
4) Finally the AFTER statement level trigger fires.

EXAMPLE 1:
Write a row trigger to insert the existing values of the salary table into a new table when the salary table is updated.
SOLUTION:
SQL> select * from employee_salary;

EMP_NO	BASIC	HRA	DA	TOTAL_DEDUCTION	NET_SALARY	GROSS_SALARY
2	15000	4000	1000	5000	15000	20000
1	31000	8000	1000	5000	35000	40000
3	14000	4000	1000	5000	15000	19000
4	14000	4000	1000	5000	15000	19000
5	13000	4000	1000	5000	15000	18000

```
SQL> get e:/p11.sql;
1 create or replace trigger t
2 after update on employee_salary
3 for each row
4 begin
5 insert into backup values
(:old.emp_no,:old.gross_salary,:new.gross_salary);
6* end;
SQL> /
```

Trigger created.

SQL> update employee_salary set gross_salary=44000 where emp_no=1;
1 row updated.

SQL> select * from backup;
EMPNO OLD_GROSS_SALARY NEW_GROSS_SALARY
---------- ---------- ----------
1 40000 44000
SQL> update employee_salary set gross_salary=20000 where emp_no=2;
1 row updated.

SQL> select * from backup;

EMPNO OLD_GROSS_SALARY NEW_GROSS_SALARY
---------- ---------- ----------
1 40000 44000
2 17600 20000

SQL> update employee_salary set gross_salary=48000 where emp_no=1;
1 row updated.

SQL> select * from backup;

EMPNO	OLD_GROSS_SALARY	NEW_GROSS_SALARY
1	40000	44000
2	17600	20000
1	44000	48000

EXAMPLE 2:
Write a trigger on the employee table which shows the old values and new values of Ename after any updations on ename on Employee table.

SOLUTION:
SQL> select * from employee;

EMP_NO	EMPLOYEE_NAME	STREET	CITY
1	rajesh	first cross	gulbarga
2	paramesh	second cross	bidar
3	pushpa	ghandhi road	banglore
4	vijaya	shivaji nagar	manglore
5	keerthi	anand sagar street	bijapur

SQL> get e:/plsql/l12.sql;

```
1 create or replace trigger show
2 before update on employee
3 for each row
4 begin
5 dbms_output.put_line('the old name was :');
6 dbms_output.put_line(:old.employee_name);
7 dbms_output.put_line('the updated new name is :');
8 dbms_output.put_line(:new.employee_name);
9* end;
SQL> /
Trigger created.

SQL> update employee set employee_name='kiran' where emp_no=1;
the old name was :
rajesh
the updated new name is :
kiran
1 row updated.

SQL> select * from employee;
```

EMP_NO	EMPLOYEE_NAME	STREET	CITY
1	kiran	first cross	gulbarga
2	paramesh	second cross	bidar
3	pushpa	ghandhi road	banglore
4	vijaya	shivaji nagar	manglore
5	keerthi	anand sagar street	bijapur

Cursors:-A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it.

This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the *active* set.

General Syntax for creating a cursor is as given below:

CURSOR cursor_name IS select_statement;

- *cursor_name – A suitable name for the cursor.*
- *select_statement – A select query which returns multiple rows.*

There are two types of cursors in PL/SQL:

Implicit cursors

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.

Explicit cursors

They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row.

Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed.

Implicit Cursors: Application

When you execute DML statements like DELETE, INSERT, UPDATE and SELECT statements, implicit statements are created to process these statements.

Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations.

Implicit Cursor Attributes

The cursor attributes available are

%FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN.

For example, when you execute INSERT, UPDATE, or DELETE statements the cursor attributes tell us whether any rows are affected and how many have been affected. When a SELECT... INTO statement is executed in a PL/SQL Block, implicit cursor attributes can be used to find out whether any row has been returned by the SELECT statement. PL/SQL returns an error when no data is selected.

