DATABASE MANAGEMENT SYSTEMS

LAB MANUAL

DEPARTMENT OF INFORMATION TECHNOLOGY

II B.Tech.(IT) II SEM

18ITL42

II Year B.Tech. IT II-Sem

Database Management Systems Lab
(Common to CSE, IT)

Pre-requisites: Database Management Systems

Course Objectives: The students will learn the following:

1. Analyze the database languages.
2. Interpret the Knowledge on database design.
3. Determine the knowledge on key constraints and Normalization.
4. Determine the knowledge on procedures and functions.

Course Outcomes: After Completion of this Course, Students will be able to:

1. Implement DDL, DML, DCL Commands using SQL.
2. Design database by using ER Diagrams.
3. Apply key constrains to get a normalized database.
4. Implement procedures and functions using PL/SQL

Experiment 1: Working with ER Diagram and Normalization

Example: ER Diagram for Sailors Database
Entities:

1. Sailor
2. Boat

Relationship: Reserves
Primary Key Atributes:

1. SID (Sailor Entity)
2. BID (Boat Entity)

Experiment 2: Working with DDL, DML, DCL and Key Constraints

Creation, Altering and Dropping of Tables and Inserting Rows into a Table (Use Constraints While Creating Tables) Examples Using Select Command.

Experiment 3: Working with Queries and Nested QUERIES

Queries (along with sub Queries) using ANY, ALL, IN, EXISTS, NOTEXISTS, UNION, INTERSET, Constraints

Experiment 4: Working with Queries USING Aggregate Operators & views

Queries using Aggregate Functions (COUNT, SUM, AVG, MAX and MIN), GROUP BY, HAVING and Creation and Dropping of Views

Experiment 5: Working with Conversion Functions & String Functions

Queries using Conversion Functions (TO_CHAR, TO_NUMBER AND TO_DATE), String Functions (CONCATENATION, LPAD, RPAD, LTRIM, RTRIM, LOWER, UPPER, INITCAP, LENGTH, SUBSTR AND INSTR), Date Functions (SYSDATE,

NEXT_DAY, ADD_MONTHS, LAST_DAY, MONTHS_BETWEEN), LEAST, GREATEST, TRUNC, ROUND, TO_CHAR, TO_DATE

Experiment 6: Working with Triggers using PL/SQL

Develop Programs using BEFORE and AFTER Triggers, Row and Statement Triggers and INSTEAD OF Triggers.

Experiment 7: Working with PL/SQL Procedures
Programs Development using Creation of Procedures, Passing Parameters IN and OUT of PROCEDURES

Experiment 8: Working with LOOPS using PL/SQL and Exception Handling
Program Development using WHILE LOOPS, Numeric FOR LOOPS.

Experiment 9: Working with Functions Using PL/SQL
Program Development using Creation of Stored Functions, Invoke Functions in SQL Statements and Write Complex Functions.

Experiment 10: Working CURSORS
Develop Programs using Features Parameters in a CURSOR, FOR UPDATE CURSOR, WHERE CURRENT of Clause and CURSOR Variables

Experiment11: Installation of SQL

Textbooks:

1. Oracle PL/SQL by Example, Benjamin Rosenzweig, Elena Silvestrova,
2. Oracle Database Logic PL/SQL Programming, Scott Urman, Tata Mc-Graw Hill.
3. SQL and PL/SQL for Oracle 10g, Black Book, Dr .P.S. Deshpande.

Database Management Systems Lab

- **Data :** Raw Facts existing in world. Consists of real world things like Numbers and Text sometimes without any context.
- **Information :** Data with Context. It is Processed Data along with Value Added to Data. Sometimes information may come in Summarized, Organized and Analyzed manner.
- **Database :** Database is a collection of data, typically describing the activities of one or more related organization. Database is an organized way to store, retrieve and perform operations on a set of interrelated data that its contents can easily be accessed, managed, and updated.
- **Database Management Systems (DBMS),** is software designed to assist in maintaining and utilizing large collection of data.
- **DBMS** contains information about a particular enterprise
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both *convenient* and *efficient* to use.
- **Database System:** This refers to the Database along with the DBMS and the programs/queries that enable the application to function as a whole.
- **Database Applications:**
  - **Banking** : all transactions
  - **Airlines/Railway/Bus**: reservations, schedules
  - **Universities/Colleges**: registration, grades, staff & student details, Account.
  - **Sales** : customers, products, purchases
  - **Online retailers** : order tracking, customized recommendations
  - **Manufacturing** : production, inventory, orders, supply chain
  - **Human Resources** : employee records, salaries, tax deduction.
  - **Hospital** : Patient Info, Doctors Details, Lab Reports.
  - **Entertainment** : Event Booking, Schedules.
  - **Shopping Mall** : Sales, Billing, Inventory, Stock.
  - **etc.**
- **Purpose of Database Systems :** In the early days, database applications were built directly on top of file systems. Drawbacks of using file systems to store data:
  - Data redundancy and inconsistency
  - Difficulty in accessing data
  - Data isolation
  - Integrity problems
  - Atomicity of updates
  - Concurrent access by multiple users
  - Security problems
- Database systems offer solutions to all the above problems

Data Definition Language (DDL)

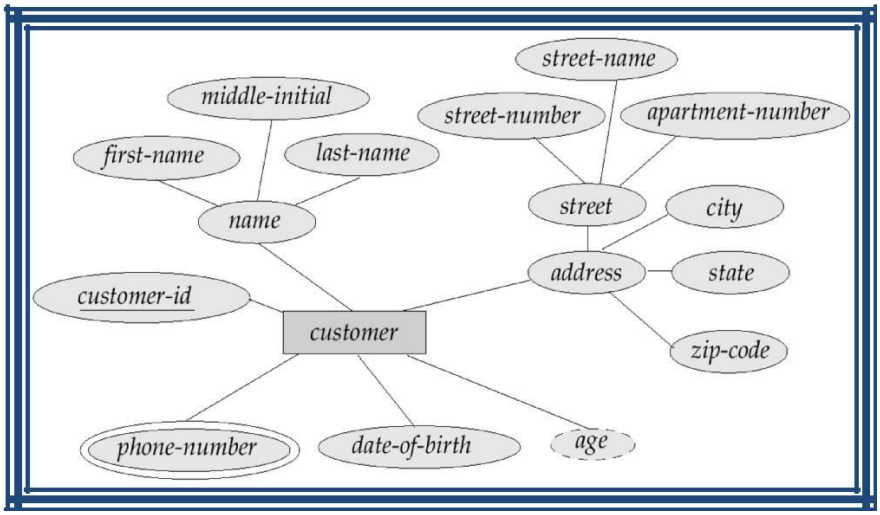
- Specification notation for defining the database schema.
- Data Definition Language (DDL) supports the creation, deletion, and modification of tables.
- The DDL basically consists of **CREATE TABLE, DROP TABLE and ALTER TABLE** command.
- Also it allows users define new domains, analogous to type definition commands in a programming language.
- DDL compiler generates a set of tables stored in a data dictionary
- Data dictionary contains metadata (i.e., data about data)
  - Database schema
  - Data storage & definition language : Specifies the storage structure and access methods used
  - Integrity constraints : Domain constraints, Referential integrity
  - Authorization

Data Manipulation Language (DML)

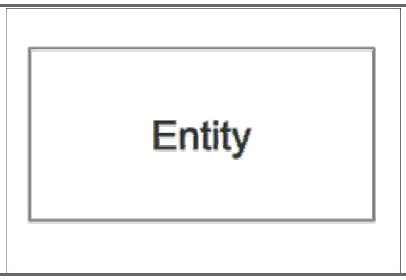
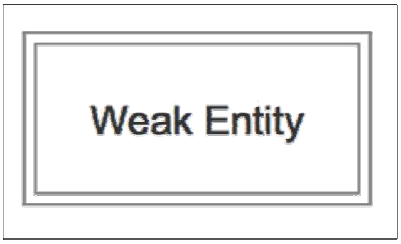
- Language for accessing and manipulating the data organized by the appropriate data
- model Two classes of languages
  - **Procedural** – user specifies what data is required and how to get those data
  - **Declarative (nonprocedural)** – user specifies what data is required without specifying how to get those data
- DML basically consist of **INSERT, DELETE & UPDATE** command.
- **SQL is the most widely used query language.**
- The main purpose of SQL is data access control; it helps us to retrieve the data from Database according to our requirements.

