Laboratory Manual

For

Database Management Systems

(16CS201)

B.Tech (II CSE I Sem)

Department of Computer Science & Engineering

Vignan's Foundation for Science, Technology and Research,

(Deemed to be University)

Vadlamudi, Guntur, AP.

Objective of the Course:

This lab course enables the students to acquire the following skills:

- Creation of an ER model for a given organization.
- Design of a database
- Creation and manipulation of a database using SQL
- Development of Database applications using PL/SQL
- Understanding the importance of log files
- Understanding the impact of Abort and Committed transactions
- Development of user interface: menus and forms

Contents

EXERCISE – I

ER Design tool (ex. TOAD)

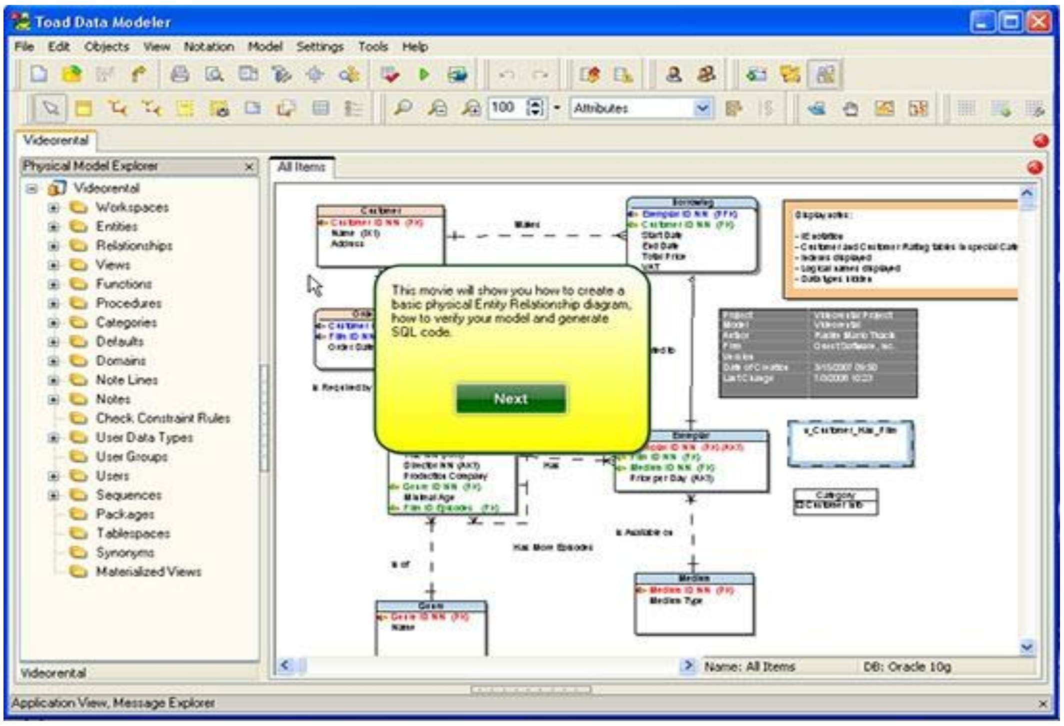
Select Tools--->ER Diagram

To create a New Diagram

- From the toolbar, select the New Diagram button. The Create ER Diagram dialog appears.
- Enter the Schema where your table resides from the drop down schema menu.
- Enter the Table you want to diagram.
- Select the number of levels of referential tables you want to load.

  Note: The more levels of referential tables you load, the more complicated the diagram will become, and the longer TOAD will take to create the diagram.
- Select your display options. You can:
  - ✓ Show primary keys
  - ✓ Show foreign keys
  - ✓ Show unique keys
  - ✓ Show data type
  - ✓ Show not nullable
  - ✓ Show indexes
- Click OK to generate the diagram.

  Sample Window:



This movie will show you how to create a basic physical Entity Relationship diagram, how to verify your model and generate SQL code.

EXERCISE – II
MYSQL RDBMS

MySQL:

MySQL is an open source SQL database, which is developed by a Swedish company – MySQL AB. MySQL is pronounced as "my ess-que-ell," in contrast with SQL, pronounced "sequel."

MySQL is supporting many different platforms including Microsoft Windows, the major Linux distributions, UNIX, and Mac OS X.

MySQL has free and paid versions, depending on its usage (non-commercial/commercial) and features. MySQL comes with a very fast, multi-threaded, multi-user and robust SQL database server.

History:

- Development of MySQL by Michael Widenius& David Axmark beginning in 1994.
- First internal release on 23rd May 1995.
- Windows Version was released on the 8th January 1998 for Windows 95 and NT.
- Version 3.23: beta from June 2000, production release January 2001.
- Version 4.0: beta from August 2002, production release March 2003 (unions).
- Version 4.01: beta from August 2003, Jyoti adopts MySQL for database tracking.
- Version 4.1: beta from June 2004, production release October 2004.
- Version 5.0: beta from March 2005, production release October 2005.
- Sun Microsystems acquired MySQL AB on the 26th February 2008.
- Version 5.1: production release 27th November 2008.