For Example:

Consider the PL/SQL Block that uses implicit cursor attributes as shown below:

```
DECLARE  var_rows number(5);
BEGIN
  UPDATE employee
  SET salary = salary + 1000;
  IF SQL%NOTFOUND THEN
    dbms_output.put_line('None of the salaries where updated');
  ELSIF SQL%FOUND THEN
    var_rows := SQL%ROWCOUNT;
    dbms_output.put_line('Salaries for ' || var_rows || 'employees are updated');
  END IF;
END;
```

In the above PL/SQL Block, the salaries of all the employees in the 'employee' table are updated. If none of the employee's salary are updated we get a message 'None of the salaries where updated'. Else we get a message like for example, 'Salaries for 1000 employees are updated' if there are 1000 rows in 'employee' table.

Explicit Cursors

An **explicit cursor** is defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. We can provide a suitable name for the cursor.

There are four steps in using an Explicit Cursor.

- DECLARE the cursor in the declaration section.
- OPEN the cursor in the Execution Section.
- FETCH the data from cursor into PL/SQL variables or records in the Execution Section.
- CLOSE the cursor in the Execution Section before you end the PL/SQL Block.

1) Declaring a Cursor in the Declaration Section:

```
DECLARE
CURSOR emp_cur IS
SELECT *
FROM emp_tbl
WHERE salary > 5000;
```

In the above example we are creating a cursor 'emp_cur' on a query which returns the records of all the employees with salary greater than 5000. Here 'emp_tbl' in the table which contains records of all the employees.

2) Accessing the records in the cursor:
Once the cursor is created in the declaration section we can access the cursor in the execution
section of the PL/SQL program.

How to access an Explicit Cursor?

These are the three steps in accessing the cursor.
1) Open the cursor.
2) Fetch the records in the cursor one at a time.
3) Close the cursor.

General Syntax to open a cursor is:

OPEN cursor_name;

General Syntax to fetch records from a cursor is:

FETCH cursor_name INTO record_name;
OR
FETCH cursor_name INTO variable_list;

General Syntax to close a cursor is:

CLOSE cursor_name;

When a cursor is opened, the first row becomes the current row. When the data is fetched it is copied to the record or variables and the logical pointer moves to the next row and it becomes the current row. On every fetch statement, the pointer moves to the next row. If you want to fetch after the last row, the program will throw an error. When there is more than one row in a cursor we can use loops along with explicit cursor attributes to fetch all the records.

General Form of using an explicit cursor is:

```
DECLARE
   variables;
   records;
   create a cursor;
BEGIN
  OPEN cursor;
  FETCH cursor;
    process the records;
  CLOSE cursor;
END;
```

Explicit Cursor example:

Example 1:

```
1> DECLARE
2>    emp_rec emp_tbl%rowtype;
3>    CURSOR emp_cur IS
4>    SELECT *
5>    FROM
6>    WHERE salary > 10;
7> BEGIN
8>    OPEN emp_cur;
9>    FETCH emp_cur INTO emp_rec;
10>      dbms_output.put_line (emp_rec.first_name || '  ' || emp_rec.last_name);
11>   CLOSE emp_cur;
12> END;
```

In the above example, first we are creating a record 'emp_rec' of the same structure as of table 'emp_tbl' in line no 2. We can also create a record with a cursor by replacing the table name with the cursor name. Second, we are declaring a cursor 'emp_cur' from a select query in line no 3 - 6. Third, we are opening the cursor in the execution section in line no 8. Fourth, we are fetching the cursor to the record in line no 9. Fifth, we are displaying the first_name and last_name of the employee in the record emp_rec in line no 10. Sixth, we are closing the cursor in line no 11.

Explicit Cursor Attributes

Oracle provides some attributes known as Explicit Cursor Attributes to control the data processing while using cursors. We use these attributes to avoid errors while accessing cursors through OPEN, FETCH and CLOSE Statements.

The cursor attributes available are

%FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN.

When does an error occur while accessing an explicit cursor?

a) When we try to open a cursor which is not closed in the previous operation.
b) When we try to fetch a cursor after the last operation.

These are the attributes available to check the status of an explicit cursor.

Example:-Create a Cursor which update the salaries of an Employee as follows.

1.if sal<1000then update the salary to 1500.

2.if sal>=1000 and <2000 then update the salary to 2500.

3.if sal>=2000 and <=3000 then update the salary to 4000.