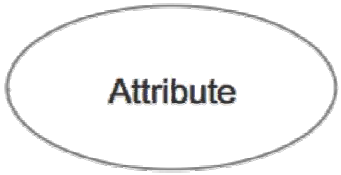
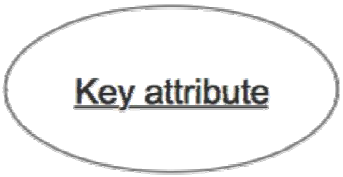
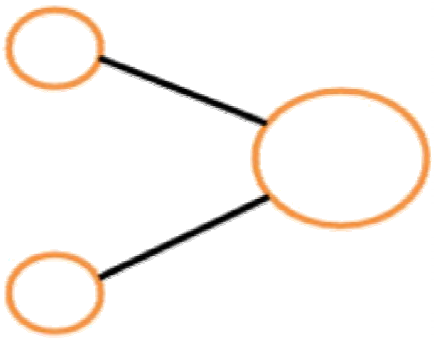
Entity Relationship (E R) Model

- The **Entity Relationship** (ER) model is one of several high-level, or semantic, data models used in database design. The goal is to create a simple description of the data that closely matches how users and developers think of the data
- A database can be modeled as : a collection of entities, relationship among entities.
- **An Entity** is real-world object that exists and is distinguishable from other objects.
- **A relationship** is an association among several (Two or more) entities.
- Entities are represented by means of their properties, called **attributes.**
- An entity set is a set of entities of the same type that share the same properties.
- Each entity set has a Key.
- Each Attribute has a Domain.
- **Types of Attributes**
- **Simple attribute** − Simple attributes are atomic values, which cannot be divided further.
- For example, a Customer's ID number is an atomic value of 6 digits.
- **Composite attribute** − Composite attributes are made of more than one simple attribute.
- For example, a customer's complete name may have first-name, middle-initial and last-name.
- **Single-value attribute** − Single-value attributes contain single value.
- For example − Customer_ID, Social_Security_Number.
- **Multi-value attribute** − Multi-value attributes may contain more than one values.
- For example, a person can have more than one phone number, email_address, etc.
- **Derived attribute** − Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example,
- age can be derived from date_of_birth.



E-R diagrams constructs (Symbols used in E-R Diagrams)

Symbol	Shape Name	Symbol Description
Entities		
	Entity	An entity is represented by a rectangle which contains the entity's name.
	Weak Entity	An entity that cannot be uniquely identified by its attributes alone. The existence of a weak entity is dependent upon another entity called the owner entity. The weak entity's i a combination of the identifier of the owner entity and the partial key of the weak entity.
Attributes		
Relationships		In the ER notation, each attribute is represented by an
	Strong relationship	A relationship where entity is existence-independent of other entities and PK of Child do contain PK component of Parent Entity. A strong relationship is represented by a single rhombus
	Weak (identifying) relationship	A relationship where Child entity is existenceunderli ed. PK of Child Entity contains PK component of Parent Entity. This relationship is represented by a double rhombus.
	Key attribute	An attribute that uniquely identifies a particular entity.
	Multivalued attribute	An attribute that can have many values (there are many distinct values entered for it in the same column of the table). Multivalued attribute is depicted by a dual oval.
	Derived attribute	An attribute whose value is calculated (derived) from other attributes. The derived attribute may or may not be physically stored in the database. In the ER notation, this attribute is represented by dashed oval.

Attributes		
	Attribute	In the ER notation, each attribute is represented by an oval containing attribute' name
	Key attribute	An attribute that uniquely identifies a particular entity. The name of a key attribute is underlined.
	Multivalued attribute	An attribute that can have many values (there are many distinct values entered for it in the same column of the table). Multivalued attribute is depicted by a dual oval.
	Derived attribute	An attribute whose value is calculated (derived) from other attributes. The derived attribute may or may not be physically stored in the database. In the ER notation, this attribute is represented by dashed oval.
	Composite attribute	An attribute that can have many sub attributes

Relational Model

- The relational model is very simple and elegant: A **Database** is a collection of one or more relations, where each relation is a table with rows and columns.
- The main construct for representing data in the relational model is a relation.
- The relational model is by far the dominant data model and the foundation for the leading DBMS products, including FoxBase, Paradox, Oracle, IBM's DB2 family, Informix, Sybase, Microsoft's Access, SQL Server, MYSQL, SQLite, Hadoop, SKYSQL, etc. A **Relation** consists
- of a relation schema and a relation instance.
- **Relation Schema** of a relation consists of
  - Relation's name
  - Each attribute definition consist of
    - Name of each field (or Column or Attribute)
    - Type/Domain
  - Integrity Constraints
- The **Relation Instance** is a table, and the relation schema describes the column heads for the table.
- An instance of a relation is a set of tuples, also called records, in which each tuple has the same number of fields as the relation schema.
- A relation instance can be thought of as a table in which each tuple is a row, and all rows have the same number of fields.
- The **Degree**, also called **Arity**, of a relation is the number of fields.
- The **Cardinality** of a relation instance is the number of tuples in it.

Experiment 1

Experiment 1: Working with ER Diagram and Normalization

Example: ER Diagram for Sailors Database

Entities:

1. Sailor
2. Boat

Relationship: Reserves

Primary Key Attributes:

1. SID (Sailor Entity)
2. BID (Boat Entity)

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

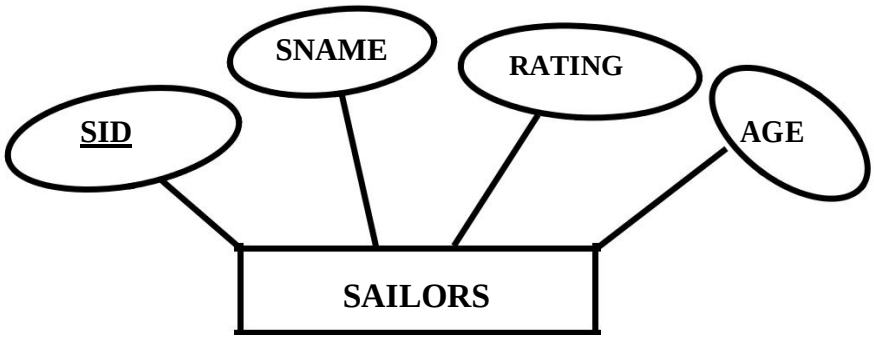
- **SAILORS (SID:INTEGER, SNAME:STRING, RATING:INTEGER, AGE:REAL)**
- **BOATS (BID:INTEGER, BNAME:STRING, COLOR:STRING)**
- **RESERVES (SID:INTEGER, BID:INTEGER, DAY:DATE)**

ER DIAGRAM

Entities:

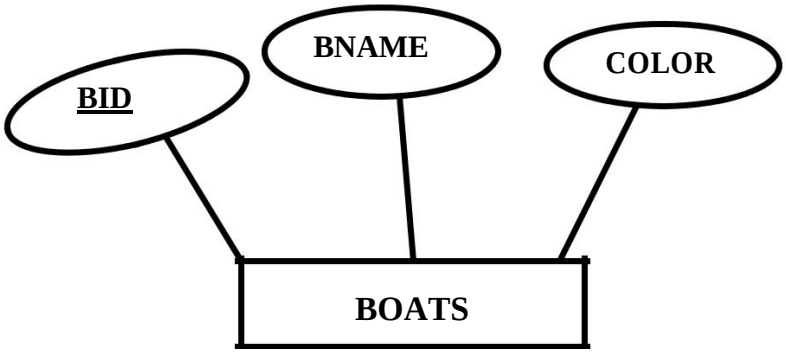
1. SAILORS:

SID
SNAME
RATING
AGE



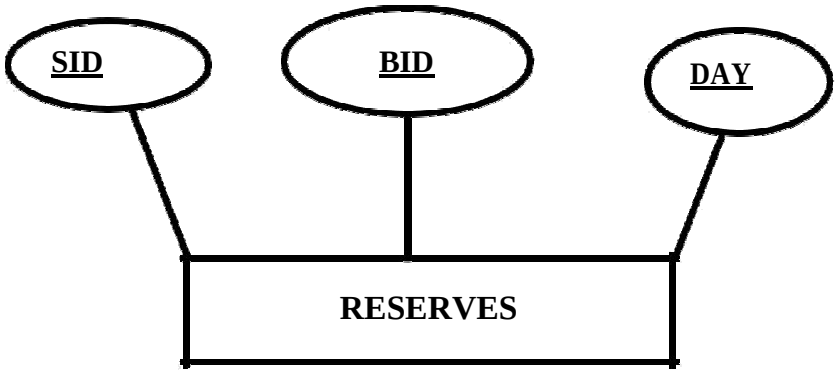
2. BOATS

BID
BNAME
COLOR

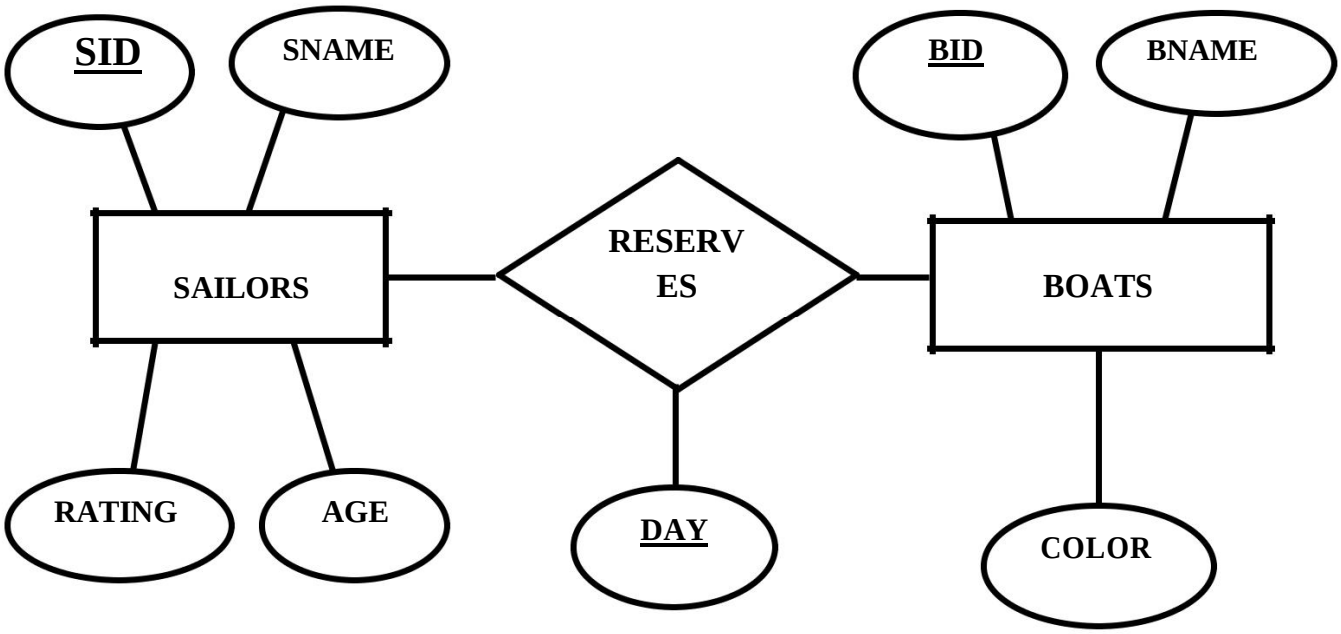


RELATIONSHIP: RESERVES

SID
BID
DAY



Primary Key Attributes
1. SID (SAILORS ENTITY)
2. BID (BOATS ENTITY)
3. SID, BID, DAY (RESERVES RELATIONSHIP)



Experiment 2

Experiment 2: Working with DDL, DML, DCL and Key Constraints

Creation, Altering and Dropping of Tables and Inserting Rows into a Table (Use Constraints While Creating Tables). Examples Using Select Command.

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

- **CHAR (SIZE)** :- The CHAR data type stores fixed-length character strings. When you create a table with a CHAR column, you must specify a string length (in bytes or characters) between 1 and 2000 bytes for the CHAR column width. The default is 1 byte.
- **VARCHAR2 (SIZE)** :- The VARCHAR2 data type stores variable-length character strings. When you create a table with a VARCHAR2 column, you specify a maximum string length (in bytes or characters) between 1 and 4000 bytes for the VARCHAR2 column.
- **VARCHAR (SIZE)** :- The VARCHAR data type is synonymous with the VARCHAR2 data type.
- **NUMBER (L)** :- The NUMBER datatype stores fixed and floating-point numbers. Numbers of virtually any magnitude can be stored and are guaranteed portable among different systems operating Oracle Database, up to 38 digits of precision.
- **NUMBER (L, D)** :- Numeric data with total number of digits L and number of digits D after decimal point.
- **DATE** :- The DATE datatype stores point-in-time values (dates and times) in a table. The DATE datatype stores the year (including the century), the month, the day, the hours, the minutes, and the seconds (after midnight).
- **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)
- **FLOAT :-** FLOAT is a 32-bit, single-precision floating-point number datatype. Each FLOAT value requires 5 bytes, including a length byte.
- **DOUBLE** :- DOUBLE is a 64-bit, double-precision floating-point number datatype. Each DOUBLE value requires 9 bytes, including a length byte.

Data Definition Language (DDL) Commands:-

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, …..);

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 Constraint

CREATE TABLE TABLENAME (ATT_NAME1 DATATYPE, ATT_NAME2 DATATYPE, ATT_NAME3 DATATYPE ….., **PRIMARY KEY(ATT_NAMES)**);

CREATE TABLE SAILORS (SID NUMBER(5), SNAME VARCHAR2(30), RATING NUMBER(5), AGE NUMBER(4,2), , **PRIMARY KEY(SID)**);

Creating tables with Foreign Key Constraint

CREATE TABLE TABLENAME (ATT_NAME1 DATATYPE, ATT_NAME2 DATATYPE, ATT_NAME3 DATATYPE ….., **FOREIGN KEY (ATT_NAME) REFERENCES TABLENAME2)**);

CREATE TABLE RESERVES (SID NUMBER(5), BID NUMBER(5), DAY DATE, **FOREIGN KEY (SID) REFERENCES (SAILORS)**);

Creating table with Check 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.

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)**);

Here are the complete details for creating three tables using Create table command

- **CREATE TABLE SAILORS(SID NUMBER(3) PRIMARY KEY, SNAME VARCHAR2(30), RATING NUMBER(3), AGE NUMBER(4,2), CHECK (RATING >=1 AND RATING <=10))**

Output :

- **CREATE TABLE BOATS(BID NUMBER(3) PRIMARY KEY, BNAME VARCHAR2(20), BCOLOR VARCHAR2(20))**

Output :

- **CREATE TABLE RESERVES (SID NUMBER(3), BID NUMBER(3), DAY DATE, PRIMARY KEY(SID,BID,DAY), FOREIGN KEY(SID) REFERENCES SAILORS, FOREIGN KEY(BID) REFERENCES BOATS)**

Output :

- **To see all tables present in your login**

  SELECT * FROM TAB;

Output :

- **To see schema of a particular table**

  DESC <TABLENAME>;
- Show schema of Sailors

  DESC SAILORS;

Output :

- Show schema of Boats

  DESC BOATS;

Output :

- Show schema of Reserves

  DESC RESERVES;

Output :

Deleting Table :- The table along with its definition & data can be deleted using following command.

DROP TABLE <TABLENAME>;

DROP TABLE SAILORS;

Output :

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

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

ALTER TABLE <TABLENAME> ADD COLUMN ATT_NAME DATATYPE;

ALTER TABLE SAILORS ADD COLUMN SALARY NUMBER (7,2);

Output :

Display schema after modification

DESC SAILORS;

Output :

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

ALTER TABLE <TABLENAME> DROP COLUMN ATT_NAME;

ALTER TABLE SAILORS DROP COLUMN SALARY;

Display schema after modification

DESC SAILORS;

Output :

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

ALTER TABLE <TABLENAME> ADD CONSTRAINT
<CON_NAME> <CON_DEFINITION>;

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>;

DROP CONSTRAINT RATE;

Similarly we can drop primary key or foreign key constraint.