Features:

- High Performance.
- High Availability.
- Scalability and Flexibility Run anything.
- Robust Transactional Support.
- Web and Data Warehouse Strengths.
- Strong Data Protection.
- Comprehensive Application Development.
- Management Ease.
- Open Source Freedom and 24 x 7 Support.
- Lowest Total Cost of Ownership.

MS SQL Server:

MS SQL Server is a Relational Database Management System developed by Microsoft Inc. Its primary query languages are −

- T-SQL
- ANSI SQL

ORACLE:

It is a very large multi-user based database management system. Oracle is a relational database management system developed by 'Oracle Corporation'.

Oracle works to efficiently manage its resources, a database of information among the multiple clients requesting and sending data in the network.

It is an excellent database server choice for client/server computing. Oracle supports all major operating systems for both clients and servers, including MSDOS, NetWare, UnixWare, OS/2 and most UNIX flavors.

History

Oracle began in 1977 and celebrating its 32 wonderful years in the industry (from 1977 to 2009).

- 1977 - Larry Ellison, Bob Miner and Ed Oates founded Software Development Laboratories to undertake development work.
- 1979 - Version 2.0 of Oracle was released and it became first commercial relational database and first SQL database. The company changed its name to Relational Software Inc. (RSI).
- 1981 - RSI started developing tools for Oracle.
- 1982 - RSI was renamed to Oracle Corporation.
- 1983 - Oracle released version 3.0, rewritten in C language and ran on multiple platforms.
- 1984 - Oracle version 4.0 was released. It contained features like concurrency control - multi-version read consistency, etc.
- 1985 - Oracle version 4.0 was released. It contained features like concurrency control - multi-version read consistency, etc.
- 2007 - Oracle released Oracle11g. The new version focused on better partitioning, easy migration, etc.

Features:

- Concurrency
- Read Consistency

- Locking Mechanisms
- Quiesce Database
- Portability
- Self-managing database
- SQL*Plus

What is RDBMS?

RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

What is a Table?

The data in an RDBMS is stored in database objects which are called as **tables**. This table is basically a collection of related data entries and it consists of numerous columns and rows.
Remember, a table is the most common and simplest form of data storage in a relational database.
The following program is an example of a CUSTOMERS table –

```
| 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|
```

What is a Field?

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.
A field is a column in a table that is designed to maintain specific information about every record in the table.

What is a Record or a Row?

A record is also called as a row of data is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table –

```
+----+----------+-----+-----------+----------+
|1|Ramesh|32|Ahmedabad|2000.00|
+----+----------+-----+-----------+----------+
```

A record is a horizontal entity in a table.

What is a Column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would be as shown below −

```
+-----------+
| ADDRESS   |
+-----------+
|Ahmedabad|
|Delhi|
|Kota|
|Mumbai|
+----+------+
```

What is a NULL value?

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is the one that has been left blank during a record creation.

Data:

Data is a collection of facts which is unorganized but can be made organized into useful information. The term Data and information come across in our daily life and are often useful interchanged.

Example: Weights, prices, costs, number of items sold etc.

Information:

The information as data that has been processed in such a way increases the knowledge of the person who uses it.

Data base:

An organized collection of logically related data is called Data Base.

DataBase Management System:

The Data Base Management System manages the database. It is software that enables us to define, create and maintain the database and provides controlled access to the database.

Data models:

Data model means to model the data i.e. to give a shape to the data to give a figure to the stored data. The DBMS allows a user to define the stored data in terms of data model.

The data models are divided into three different groups,

1. Object Based Logical model

- Entity Relationship model
- Object oriented model
- Semantic Data model
- Functional Data model

2. Record Based Logical model.
    - Relational Model
    - Network Model
    - Hierarchical model
3. Physical model.
    - Unifying model
    - Frame memory model

DATA TYPES:

Data types are a classification of a particular type of information. It is easy for humans to distinguish between types of data. SQL supports following data types.

There are mainly 09 types of data types used in SQL. Those are

1. **Char (N):**

This data type represents a fixed length string of exactly n characters. Where n is greater than zero and should be an integer. The maximum number of characters this data type can hold is 255 characters. Oracle compares char values using Blank padded comparison semantics.

Ex: Name char (10); The name is a string of 10 characters.

2. **Varchar(N):**

This data type represents a varying length string whose maximum length is n characters. The maximum number of characters this data type can hold is 2000 characters. One difference between these data types and the char data types is oracle compares varchar values using non padded comparison semantics.

Ex: Name varchar2 (n)

3. **Number (P,S):**

The NUMBER data type is used to store numbers. Numbers of virtually any magnitude may be stored up to 38 digits of precision. The precision (p) determines the maximum length of the data, whereas the scale determines number of places to the right of the decimal.