And also count the no.of records have been updated.*/

Cursor my_cur is select empno,sal from emp;

Xno emp.empno%type;
Xsal emp.sal%type;
C number;

Begin

Open my_cur;

C:=0;

Loop

Fetch my_cur into xno,xsal;

If(xsal<1000) then

Update emp set sal=3000 where empno=xno;

C:=c+1;

Else if(xsal>=2000) and xsa<3000) then

Update emp set sal=4000 where empno=xno;

C:=c+1;

End if;

End if;

Exit when my_cur%NOTFOUND;

End loop;

Close my_cur;

Dbma_output.put_line(c||'records have been successfully updated');

End;

Sql>@a.sql;

records have been successfully updated

pl/sql procedure successfully completed.

Valid Test Data

Before executing the cursor,the records in emp table as follows

Sql>select * from emp;

OUTPUT:

EMPNO ENAME JOB MGR HIREDATE SAL COMMD EPTNO

7369 SMITH CLERK 7902 17-DEC-80 2000 20

7499 ALLEN SALESMAN 7698 20-FEB-81 1600 300 30

7521 WARD SALESMAN 7698 22-FEB-81 1250 500 30

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO

--- --- --- --- --- --- ---

7566 JONES MANAGER 7839 02-APR-81 2975 20

7654 MARTIN SALESMAN 7698 28-SEP-81 1250 1400 30

7698 BLAKE MANAGER 7839 01-MAY-81 2850 30

…

….

…

14 rows selected.

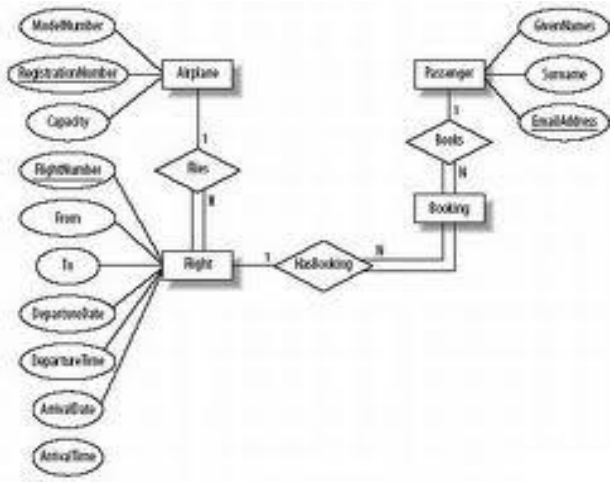
ADDITIONAL EXERCISES

- Design and Develop applications like banking, reservation system, etc.

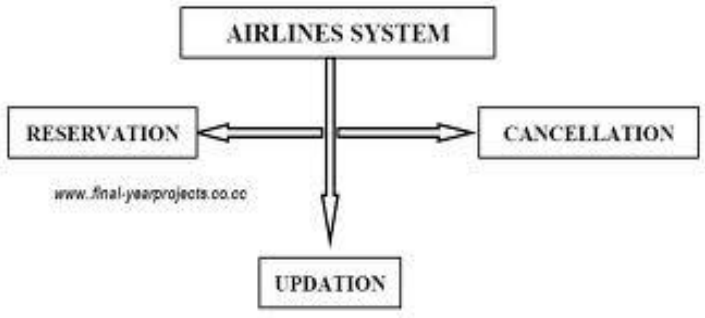
(a) Airline Reservation System

The Airline Reservation System project helps the customers to search the availability and prices of various airline tickets, along with the different packages available with the reservations. This project also covers various features like online registration of the users, modifying the details of the website by the management staff or administrator of the website, by adding, deleting or modifying the customer details, flights or packages information. In general, this website would be designed to perform like any other airline ticketing website available online.

Designing ER Diagram:-



Designing for Airline Reservation System:-



(b) Banking System

This project PRACTICAL s at creation of a secure Internet banking system. This will be accessible to all customers who have a valid User Id and Password. This is an approach to provide an opportunity to the customers to have some important transactions to be done from where they are at present without moving to bank. In this project we are going to deal the existing facts in the bank i.e.; the transactions which takes place between customer and bank. We provide a real time environment for the existing system in the bank. We deal in the method transaction in the bank can be made faster and easier that is our project is an internet based computerized approach towards banking.

Modules:

1. Balance enquiry
2. FundsTransfer to another account in the same bank
3. Request for cheque book/change of address/stop payment of Cheques
4. Viewing Monthly and annual statements.
5. System help.

ER Diagram for Banking System:-