Data Manipulation Language (DML) Commands :-

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, …..);

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, …..);

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.

Insert data in SAILORS Table

- INSERT INTO SAILORS VALUES(1,'VIJAY', 9, 36);
- INSERT INTO SAILORS VALUES(2,'RAJESH', 10, 25);
- INSERT INTO SAILORS VALUES(3,'MOHAN', 8, 23);
- INSERT INTO SAILORS VALUES(4,'KUMAR', 7, 28);
- INSERT INTO SAILORS VALUES(5,'SAGAR', 9, 21);
- INSERT INTO SAILORS VALUES(6,'MAHESH', 9, 36);

Insert data in BOATS Table

- INSERT INTO BOATS VALUES(1,'GANGA', 'RED');
- INSERT INTO BOATS VALUES(2,'JAMUNA', 'GREEN');
- INSERT INTO BOATS VALUES(3,'KAVERI', 'PINK');
- INSERT INTO BOATS VALUES(4,'GODAVARI', 'RED');
- INSERT INTO BOATS VALUES(5,'KRISHNA', 'BLUE');

Insert data in RESERVES Table

- INSERT INTO RESERVES VALUES(1,1,'12-FEB-2017');
- INSERT INTO RESERVES VALUES(1,2, '12-FEB-2017');
- INSERT INTO RESERVES VALUES(2,1,'13-FEB-2017‘);
- INSERT INTO RESERVES VALUES(3,2, '14-FEB-2017');
- INSERT INTO RESERVES VALUES(3,3,'14-FEB-2017');

To see the records :- To view all records present in the table.

SELECT * FROM <TABLENAME>

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>;

DELETE FROM SAILORS WHERE SNAME = _Rajesh‘;

Output :

To delete all records from the table

DELETE FROM <TABLENAME>;

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;

UPDATE SAILORS SET RATING = 9 WHERE SID = 1;

Output :

- To update all records without any condition.

UPDATE SAILORS SET RATING = RATING +

1; Output :

Examples using SELECT Command

Complete Syntax of SQL Queries consist of six clauses as follows.

SELECT [DISTINCT] <attribute list>
FROM <table list>
[WHERE <condition>]
[GROUP BY <grouping attribute(s)>]
[HAVING <group condition>]
[ORDER BY <attribute list>]

Note – 1) Keywords (UPPERCASE) mentioned in [ ] are optional
2) Text in < > - data need to be provided.

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query.
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order (sorting) for displaying the result of a query
- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause

Examples Using Select Command :

Show all records of Sailors

SELECT * FROM SAILORS;

Output :

- Show all records of Boats

SELECT * FROM BOATS;

Output :

- Show all records of Reserves

SELECT * FROM RESERVES;

Output :

- Find the names and ages of all sailors.

  SELECT DISTINCT S.SNAME, S.AGE FROM SAILORS S; Output :

- Find all sailors with a rating above 8.

  SELECT S.SID, S.SNAME, S.RATING, S.AGE FROM SAILORS S
  WHERE S.RATING > 8;

Same query can be written as

  SELECT * FROM SAILORS WHERE RATING > 8;

Output :

- Find sailors name with a rating above 7 & age above 25.

  SELECT SNAME FROM SAILORS S WHERE S.RATING > 7 AND S.AGE > 25;

Output :

- Display all the names & colors of the boats.

  SELECT BNAME, BCOLOR FROM BOATS;

Output :

- Find all the boats with Red color.

  SELECT * FROM BOATS WHERE BCOLOR=‘RED’;

Output :

- Find the names of sailors who have reserved boat number 123.

```
SELECT S.SNAME FROM SAILORS S, RESERVES R
WHERE S.SID = R.SID AND R.BID = 123;
```

Output :

- Find the names of sailors ' who have reserved Red boat.

```
SELECT S.SNAME FROM SAILORS S, RESERVES R, BOATS B WHERE
S.SID = R.SID AND R.BID = B.BID AND B.BCOLOR = 'RED' ;
```

Output :

- Find the colors of boats reserved by Rajesh.

```
SELECT B.BCOLOR FROM SAILORS S, RESERVES R, BOATS B WHERE S.SID
= R.SID AND R.BID = B.BID AND S.SNAME = ‘RAJESH' ;
```

Output :

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

```
SELECT DISTINCT S.SNAME FROM SAILORS S, RESERVES R WHERE S.SID
= R.SID;
```

Output :

- Find names of the sailors who have reserved two different boats.

```
SELECT DISTINCT S.SNAME FROM SAILORS S, RESERVES R1, RESERVES R2
WHERE S.SID = R1.SID AND S.SID = R2.SID AND R1.BID <> R2.BID
```

Output :

Data Control Language (DCL) Commands:

Data Control Language (DCL) is used to control privileges in Database. To perform any operation in the database, such as for creating tables and views - a user needs privileges.

Privileges are of two types

System: This includes permissions for creating session, tables, etc and all types of other system privileges.

Object: This includes permissions for any command or query to perform any operation on the database tables.

DCL have two commands,

GRANT: Used to provide any user access privileges or other privileges for the database.

REVOKE: Used to take back permissions from any user.

Allow a User to create session : When we create a user in SQL, it is not even allowed to login and create a session until and unless proper permissions/privileges are granted to the user. Following command can be used to grant the session creating privileges.

GRANT CREATE SESSION TO username;

Allow a User to create table : To allow a user to create tables in the database, we can use the below command,

GRANT CREATE TABLE TO username;

Grant permission to drop any table : If you want to allow user to drop any table from the database, then grant this privilege to the user,

GRANT DROP ANY TABLE TO username

To take back Permissions : If you want to take back the privileges from any user, use the REVOKE command.

REVOKE CREATE TABLE FROM username

Experiment 3

Experiment 3: Working with Queries and Nested QUERIES

Queries (along with sub Queries) using ANY, ALL, IN, EXISTS, NOTEXISTS, UNION, INTERSET, MINUS.

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

Ex:- Find the Names and Ages of all sailors.

SELECT DISTINCT S.SNAME, S.AGE FROM SAILORS S;

Output :

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-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 : 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 data 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.

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');

Output :

Same query can be written using **UNION** as follows.

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.

Output :

- Find all sids of sailors who have a rating of 10 or reserved boat number 1.

SELECT S.SID FROM SAILORS S WHERE S.RATING = 10
UNION
SELECT R.SID FROM RESERVES R WHERE R.BID = 1;

Output :

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.

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';

Output :

Same query can be written using **INTERSECT** as follows.

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';

Output :

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

Ex:- Find the SID of all sailors who have reserved red boats but not green boats.

```
SELECT R1.SID FROM BOATS B1, RESERVES R1 WHERE R1.BID = B1.BID AND
B1.COLOR = 'RED'
MINUS
SELECT R2.SID FROM BOATS B2, RESERVES R2 WHERE R2.BID = B2.BID AND
B2.COLOR = ‘GREEN’;
```

Output :

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 using IN Operator.

```
SELECT S.SNAME FROM SAILORS S WHERE S.SID
IN (SELECT R.SID FROM RESERVES R WHERE R.BID = 103 );
```

Output :

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 using NOT IN Operator.

```
SELECT S.SNAME FROM SAILORS S WHERE S.SID
NOT IN (SELECT R.SID FROM RESERVES R WHERE R.BID = 103 );
```

Output :

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 using EXISTS Operator.

```
SELECT S.SNAME FROM SAILORS S WHERE
EXISTS (SELECT * FROM RESERVES R WHERE R.BID = 103 AND R.SID = S.SID );
```

Output :

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 using NOT EXISTS Operator.

SELECT S.SNAME FROM SAILORS S WHERE NOT EXISTS (SELECT * FROM RESERVES R WHERE R.BID = 103 AND R.SID = S.SID); Output :

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 using ANY Operator.

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.

Output :

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 Operator.