Ex: Price number (7,2);

4. **Integer:**

An integer represents a signed integer decimal in binary.

Ex: Roll no integer;

5. **Date:**

This data type is used to represent date and time. The standard format is YY-MON-DD. To enter dates other than the standard format, use appropriate functions. Date Time stores date in the 24 hour format. The default date for a date field is the first day of the current month.

Ex: Jdate date;

6. **BLOB (binary large object):**
   To store graphics, video clips and sound files. The max. Size is 4GB.
7. **CLOB (character large object):**
   Containing single byte characters. Max. Size is 4GB.
8. **Long:**
   This data type is used to store variable length character strings containing up to 2GB.It can be used to store arrays of binary data in ASCII format. Long values cannot be indexed and the normal character functions such as SUBSTR cannot be applied to LONG values.
9. **Rawlongraw:**
   The RAW/LONGRAW data type is used to store binary data such as Picture or images.
   - The RAW can have a maximum length of 255 bytes.
   - LONGRAW data type can contain up to 2GB.

Data Definition Language (DDL)

The SQL sentences that are used to create the objects are called DDL or Data Definition Language.

Commands in DDL: There are four commands in DDL. They are

10. CREATE
11. ALTER
12. DROP
13. TRUNCATE

1. **Create:**

The CREATE command is used to create the schema for the table.

SYNTAX:

```
CREATE TABLE <TABLE NAME> (COLUMNNAME
DATATYPE (SIZE),);
```

2. **Alter:**

The ALTER command is used to modify the field types and size of the fields. It is also used to add a column or field to the table.

SYNTAX:

ADD:

```
ALTER TABLE <TABLE NAME>
ADD COLUMN NAME DATATYPE (SIZE),
  . . . . . . .. . . ;
```

MODIFY:

1. Column Renaming and DataType changing:

   ALTER TABLE <TABLE NAME>

   MODIFY COLUMN NAME DATATYPE (SIZE),. . . . . . . ;

2. Table Name Renaming

   ALTER TABLE <TABLE NAME> RENAME TO <NEW TABLE NAME>;

3. **Drop:**

   This DROP command is used to remove table from the database.

SYNTAX: DROP TABLE <TABLE NAME>;

4. **Truncate:**

   This TRUNCATE command is used to delete from the database.

SYNTAX: TRUNCATE TABLE <TABLE NAME>;

5. SQL DESC:

   This command will give the description about the table.

   DESC <TABLE NAME>;

Data Manipulation Language (DML)

The SQL sentences used to manipulate data within the objects are called DML or Data Manipulation Language. Data is retrieved and manipulated using DML.

COMMANDS IN DML:

There are four commands in DML. They are

1. INSERT
2. DELETE
3. UPDATE
4. SELECT

1. **INSERT**:

In Addition to inserting data one row at a time into a table, It is quite possible to populate a table with data that already exists in another table. The syntax for doing so is described under.

Syntax: METHOD 1

```
INSERT INTO  <TABLE NAME>
             (ATTRIBUTES1, ATTRIBUTS2, ATTRIBUTSN)
                 VALUES
                 (VALUE1, VALUE2, VALUEN);
```

METHOD 2

INSERT INTO <TABLE NAME>

VALUES

(VALUE1, VALUE2, VALUEN);

METHOD 3 (Dumping from other tables)

INSERT INTO <TABLE NAME>

SELECT <COLUMN NAME>, <COLUMN NAME>,

FROM <TABLE NAME>;

Insert of a data set into a table from another table

Syntax:

INSERT INTO <TABLE NAME>

SELECT <COLUMN NAME>, **<**COLUMN NAME>,

FROM <TABLE NAME>

WHERE <COLUMN NAME> = <EXPRESSION>;

2. **DELETE**:

   This DELETE command is used to delete data from the database. To remove

   - All rows from a table
   - A select set of rows from a table

Removal of all rows:

Syntax:

DELETE FROM <TABLE NAME>;

Removal of specified rows

Syntax:

DELETE FROM <TABLE NAME> **WHERE** <CONDITION>;

3. **UPDATE:**

This UPDATE command is used to modify data in the table. To update

- All the rows from a table
- A select set of rows from a table

Updating of all rows:

Syntax:

UPDATE<TABLENAME>

SET<COLUMN_NAME>=<EXPRESSION>, < COLUMN_NAME > = <EXPRESSION>;

4. **SELECT:**

   This SELECT command is used to retrieve data from the database.

SYNTAX: (*To Display all Records)*

```
SELECT *
FROM <TABLE_NAME>;
```

SYNTAX: (*To Display some specific columns)*

```
SELECT <COLUMN_NAME 1>, < COLUMN_NAME2>. . . . .
FROM <TABLE_NAME>;
```

SYNTAX: (*To Display some distinct records)*

```
SELECT DISTINCT <COLUMN_NAME 1>, < COLUMN_NAME2>. . . . .
FROM <TABLE_NAME>;
```

SYNTAX: (*To Display some specific records with constraints using where clause)*

```
SELECT <COLUMN_NAME 1>, < COLUMN_NAME2>. . . . .
FROM <TABLE_NAME>
WHERE CONDITION;
```

SYNTAX: (*To Display some logically satisfied records using* AND|OR *clause)*

```
SELECT DISTINCT <COLUMN_NAME 1>, < COLUMN_NAME2>. . . . .
FROM <TABLE_NAME>
WHERE CONDITION-1 {AND|OR} CONDITION-2;
```

SYNTAX: (*To Display some records using IN clause)*

```
SELECT <COLUMN_NAME 1>, < COLUMN_NAME2>. . . . .
FROM <TABLE_NAME1>
WHERE <COLUMN_NAME 1> IN (val-1, val-2,...val-N);
```

SYNTAX: (*To Display some records using BETWEEN Clauses)*

```
SELECT <COLUMN_NAME 1>, < COLUMN_NAME2>. . . . .
FROM <TABLE_NAME1>
WHERE <COLUMN_NAME> BETWEEN val-1 AND val-2;
```

SYNTAX: (*To Display some records using LIKE Operator)*

```
SELECT <COLUMN_NAME 1>, < COLUMN_NAME2>. . . . .
FROM <TABLE_NAME1>
WHERE <COLUMN_NAME> LIKE {PATTERN};
```

POSSIBLE METHODS:

WHERE <COLUMN_NAME> LIKE 'a%'	Finds any values that start with "a"
WHERE <COLUMN_NAME> LIKE '%a'	Finds any values that end with "a"
WHERE <COLUMN_NAME> LIKE '%or%'	Finds any values that have "or" in any position

WHERE <COLUMN_NAME> LIKE '_r%'	Finds any values that have "r" in the second position
WHERE <COLUMN_NAME> LIKE 'a_%_%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE <COLUMN_NAME> 'a%o'	Finds any values that start with "a" and ends with "o"

SYNTAX: (*To Display records in specific order using* ORDER *Operator*)

```
SELECT <COLUMN_NAME 1>, < COLUMN_NAME2>. . . . .
FROM <TABLE_NAME>
WHERE <CONDITION>
ORDER BY <COLUMN_NAME> {ASC|DESC};
```

SYNTAX: (*To Display records into groups using* GROUP *Operator*)

```
SELECT <COLUMN_NAME 1>, < COLUMN_NAME2>. . . . .
FROM <TABLE_NAME>
WHERE <CONDITION>
GROUP BY <COLUMN_NAME>;
```

SYNTAX: (*To Display count of records using* COUNT *Operator*)

```
SELECT COUNT<COLUMN_NAME>
FROM <TABLE_NAME>
WHERE <CONDITION>;
```

SYNTAX: (*To Display count of records using* HAVING *Operator*)

```
SELECT <COLUMN_NAME>
FROM   <TABLE_NAME>
WHERE CONDITION
GROUP BY <COLUMN_NAME>
HAVING (arithmetic function condition);
```

Data Control Language (DCL)

This Data Control Language consists of two commands. They are

1. GRANT
2. REVOKE

1. **GRANT:**

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

SYNTAX:

```
GRANT {OBJECT PRIVILIGE}
ON OBJECTNAME
TO USERNAME
[WITH GRANT OPTION];
```

2. **REVOKE:**

The REVOKE statement is used to deny the GRANT on the object.

SYNTAX:

```
REVOKE {OBJECT PRIVILIGE}
ON OBJNAME
FROM USERNAME
[WITH REVOKE OPTION];
```

Transaction Control Language (TCL)

The Transaction Control Language consists of three commands. They are

1. COMMIT
2. ROLE BACK
3. SAVE POINT

1. **COMMIT:**

The COMMIT statement ends the current transaction and makes permanent any changes made during transaction. Until we commit the changes other users cannot access the changed data.

2. **ROLLBACK:**

The ROLLBACK statement is the inverse of COMMIT. It ends the current transaction and undoes any changes made during transaction.

3. **SAVE POINT:**

SAVE POINT names and marks the current point in the processing of a transaction. Used with the ROLLBACK to statement SAVE POINT let us undo parts of a transaction instead of the whole transaction.

Constraints in SQL:

Rules which are enforced on data being entered, and prevents the user from entering invalid data into tables are called constraints.

TYPES OF CONSTRAINTS:

There are six types of constraints.