SELECT S.SID FROM SAILORS S WHERE S.RATING >= ALL (SELECT S2.RATING FROM SAILORS S2)

Output :

Experiment 4

Experiment 4: Working with Queries USING Aggregate Operators & views
Queries using Aggregate Functions (COUNT, SUM, AVG, MAX and MIN), GROUP BY, HAVING and Creation and Dropping of Views

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 **SELECT**
COUNT (SID) FROM SAILORS; Output :

2) To count numbers of boats booked in Reserves table.
SELECT COUNT (DISTINCT BID) FROM
RESERVES; Output :

3) To count number of Boats in Boats table.
SELECT COUNT (*) FROM BOATS;
Output :

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 **SELECT SUM (RATING)**
FROM SAILORS; Output :

2) To find sum of distinct age of Sailors (Duplicate ages are eliminated).
SELECT SUM (DISTINCT AGE) FROM
SAILORS; Output :

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. **SELECT AVG (AGE) FROM SAILORS; Output :**

2) To find average of distinct age of Sailors (Duplicate ages are eliminated).
SELECT AVG (DISTINCT AGE) FROM SAILORS; Output :

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

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

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.
Query :

Output :

2) Count the number of different sailor names.
Query :

Output :

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

Output :

4) Count the number of Sailors.
Query :

Output :

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

Output :

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.
SELECT * FROM SAILORS ORDER BY AGE;
Output :

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

3) Displays all the sailors according to rating, if rating is same then sort according to age.

SELECT * FROM SAILORS ORDER BY RATING, AGE;

Output :

Write the query

1) To display names of sailors according to alphabetical order.

Query :

Output :

2) Displays all the sailors according to rating (Topper First), if rating is same then sort according to age (Older First).

Query :

Output :

3) Displays all the sailors according to rating (Topper First), if rating is same then sort according to age (Younger First).

Query :

Output :

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

Query :

Output :

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.

SELECT S.RATING, MIN (S.AGE) FROM SAILORS S GROUP BY S.RATING; Output :

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.

SELECT S.RATING, MIN (S.AGE) AS MINAGE FROM SAILORS
S WHERE S.AGE >= 18
GROUP BY S.RATING HAVING COUNT (*) > 1;
Output :

Write following queries in SQL.

1) For each red boat; find the number of reservations for this boat.

Query :

Output :

2) Find the average age of sailors for each rating level that has at least two sailors.
Query :

Output :

3) Find those ratings for which the average age of sailors is the minimum overall ratings.
Query :

Output :

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 an Expert Sailor if his rating is more than 8).
CREATE VIEW EXPERTSAILOR AS SELECT SID, SNAME, RATING FROM SAILORS WHERE RATING > 9;

Now on this view we can use normal SQL statements as we are using on Base tables.
Eg:- Find average age of Expert sailors.
SELECT AVG (AGE) FROM EXPERTSAILOR; Output :

Write the following queries on Expert Sailor View.

- Find the Sailors with age > 25 and rating equal to 10.

Query :

Output :

- Find the total number of Sailors in Expert Sailor view.

Query :

Output :

- Find the number of Sailors at each rating level (8, 9, 10).

Query :

Output :

- Find the sum of rating of Sailors.

Query :

Output :

- Find the age of Oldest as well as Youngest Expert Sailor.

Query :

Output :

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.

DROP VIEW EXPERTSAILOR;

Experiment 5

Experiment 5: Working with Conversion Functions & String Functions Queries using

- Conversion Functions (TO_CHAR, TO_NUMBER AND TO_DATE),
- String Functions (CONCAT, LPAD, RPAD, LTRIM, RTRIM, LOWER, UPPER, INITCAP, LENGTH, SUBSTR AND INSTR),
- Date Functions (SYSDATE, NEXT_DAY, ADD_MONTHS, LAST_DAY, MONTHS_BETWEEN)
- LEAST, GREATEST, TRUNC, ROUND

Conversion Functions :- These functions are used to convert the value from one type to another type.

1) TO_CHAR (N [,FMT]) :- Converts _N‘ of numeric data type to Varchar2 datatype using optional number format FMT.

TO_CHAR (N [,FMT])

Example 1) Convert 12345 to string.

SELECT TO_CHAR (12345) FROM DUAL;

Output :

2) Display system date after converting to varchar2 data type.

SELECT TO_CHAR (SYSDATE) FROM DUAL; Output :

3) Display system date in _MON-DD-YYYY‘ format after converting to varchar2 data type.

SELECT TO_CHAR (SYSDATE, ‘MON-DD-YYYY’) FROM DUAL; Output :

2) TO_NUMBER (CHAR) :- This conversion function is used to convert string to number data type.

Ex :- Convert string _123.45‘ to number data type.

SELECT TO_NUMBER (‘123.45’) FROM DUAL;

Output :

3) TO_DATE :- Converts character data type data to date type data.
Ex:- Display '09-02-2010' converted to DDD-MM-YY format using to_char & to_date functions.
SELECT TO_CHAR (TO_DATE ('09-02-2010', 'DD-MM-YYYY'),('DDD-MM-YY') FROM DUAL;
Output :

Formats we can use for Date :- Following formats can be used with any function related to dates.

- **Y** → Last digit in the year.
- **YY** → Last two digits in the year.
- **YYYY** → Year in four digits.
- **YEAR** → Year in characters.
- **MM** → Month in digits.
- **MONTH** → Month in characters.
- **MON** → Month in formatted with three characters.
- **D** → Day of Week.
- **DD** → Day of the Month.
- **DDD** → Day of the Year.
- **DAY** → Day name.
- **HH** or **HH12** or **HH24** → Hours in 12 Hrs or 24 Hrs format.
- **MI** → Minutes.
- **SS** → Seconds.

- To display today's date using all formats

SELECT TO_CHAR (SYSDATE,'D-DD-DDD-DAY-MM-MON-MONTH-Y-YY-YYYY-YEAR-HH-HH24-MI-SS') FROM DUAL;

Output :

- To display 27-02-2018 date using all formats

SELECT TO_CHAR (TO_DATE('27-02-2018', 'DD-MM-YYYY'),'D-DD-DDD-DAY-MM-MON-MONTH-Y-YY-YYYY-YEAR-HH-HH24-MI-SS') FROM DUAL;

Output :

String Functions :-

1) Concatenation :- Concatenates two stings from a given list.

CANCAT (CHAR1, CHAR2)

EX 1) Concate the string _Rajesh‘ with _Raghu‘

SELECT CONCAT (‘Rajesh’, ‘Raghu’) FROM DUAL;

Output :

2) Concat bid & bname of Boats & display along with color.

SELECT CONCAT (BID, BNAME), COLOR FROM BOATS;

Output :

2) LPAD (CHAR1, N, CHAR2) :- Returns CHAR1 left padded to length _N‘ with sequence of characters in CHAR2. The default value of CHAR2 is a single blank space.

Ex 1) Lpad the string _Rajesh‘ to length 30 with the set of characters in string _-*-‘

SELECT LPAD (‘Rajesh’, 30, ‘-*-’) FROM DUAL;

Output :

2) Lpad the string bname to length 20 with _-*‘ set of characters and string color by _/‘.

Query :

Output :

3) RPAD (CHAR1, N, CHAR2) :- Returns CHAR1 right padded to length _N‘ with sequence of characters in CHAR2. The default value of CHAR2 is a single blank space.

Ex 1) Rpad the string _Rajesh‘ to length 30 with the set of characters in string _*#‘

SELECT RPAD (‘Rajesh’, 30, ‘*#’) FROM DUAL;

Output :

2) Rpad the string sname to length 25 with _-*‘ set of characters and remaining attributes in normal way.