1. NOT NULL
2. UNIQUE
3. PRIMARY KEY
4. FOREIGN KEY
5. CHECK
6. DEFAULT

1. **NOT NULL:** NOT NULL means that column never empty.

```
EX: CREATE TABLE CUSTOMER (CNO INT NOTNULL,
                           CNAME VARCHAR(20) NOTNULL,
                           ADDRESS VARCHAR(20),
                           PHONENO INT(10));
```

2. **UNIQUE:**

It is used to uniquely identify each value in a column.

```
EX: CREATE TABLE CUSTOMER (CNO INT UNIQUE NOTNULL,
                           CNAME VARCHAR(20) NOTNULL,
                           ADDRESS VARCHAR(20),
                           PHONENO INT(10));
```

3. **PRIMARY KEY:**

A primary key is one or more columns in a table used to uniquely identify each row in a table.

```
EX: CREATE TABLE CUSTOMER (CNO INT(4),
                           CNAME VARCHAR(20),
                           ADDRESS VARCHAR(20)
                           PRIMARY KEY(CNO,CNAME));
```

4. **FOREIGN KEY:**

A foreign key is an attribute in one relation which acts as a primary key in another relation of the same database. It is used to establish the relationship between two tables.

```
EX:  CREATE TABLE EMP (EMPNO   INT(4) PRIMARY KEY,
                       NAME VARCHAR2(20),
                         SAL FLOAT (7,2),
                         JDATE DATE,
                         DEPTNO INT(4) REFERENCES
```

```
                    DEPT (DEPTNO));
CREATE TABLE DEPT (DEPTNO INT(4)PRIMARY KEY,
                   DNAME VARCHAR(20),
                   LOC VARCHAR(20));
```

5. CHECK:

Many columns must have values that are within a certain range or that satisfy certain condition with a CHECK constraint .A condition can be specified by the value entered for the column.

```
EX:    CREATE TABLE CUSTOMER (CNO NUMBER (4),
                              CNAME VARCHAR (20),
                              ADDRESS VARCHAR (20) CHECK
                              ADDRESS IN (‘GNT’,’TNL’,’VJA’),
                              PHONE INT (12));
```

6. DEFAULT:

While inserting a row into a table without having values for every column SQL must insert a default value to fill in the executed columns. The most common default value is NULL.

```
EX:      CREATE TABLE CUSTOMER (CNO INT(4),
                                CNAME VARCHAR(20),
                                ADDRESS VARCHAR(20),
                                PHONE   INT (12),
                                PINCODE  INT(6) DEFAULT
                                PINCODE IS NULL);
```

Exercise – III
Table Operations- Creation, Insertion, Selection, Updating, Deletion and Constraints.

In the Exercise we are going to create a sailor tables

Following tables (Relations) are considered for the lab purpose.

- **SAILORS (*<u>SID: INTEGER</u>, SNAME: STRING, RATING: FLOAT, AGE: REAL*)**
- **BOATS (*<u>BID: INTEGER</u>, BNAME: STRING, COLOR: STRING*)**
- **RESERVES (*SID: INTEGER, BID: INTEGER, DAY: DATE*)**

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

Syntax: Creating SAILORS **Table with Primary Key:**

SQL> CREATE TABLE SAILORS (SID INT (5), SNAME VARCHAR(30), RATING INT(5), AGE INT(4), **PRIMARY KEY(SID)**);

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.

Syntax: Creating RESERVES **Tables with Foreign Key**

The following example gives the complete definition to create Reserves table (Defines Primary Key as well as Foreign Keys).

SQL> CREATE TABLE RESERVES (SID INT(5), BID INT(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.

SQL> CREATE TABLE BOAT (BID INT(5), BNAME VARCHAR(20), BCOLOR VARCAHR(20));

Creating table with some Constraint:- Suppose we want to add rule for rating of the sailors - "Rating should be between 1 to 10", His ID should be unique, He should have some name and some default ratings value while creating table then we can use following command.
SQL> CREATE TABLE SAILORS (SID INT(5) **UNIQUE**, SNAME VARCHAR(30) **NOT NULL**, RATING INT(5) **DEFAULT 2**, AGE NUMBER(4,2), **PRIMARY KEY**(SID), **CHECK** (RATING >=1 ANDRATING <=10));

Deleting Table:- The table along with its definition & data can be deleted using following command.
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.
SQL> ALTER TABLE SAILORS ADD COLUMN SALARY FLOAT(4,2);

To remove the attribute from an existing relation we can use following Command.
SQL> ALTER TABLE SAILORS DROP COLUMN SALARY;

To add the constraint to existing relation we can use ALTER TABLE Command.
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.
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.
SQL> INSERT INTO SAILORS VALUES (1, 'Shan, 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.

SQL> INSERT INTO SAILORS (SNAME, SID, AGE, RATING) VALUES (‘Shan’, 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.

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.

SQL> DELETE FROM SAILORS WHERE SNAME = ‘Shan’;

To delete all records from the table

SQL> DELETE FROM SAILORS;

To change particular value:- We can modify the column values in an existing row using the UPDATE command.

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:

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.

3) Find names of the sailors who have reserved at least one boat.

EXERCISE – IV

Special Operations, Aggregate Functions and 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.

DERIVED ATTRIBUTES:

SQL provides the way to derive values from stored attributes in the name of Derived Attributes.

Ex:- Retrieve the age of the sailors from the date of birth.