Query :

Output :

3) Rpad the string bname to length 20 with _-*‘ set of characters and lpad the same to make it length 30 with _#‘ and remaining attributes in normal way.
Query :

Output :

4) LTRIM (CHAR, SET) :- Returns characters from the left of CHAR by deleting all leftmost characters that appear in set.
Ex:- 1) Display all sailors information by removing characters of sname if starts with _R‘.
SELECT SID, LTRIM (SNAME,’R’) FROM SAILORS; Output :

2) Display all sailors information by removing characters of boat name if starts with _T‘
Query :

Output :

5) RTRIM (CHAR, SET) :- Returns characters from the right of CHAR by deleting all rightmost characters that appear in set.
Ex:- 1) Display all sailors information by removing characters of sname if ends with _i‘.
SELECT SID, RTRIM (SNAME,’i’) FROM SAILORS;
Output :

2) Display all Boats information by removing characters of color if ends with _d‘.
Query :

Output :

6) LOWER(CHAR) :- Converts all characters to lowercase characters in a sting CHAR.
Ex:- 1) Display all Boats information by showing their names in lower case.
SELECT BID, LOWER (BNAME), COLOR FROM BOATS;
Output :

2) Display all Sailors information by showing their names in lower case.
Query :

Output :

7) UPPER(CHAR) :- Converts all characters to uppercase characters in a sting CHAR.
Ex:- 1) Display all Sailors information by showing their names in Upper case.
SELECT SID, UPPER (SNAME), AGE, RATING FROM SAILORS; Output :

2) Display all Boats information by showing their color in Upper case.
Query :

Output :

8) INITCAP(CHAR) :- Converts first character of each word in a sting CHAR to uppercase.
Ex:-1) Display all Sailors information by showing their names in Capitalizing first char.
SELECT SID, INITCAP (SNAME), AGE, RATING FROM SAILORS;
Output :

2) Capatilize first letter of each word in ‗rajesh raghu'
SELECT INITCAP ('rajesh raghu') FROM DUAL; Output :

9) LENGTH (CHAR) :- Returns the length of the string CHAR i.e. number of characters present in the given string.
Ex:-1) Find the number of characters in the string ‗Information Technology'
SELECT LENGTH ('Information Technology') FROM DUAL; Output :

2) Display length of string SID, SNAME from Sailors along with their values.
SELECT SID, LENGTH (SID), SNAME, LENGTH (SNAME) FROM SAILORS; Output :

10) SUBSTR (CHAR, M, N) :- It returns substring from CHAR string starting with index M & gives N characters.
Ex : Display boats information by starting their names with 3rd character & show only 4 characters.
SELECT BID, SUBSTR (BNAME, 3, 4), COLOR FROM BOATS; Output :

11) INSTR (CHAR1, CHAR2, M, N) :- It searches CHAR1 beginning with Mth character for Nth occurrence of CHAR2 and returns the position after character in CHAR1.
If N is negative value, then it searches backwards from the end of
CHAR1. The default value of M & N is 1.
Ex : Display the index of string _AB' after 2nd character & 3rd occurrence in the given string _ABCDABCDABABAB'.
SELECT INSTR ('ABCDABCDABABAB','AB', 2, 3) FROM DUAL;
Output :

12) TRANSLATE (CHAR, FROM, TO) :- It returns characters with all occurrences of each character in FROM replaced by its corresponding character in TO.
Ex :1) Replace _A' with _D' in the given string _ABCDABCDABABAB'.
SELECT TRANSLATE ('ABCDABCDABABAB','A','B') FROM DUAL; Output :

2) Display Sailors information by replacing _A' with _I' from SNAME, if any.
SELECT SID, TRANSLATE (SNAME,'A','I') FROM SAILORS;
Output :

13) REPLACE (CHAR, S, R) :- It returns characters with every occurrences of S replaced with R. If R is not given or NULL, all occurrences of S are removed or deleted.

Ex :1) Display BNAME by replacing _DA‘ with _MA‘.
SELECT REPLACE (BNAME, ‘DA’, ‘MA’) FROM BOATS;
Output :

Date Functions :-
1) SYSDATE :- Displays the system date for a system.
SELECT SYSDATE FROM
DUAL; Output :

SELECT TO_CHAR (SYSDATE, 'DD-MON-YYYY HH:MI:SS') FROM dual;
Output :

2) NEXT_DAY (D, DAY) :- Displays next date on DAY after date D.
Ex: Display date on Thu after 20th Feb, 2018.
SELECT NEXT_DAY (’20-FEB-2018’, ‘THU’) FROM DUAL;
Output :

3) ADD_MONTHS (D, N) :- Returns a date after adding a specified day D with specified number of months N.
Ex: Display SID, Day of Reservation by adding 20 months to given day.
SELECT SID, DAY, ADD_MONTHS (DAY, 20) FROM RESERVES;
Output :

4) LAST_DAY(D) :- Returns the date corresponding to last day of the month.
Ex: Display Sname, Day of Reservation and date corresponding to last date of the month.
SELECT S.SNAME, DAY, LAST_DAY (DAY) FROM SAILORS S, RESERVES R WHERE S.SID = R.SID;
Output :

5) MONTHS_BETWEEN (D1, D2) :- Returns number of months between given two dates D1 & D2.

Ex: Display SID, Day of Reservation and months between System Date & day of reservation.

SELECT SID, DAY, MONTHS_BETWEEN (SYSDATE, DAY) FROM RESERVES;

Output :

6) LEAST (EXPR1 [, EXPR2, ... EXPR_N]) :- Returns the smallest value in a list of expressions.

Ex : 1) Find least value in 12, 53, 17, 2.

SELECT LEAST (12, 53, 17, 2) FROM
DUAL; Output :

2) Find least value in '12', '53', '17', '2'

Select LEAST ('12', '53', '17', '2') from dual;

Output :

3) Find least value in 'APPLES', 'ORANGES', and _BANANAS‘

SELECT LEAST ('APPLES', 'ORANGES', 'BANANAS') FROM DUAL;

Output :

7) GREATEST (EXPR1 [, EXPR2, ... EXPR_N]) :- Returns the largest value in a list of expressions

Ex : 1) Find least value in 12, 13, 17, 2.

SELECT GREATEST (12, 13, 17, 2) FROM
DUAL; Output :

2) Find least value in '12', '13', '17', '2'

SELECT GREATEST ('12', '13', '17', '2') FROM DUAL;

Output :

3) Find least value in 'APPLES', 'ORANGES', 'BANANAS'

SELECT GREATEST ('APPLES', 'ORANGES', 'BANANAS') FROM DUAL;

Output :

8) TRUNC (NUMBER [, DECIMAL_PLACES]) :- returns a number truncated to a certain number of decimal places.

Ex :- Truncate the 5632.98345 number to 0, 1, 2 decimal places.

SELECT TRUNC (5632.98345) FROM DUAL;

or

SELECT TRUNC (5632.98345, 0) FROM DUAL;

Output :

SELECT TRUNC (5632.98345, 1) FROM DUAL;

Output :

SELECT TRUNC (5632.98345, 2) FROM DUAL;

Output :

9) ROUND (NUMBER [, DECIMAL_PLACES]) :- Returns a number rounded to a certain number of decimal places.

Ex :- Round off the 5632.98345 number to 0, 1, 2 decimal places.

SELECT ROUND (5632.98345) FROM DUAL;

or

SELECT ROUND (5632.98345, 0) FROM DUAL;

Output :

SELECT ROUND (5632.98345, 1) FROM DUAL;

Output :

SELECT ROUND (5632.98345, 2) FROM DUAL;

Output :

Experiment 6

Experiment 6 : Working with Triggers using PL/SQL

Develop Programs using BEFORE and AFTER Triggers, Row and Statement Triggers and INSTEAD OF Triggers

Trigger :- A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA..

A database that has a set of associated triggers is called an Active Database.

A trigger description contains three parts:

- **Event :** A change to the database that activates the trigger.
- **Condition :** A query or test that is run when the trigger is activated.
- **Action :** A procedure that is executed when the trigger is activated and its condition is true.

An insert, delete, or update statement could activate a trigger, regardless of which user or application invoked the activating statement; users may not even be aware that a trigger was executed as a side effect of their program.

Trigger Examples :-

1) Trigger to convert SNAME field lowercase to uppercase - executes BEFORE INSERT.

```
CREATE OR REPLACE TRIGGER UPPERNAME
BEFORE INSERT ON SAILORS FOR EACH ROW
BEGIN
:NEW.SNAME := UPPER(:NEW.SNAME);
END;
/
```

Output :

To test whether **UPPERNAME** Trigger working properly or not, insert one record in Sailors table with **SNAME in lowercase**.

Query

Now to verify it is converted in Uppercase or not, display all records of Sailors

Query

Output

2) Trigger to restrict Deleting - This trigger prevents deleting **SID=1** row.

```
CREATE OR REPLACE TRIGGER NODELETE
AFTER DELETE ON SAILORS FOR EACH
ROW BEGIN
IF :OLD.SID = 1 THEN
RAISE_APPLICATION_ERROR (-20015, 'YOU CAN NOT DELETE THIS ROW');
END IF;
END;
/
```

Output :

To test whether NODELETE Trigger working properly or not, delete sailors record with SID=1. **Query**

Output

3) Program to indicate invalid age (if sailors age is more than 65) condition using Trigger

```
CREATE OR REPLACE TRIGGER AGELIMIT BEFORE INSERT ON SAILORS
FOR EACH ROW
WHEN (NEW.AGE>60)
BEGIN
 RAISE_APPLICATION_ERROR (-20998,'ERROR : INVALID AGE');
END;
/
```

Output :

To test whether **AGELIMIT** Trigger working properly or not, insert one record in Sailors table with sailors age more than 65.

Query

Output

Or same can be written using IF statement

```
CREATE OR REPLACE TRIGGER AGELIMIT BEFORE INSERT ON SAILORS
FOR EACH ROW
BEGIN
IF (:NEW.AGE>60) then
 RAISE_APPLICATION_ERROR (-20998,'ERROR : INVALID AGE');
END IF;
END;
/
```

Output :

To test whether **AGELIMIT** Trigger working properly or not, insert one record in Sailors table with sailors age more than 65.

Query

Output

- To see all user defined triggers details (all parameters of trigger)

SELECT * FROM USER_TRIGGERS;

Output :

- To see all user defined trigger names

SELECT TRIGGER_NAME FROM USER_TRIGGERS;

Output :

- To See particular trigger‘s definition

SELECT DESCRIPTION, TRIGGER_BODY FROM USER_TRIGGERS WHERE TRIGGER_NAME = 'AGELIMIT';

Output :

- To see different parameters of triggers

SELECT TRIGGER_NAME, TRIGGER_TYPE, TRIGGERING_EVENT, TRIGGER_BODY FROM USER_TRIGGERS;

Output :

- To delete particular user defined trigger

DROP TRIGGER AGELIMIT

Output :

Experiment 7

Experiment 7 : Working with PL/SQL Procedures

Programs Development using Creation of Procedures, Passing Parameters IN and OUT of Procedures

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;
```

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

  SET SERVEROUTPUT ON

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;
/
```

Output :

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) Using IF-

```
ELSE 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;
/
```

Output :

2) Using IF—ELSIF—ELSE

```
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;
/
```

Output :

Procedure in PL/SQL

Procedures are written for doing specific tasks. The general syntax of procedure is

CREATE OR REPLACE PROCEDURE <Pro_Name> (Par_Name1 [IN / OUT/ IN OUT] Par_Type1, ….) IS (Or we can write **AS**)
Local declarations;
BEGIN
PL/SQL Executable statements;
..
..
..
EXCEPTION
Exception Handlers;
END <Pro_Name>;

Mode of parameters

1) **IN Mode :-** IN mode is used to pass a value to Procedure/Function. Inside the procedure/function, IN acts as a constant and any attempt to change its value causes compilation error.
2) **OUT Mode :** The OUT parameter is used to return value to the calling routine. Any attempt to refer to the value of this parameter results in null value.
3) **IN OUT Mode :** IN OUT parameter is used to pass a value to a subprogram and for getting the updated value from the subprogram.

- **To use/call procedure, write a PL/SQL code and include call in the code using**
  Pro_Name(Par_List);
  Or you can execute from SQL Prompt as
  execute Pro_Name(Par_List)

- **For dropping/deleting Procedure**

DROP PROCEDURE Pro_Name;

Examples

1) Simple program to illustrate Procedure.

```
-- Assume file name P1
CREATE OR REPLACE PROCEDURE P1(A NUMBER)
AS BEGIN
DBMS_OUTPUT.PUT_LINE('A:'||A);
END P1;
/
```

Output :

Now write PL/SQL code to use procedure in separate file.

```
-- Assume file name testP1
DECLARE
BEGIN
P1(100);
END;
/
```

Output :

2) Program to illustrate Procedure with IN mode parameter. -- Assume file name P2

```
CREATE OR REPLACE PROCEDURE P2(A IN NUMBER) AS
BEGIN
DBMS_OUTPUT.PUT_LINE('A:'||A);
END P2;
/
```

Output :

-- Assume file name testP2

```
DECLARE X
NUMBER;
BEGIN
X:=10;
DBMS_OUTPUT.PUT_LINE('X:'||X);
P2(X);
DBMS_OUTPUT.PUT_LINE('X:'||X);
END;
/
```

Output :

3) Program to illustrate Procedure with OUT mode parameter.

```
-- Assume file name P3
CREATE OR REPLACE PROCEDURE P3(A OUT NUMBER) AS
 BEGIN
 A:=100;
 DBMS_OUTPUT.PUT_LINE('A:'||
 A); END P3;
 /
```

Output :

```
-- Assume file name testP3
DECLARE X
NUMBER;
BEGIN
X:=50;
DBMS_OUTPUT.PUT_LINE('X:'||X);
P3(X);
DBMS_OUTPUT.PUT_LINE('X:'||X);
END;
/
```

Output :

4) Program to illustrate Procedure with OUT mode parameter.

```
-- Assume file name P4
CREATE OR REPLACE PROCEDURE P4(A OUT NUMBER) AS
BEGIN
DBMS_OUTPUT.PUT_LINE('A:'||A);
END P4;
/
```

Output :

```
-- Assume file name testP4
DECLARE X
NUMBER;
BEGIN
X:=10;
DBMS_OUTPUT.PUT_LINE('X:'||X);
P4(X);
DBMS_OUTPUT.PUT_LINE('X:'||X);
END;
/
```

Output :

5) Program to illustrate Procedure with IN OUT mode parameter. --Assume file name P5

```
CREATE OR REPLACE PROCEDURE P5(A IN OUT NUMBER)
AS BEGIN
DBMS_OUTPUT.PUT_LINE ('A:' || A);
END P5;
/
```

Output :

```
-- Assume file name testP5
DECLARE X
NUMBER;
BEGIN
X:=10;
DBMS_OUTPUT.PUT_LINE ('X:'|| X);
P5(X); DBMS_OUTPUT.PUT_LINE
('X:'|| X); END;

/
```

Output :

Experiment 8

Experiment 8: Working with LOOPS using PL/SQL and Exception Handling

Program Development using WHILE LOOPS, Numeric FOR LOOPS, Nested Loops using ERROR Handling, BUILT-IN Exceptions, USE Defined Exceptions, RAISE- APPLICATION ERROR

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 to 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;
/
```

Output :

FOR LOOP EXAMPLE

```
BEGIN
 FOR I IN 1..10 LOOP
  DBMS_OUTPUT.PUT_LINE(I);
END LOOP;
END;
/
```

Output :

WHILE EXAMPLE

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

Output :

LOOP EXAMPLE

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

Output :

Write PL/SQL Program to display (Using different looping statements)

*

* *

* * *

* * * *

Program

Output :

DATA TYPES

We know basic data types in Oracle. 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;
/
```

Output :

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;
/
```

Output :

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;
/
```

Output :

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;
/
```

Output :

Error Handling using RAISE_APPLICATION_ERROR

Procedure RAISE_APPLICATION_ERROR is used to generate user-defined errors in the PL/SQL. The general syntax is

RAISE_APPLICATION_ERROR (ErrorCode, Error_Message [, TRUE/FALSE]);

The valid Error_Code is in range from –20000 to –20999. The Error_Message length is maximum 2048 bytes.

The optional third parameter TRUE indicates that error message is put in stack. If FALSE is mentioned then error replaces all previous errors.

Example to illustrate RAISE_APPLICATION_ERROR