SQL> SELECT SID, CURDATE(), **TIMESTAMPDIFF(YEAR, DOB, CURDATE())** FROM SAILORS;

Similarly we can retrieve days, month and we can give name to the derived attributes using **Aliases** keyword.

SQL> SELECT SID, **TIMESTAMPDIFF(MONTH, DOB, CURDATE()) as Months old** FROM SAILORS;

SQL> SELECT SID, **TIMESTAMPDIFF(DAYS, DOB, CURDATE()) as Days Old** FROM SAILORS;

SET OPERATIONS:

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

Note that UNION, INTERSECT, and MINUS can be used on any two tables that are union compatible, that is, have the same number of columns and the columns, taken in order, have the same types.

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 boats and 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’;

AGGREGATE FUNCTIONS:- In addition to simply retrieving data, we often want to perform 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 significant extension of relational algebra. SQL supports five aggregate operations, which can be applied on any 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;

Note that it does not make sense to specify DISTINCT in conjunction with MIN or MAX (Although SQL does not preclude this).

Write the following queries using Aggregate Functions.

1) Find the average age of sailors with a rating of 10.

2) Count the number of different sailor names.

3) Find the name and age of the oldest sailor.

4) Count the number of Sailors.

5) Find the names of sailors who are older than the oldest sailor with a rating of 10.

CLAUSE FUNCTIONS:

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).

4) Displays all the sailors according to rating (Lower Rating First), if rating is same then sort according to age (Younger First).

GROUP BY and HAVING Clauses:-Thus far, we have applied aggregate operations to all (qualifying) rows in a relation. Often we want to apply aggregate operations to each of a number of groups of rows in a relation, where the number of groups depends on the relation instance. 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, where the 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;

NESTED QUERIES:- For retrieving data from the tables we have seen the simple & basic queries. These queries extract the data from one or more tables. Here we are going to see some complex & powerful queries that enables us to retrieve the data in desired manner. One of the most powerful features of SQL is nested queries. A nested query is a query that has another query 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.

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)

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.

EXERCISE – VI

Join Queries

SQL JOINS

1. INNER JOIN
2. OUTER JOIN
  a. LEFT OUTER JOIN
  b. RIGHT OUTER JOIN
  c. FULL OUTER JOIN
3. SELF JOIN
4. EQUI JOIN
5. NON EQUI JOIN

TABLE cseitstudent

```
CREATE TABLE CSEITSTUDENT
(
        STUDENTID NUMBER PRIMARY KEY,
        SNAME VARCHAR(30),
        DEPARTMENT CHAR(5),
        SEM NUMBER
);
```

TABLE placement

```
CREATE TABLE PLACEMENT
(
        PLACEMENTID NUMBER PRIMARY KEY,
        STUDENTID NUMBER,
        DEPARTMENT CHAR(5),
        COMPANY VARCHAR2(30),
        SALARY NUMBER
);
```

// Inserting values into cseitstudent table

```
 INSERT INTO CSEITSTUDENT (STUDENTID, SNAME, DEPARTMENT, SEM) VALUES(101,'REEMA',
'IT',5);

 INSERT INTO CSEITSTUDENT (STUDENTID, SNAME, DEPARTMENT, SEM) VALUES(102,'REENU',
'IT',3);

INSERT INTO CSEITSTUDENT (STUDENTID, SNAME, DEPARTMENT, SEM) VALUES(103,'SHEELA',
'CSE',3);

INSERT INTO CSEITSTUDENT (STUDENTID, SNAME, DEPARTMENT, SEM) VALUES(104,'NIRMAL',
'IT',3);

INSERT INTO CSEITSTUDENT (STUDENTID, SNAME, DEPARTMENT, SEM)
VALUES(105,'ESHWAR', 'CSE',5);
```

// Inserting values into placement table

```
INSERT INTO PLACEMENT VALUES(1, 104, 'IT', 'INFOSYS', 25000);
INSERT INTO PLACEMENT VALUES(2, 105, 'CSE', 'WIPRO', 22000);

INSERT INTO PLACEMENT VALUES(3, 204, 'MECH', 'HYUNDAI', 30000);

INSERT INTO PLACEMENT VALUES(4, 102, 'IT', 'INFOSYS', 25000);

INSERT INTO PLACEMENT VALUES(5, 103, 'CSE', 'INFOSYS', 25000);
```

// Display the values in the table as rows

```
SELECT * FROM CSEITSTUDENT;
```

```
        STUDENTID   SNAME          DEPARTMENT      SEM
        101         reema          IT              5
        102         reenu          IT              3
        103         sheela         CSE             3
        104         nirmal         IT              3
        105         eshwar         CSE             5
```

```
SELECT * FROM PLACEMENT;
```

```
    PLACEMENTID        STUDENTID   DEPARTMENT        COMPANY     SALARY

        1                 104      IT                infosys     25000
        2                 105      CSE               Wipro       22000
        3                 204      MECH              HYUNDAI     30000
        4                 102      IT                Infosys     25000
        5                 103      CSE               Infosys     25000
```

// INNER JOIN

```
SELECT *
FROM CSEITSTUDENT
INNER JOIN PLACEMENT
ON CSEITSTUDENT.STUDENTID=PLACEMENT.STUDENTID;
```

```
STUDENTID  SNAME        DEPARTMENT  SEM  PLACEMENTID   STUDENTID  DEPARTMENT  COMPANY
        SALARY
  104        nirmal     IT            3            1   104        IT          infosys
        25000
  105       eshwar      CSE           5            2   105        CSE         Wipro
        22000
  102        reenu      IT            3            4   102        IT          infosys
        25000
  103        sheela     CSE           3            5   103        CSE         infosys
        25000
```

```
SELECT CSEITSTUDENT.STUDENTID, CSEITSTUDENT.SNAME,PLACEMENT.COMPANY,
PLACEMENT.SALARY
FROM CSEITSTUDENT
INNER JOIN PLACEMENT
ON CSEITSTUDENT.STUDENTID=PLACEMENT.STUDENTID;
```

```
        STUDENTID   SNAME       COMPANY     SALARY
        104         nirmal      infosys     25000
        105         eshwar      Wipro       22000
        102         reenu       infosys     25000
        103         sheela      infosys     25000
```

//LEFT OUTER JOIN

```
SELECT *
FROM CSEITSTUDENT
LEFT OUTER JOIN PLACEMENT
ON CSEITSTUDENT.STUDENTID=PLACEMENT.STUDENTID;
```

```
STUDENTID     SNAME  DEPARTMENT  SEM  PLACEMENTID  STUDENTID  DEPARTMENT  COMPANY
       SALARY
   104        nirmal  IT          3    1            104        IT           infosys
       25000
   105        eshwar  CSE         5    2            105        CSE          Wipro
       22000
   102        reenu   IT          3    4            102        IT           infosys
       25000
   103        sheela  CSE         3    5            103        CSE          infosys
       25000
   101        reema   IT          5    -            -          -            -          -
```

```
SELECT CSEITSTUDENT.SNAME,PLACEMENT.PLACEMENTID, PLACEMENT.COMPANY
FROM CSEITSTUDENT
LEFT OUTER JOIN PLACEMENT
ON CSEITSTUDENT.STUDENTID=PLACEMENT.STUDENTID;
```

```
SNAME        PLACEMENTID       COMPANY
nirmal            1            infosys
eshwar            2            Wipro
reenu             4            infosys
sheela            5            infosys
reema             -             -
```

//RIGHT OUTER JOIN

```
SELECT *
FROM CSEITSTUDENT
RIGHT OUTER JOIN PLACEMENT
ON CSEITSTUDENT.STUDENTID=PLACEMENT.STUDENTID;
```

```
STUDENTID     SNAME   DEPARTMENT     SEM  PLACEMENTID  STUDENTID     DEPARTMENT  COMPANY
SALARY
102           reenu   IT              3      4              102       IT          infosys
25000
103           sheela  CSE             3      5              103       CSE         infosys
25000
104           nirmal  IT              3      1              104       IT          infosys
25000
105           eshwar  CSE             5      2              105       CSE         Wipro
22000
  -          -   -        -                  3              204       MECH        HYUNDAI
30000
```

```
SELECT CSEITSTUDENT.SNAME,PLACEMENT.PLACEMENTID, PLACEMENT.COMPANY
FROM CSEITSTUDENT
RIGHT OUTER JOIN PLACEMENT
ON CSEITSTUDENT.STUDENTID = PLACEMENT.STUDENTID;
```

```
SNAME          PLACEMENTID COMPANY
reenu               4           infosys
```

```
sheela          5          infosys
nirmal          1          infosys
eshwar          2          Wipro
                3          HYUNDAI
```

//FULL OUTER JOIN

```
SELECT *
FROM CSEITSTUDENT
FULL OUTER JOIN PLACEMENT
ON CSEITSTUDENT.STUDENTID=PLACEMENT.STUDENTID;
```

```
STUDENTID   SNAME   DEPARTMENT   SEM  PLACEMENTID  STUDENTID   DEPARTMENT  COMPANY
      SALARY
   104      nirmal   IT          3    1            104         IT          infosys
      25000
   105      eshwar   CSE         5    2            105         CSE         Wipro
      22000
    -        -       -           -    3            204         MECH        HYUNDAI
      30000
   102      reenu    IT          3    4            102         IT          infosys
      25000
   103      sheela   CSE         3    5            103         CSE         infosys
      25000
   101      reema    IT          5    -            -           -           -        -
```

```
SELECT CSEITSTUDENT.SNAME,PLACEMENT.PLACEMENTID, PLACEMENT.COMPANY
FROM CSEITSTUDENT
FULL OUTER JOIN PLACEMENT
ON CSEITSTUDENT.STUDENTID=PLACEMENT.STUDENTID;
```

```
SNAME        PLACEMENTID        COMPANY
nirmal            1             infosys
eshwar            2             Wipro
   -              3             HYUNDAI
reenu             4             infosys
sheela            5             infosys
reema             -              -
```

//SELF JOIN

Returns rows by comparing the values of the same table.

//TABLE CREATION

```
CREATE TABLE EMPLOYEE
(
        EMPID NUMBER,
        EMPNAME VARCHAR2(25),
         REPORTINGID NUMBER
);
```

//INSERTING VALUES IN THE TABLE

```
INSERT INTO EMPLOYEE VALUES(1, 'PRINCIPAL', NULL);
INSERT INTO EMPLOYEE VALUES(2, 'HOD', 1);

INSERT INTO EMPLOYEE VALUES(3, 'PO', 1);

INSERT INTO EMPLOYEE VALUES(4, 'STAFF', 2);

INSERT INTO EMPLOYEE VALUES(5, 'NON TEACHING STAFF', 2);
```

//DISPLAYS VALUES IN THE TABLE

```
SELECT * FROM EMPLOYEE;
```

```
EMPID          EMPNAME        REPORTINGID
1              Principal            -
2              HOD                  1
3              PO                   1
4              Staff                2
5              Non Teaching Staff   2
```

```
SELECT E1.EMPID, E1.EMPNAME, E2.EMPNAME AS HEADNAME
FROM EMPLOYEE E1, EMPLOYEE E2
WHERE E1.REPORTINGID=E2.EMPID;
```

```
EMPID  EMPNAME              HEADNAME
2      HOD                  Principal
3      PO                   Principal
4      Staff                HOD
5      Non Teaching Staff   HOD
```

//EQUI JOIN

```
SELECT * FROM CSEITSTUDENT, PLACEMENT WHERE
CSEITSTUDENT.STUDENTID=PLACEMENT.STUDENTID;
```

```
STUDENTID    SNAME   DEPARTMENT    SEM  PLACEMENTID  STUDENTID   DEPARTMENT  COMPANY
        SALARY
  104        nirmal   IT           3    1             104        IT            infosys
        25000
  105        eshwar   CSE          5    2             105        CSE           Wipro
        22000
  102        reenu    IT           3    4             102        IT            infosys
        25000
  103        sheela   CSE          3    5             103        CSE           infosys
        25000
```

// NON EQUI JOIN

```
SELECT CSEIT.STUDENTID, CSEIT.SNAME FROM CSEITSTUDENT CSEIT, PLACEMENT PLACED
WHERE CSEIT.STUDENTID>PLACED.STUDENTID;
```

```
STUDENTID     SNAME
105           eshwar
105           eshwar
105           eshwar
104           nirmal
104           nirmal
103           sheela
```

EXERCISE – VII

VIEWS.

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;
```

EXERCISE – VIII

High level programming language extensions (Control structures, Procedures and Functions).

A) Creation of simple PL/SQL program which includes declaration section, executable section and exception – handling section

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;
```

B) Insert data into student table and use COMMIT, ROLLBACK and SAVEPOINT in PL/SQL block.

AIM:

1. To study Data Control Language commands and build in functions.

DCL:

The commands under DCL are

1. Savepoint
2. commit
3. Rollback.

1. SAVEPOINT

Syntax:

```
savepoint username;
```

E.g

```
SQL>savepointemp;
```

Output:

```
savepoint created;
```

Purpose:

It is a point within a transaction to which the users can rollback

2. COMMIT

Syntax:

```
commit;
```

E.g

```
SQL> commit;
```

Output:

```
Commit complete.
```

Purpose:

It indicates that changes have been made permanent & it is visible to other users.

3. ROLLBACK

Syntax:

rollback to *savepoint_text_identifier*;

E.g

```
SQL> rollback to emp;
```

Output:

```
Rollback complete.
```

Purpose:

It restores the database to original since the last commit

Use of Commit, Savepoint& Rollback in PL/SQL

We can use these commands in PL/SQL. We can use any (Insert, Delete or Update) operations for Save point & Rollback.

The following program inserts a record into Sailors table then updates a record before we Commit a Save point is defined and we can use Rollback to undo the operations we have done after the Save point (i.e. deleting a Sailors record is undone). We can define number of Savepoint statements in a program and Rollback to any point.

```
BEGIN
INSERT INTO SAILORS VALUES('32','HEMANT',10, 30);
UPDATE SAILORS SET SNAME='RAJENDRA' WHERE SID='10';
SAVEPOINT S1;
DELETE FROM SAILORS WHERE SID='11';
ROLLBACK TO S1;
COMMIT;
END;
/
```

(You can even read these values from Keyboard)

C) Program development using WHILE loops, numeric FOR loops, nested loops using error handling, built-in exceptions, user defined exceptions, RAISE application error.

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;
```