```
-- Assume file name Raise_Application_Error
DECLARE
A  INTEGER:=&A;
B INTEGER:=&B;
C INTEGER;
BEGIN
IF(B=0)THEN
RAISE_APPLICATION_ERROR (-20001,'DIVISION BY ZERO');
ELSE
C:=A/B;
DBMS_OUTPUT.PUT_LINE ('RESULT IS :'||C);
END IF;
END;
/
```

Output :

Experiment 9

Experiment 9: Working with Functions Using PL/SQL

Program Development using Creation of Stored Functions, Invoke Functions in SQL Statements and Write Complex Functions.

Functions in PL/SQL

Similar to Procedure we can create Functions which can return one value to the calling program. The syntax of function is

CREATE OR REPLACE FUNCTION <Fun_Name> (Par_Name1 [IN/OUT/ IN OUT] Par_Type1, ….)
RETURN return_datatype IS
Local declarations;
BEGIN
PL/SQL Executable statements;
…….
EXCEPTION
Exception Handlers;
END <Fun_Name>;

- To use/call Function we have two ways.
  1) Write a PL/SQL code and include call in the code using **X=Fun_Name(Par_List);**
  2) You can execute from SQL Prompt as a select query **Select Fun_Name(Par_List) from Dual;**

- **For dropping/deleting Function**

DROP FUNCTION Fun_Name;

Ex :- Program to illustrate Function. (Finding Square of a number)

```
-- Assume file name Fun
CREATE OR REPLACE FUNCTION FUN (A NUMBER)
RETURN NUMBER IS
BEGIN
RETURN (A*A);
END FUN;
```

Output :

```
-- Assume file name testFun
DECLARE
X NUMBER:=&X;
S NUMBER;
BEGIN
S:=FUN(X);
DBMS_OUTPUT.PUT_LINE('SQUARE OF A NUMBER '|| S);
END;
```

Output :

Experiment 10

Experiment 10: Working CURSORS

Develop Programs using Features Parameters in a CURSOR, FOR UPDATE CURSOR, WHERE CURRENT of Clause and CURSOR Variables

Cursors : - Oracle uses temporary work area cursor for storing output of an SQL statement. Cursors are defined as

CURSOR C1 IS SELECT SID, SNAME, RATING, AGE FROM SAILORS;

OR

CURSOR C1 IS SELECT * FROM SAILORS;

Generally while using cursors we have to Open the cursor then extract one row (record) from the cursor using Fetch operation, do the necessary operations on the Record. After completing the work Close the cursor. But if we want to do automatic opening & closing to cursor then we can use FOR loop in cursor.

FOR loop in Cursor

The cursor FOR loop gives easy way to handle the Cursor. The FOR loop opens the Cursor, fetches rows and closes the cursor after all rows are processed. For **Eg:**

```
FOR Z IN C1 LOOP
- - - - - - -
END LOOP;
```

The cursor FOR loop declares Z as record, which can hold row, returned from cursor.

Example using Cursor

```
DECLARE
 CURSOR C1 IS SELECT * FROM SAILORS;
BEGIN
FOR Z IN C1 LOOP
DBMS_OUTPUT.PUT_LINE
(Z.SID || ' ' || Z.SNAME);
END LOOP;
END;
/
```

Output :

Same example using While

```
DECLARE
 CURSOR C1 IS SELECT * FROM SAILORS;
Z C1%ROWTYPE;
BEGIN
OPEN C1;
FETCH C1 INTO Z;
WHILE (C1%FOUND) LOOP
DBMS_OUTPUT.PUT_LINE(Z.SID || '  ' || Z.SNAME);
FETCH C1 INTO
Z; END LOOP;
CLOSE C1;
END;
/
```

Output :

Suppose we want to display all sailors‘ information according to their rating with proper heading. (eg: _Sailors with rating 1‘….. etc) then we can write program as follows.

Ex:- Display records according to rating with proper heading.

```
DECLARE
I INTEGER:=1;
CURSOR C1 IS
SELECT * FROM SAILORS ORDER BY
RATING; Z C1%ROWTYPE;
BEGIN
WHILE(I<=10)LOOP
DBMS_OUTPUT.PUT_LINE(_SAILORS WITH RATING‘||I);
FOR Z IN C1 LOOP
IF(Z.RATING=I)THEN
DBMS_OUTPUT.PUT_LINE(Z.SID||_  ‘||Z.SNAME||_  ‘||Z.AGE||_ ‘||Z.RATING);
END IF;
END LOOP;
I:=I+1;
END LOOP;
END;
/
```

Output :

Multiple cursors in a program :- We can use multiple cursors in a program.

Ex:- To display details of particular table sailors, boats, reserves according to users choice.

```
DECLARE
INPUT VARCHAR2(30):= _&INPUT‘;
CURSOR C1 IS SELECT * FROM SAILORS;
CURSOR C2 IS SELECT * FROM BOATS;
CURSOR C3 IS SELECT * FROM RESERVES;
BEGIN
IF(INPUT=‘SAILORS) THEN
DBMS_OUTPUT.PUT_LINE(_SAILORS INFORMATION:‘);
FOR Z IN C1 LOOP
DBMS_OUTPUT.PUT_LINE(Z.SID||_ ‘||Z.SNAME||_ ‘||Z.AGE||_ ‘||Z.RATING);
END LOOP;
ELSIF(INPUT=_BOATS‘)THEN
DBMS_OUTPUT.PUT_LINE(_BOATS INFORMATION:‘);
FOR X IN C2 LOOP
DBMS_OUTPUT.PUT_LINE(X.BID||_ ‘||X.BNAME||_ ‘||X.COLOR);
END LOOP;
ELSIF(INPUT=_RESERVES‘)THEN
DBMS_OUTPUT.PUT_LINE(_RESERVES INFORMATION:‘);
FOR Y IN C3 LOOP
DBMS_OUTPUT.PUT_LINE(Y.SID||_ ‘||Y.BID||_ ‘||Y.DAY);
END LOOP;
ELSE
DBMS_OUTPUT.PUT_LINE(_NO SUCH TABLE EXISTS‘);
END IF;
END;
/
```

Output :

Updating the Records :- Similar to inserting the values as well as selecting the values we can use the PL/SQL programming for updating the records in the given table.

Ex:- To update rating of sailors by 2 if rating is less than 5, by 1 if rating is >5 and doesn‘t change the rating if it is equal to 10.

```
DECLARE
CURSOR C1 IS SELECT * FROM SAILORS;
Z  C1%ROWTYPE;
BEGIN
FOR Z IN C1 LOOP
IF (Z.RATING<5) THEN
UPDATE SAILORS SET RATING=RATING+2 WHERE SID=Z.SID;
ELSIF (Z.RATING>5 AND Z.RATING<10) THEN
UPDATE SAILORS SET RATING=RATING+1 WHERE SID=Z.SID;
END IF;
END LOOP;
FOR Z IN C1 LOOP
DBMS_OUTPUT.PUT_LINE (Z.SID||_ ‘||Z.RATING);
END LOOP;
END;
/
```

Output :

Deleting the Records :- Similar to inserting and updating the values as well as selecting the values we can use the PL/SQL programming for deleting the records from the given table.

Ex:- Write a program to delete records from sailors table by reading SID from Keyboard.

```
DECLARE
BEGIN
DELETE FROM SAILORS WHERE SID=_&SID‘;
END;
/
```

Output :

Passing parameters to Cursor

We can pass parameters to cursor. When you open cursor the value is passed to cursor and processing can be done there in cursor definition.

Ex:- suppose we want to display all sailors information according to their rating with proper heading. (eg: _Sailors with rating 1‘….. etc). Already we have seen same program with parameters to the cursor.

Following program illustrates how we can pass parameters to the cursor.

```
--Assume file name Cur_Par
DECLARE
CURSOR C1(R NUMBER) IS SELECT * FROM SAILORS WHERE RATING=R;
I INTEGER;
BEGIN
FOR I IN 1..10 LOOP
DBMS_OUTPUT.PUT_LINE('SAILORS WITH RATING '|| I || ' ARE');
DBMS_OUTPUT.PUT_LINE('SID NAME AGE'); FOR
Z IN C1(I) LOOP
/* It‘s not compulsory to define variable using rowtype for simple cursor as well as for
update cursor */
DBMS_OUTPUT.PUT_LINE(Z.SID ||' ' ||Z.SNAME ||' '||Z.AGE);
END LOOP;
END LOOP;
END;
/
```

Output :

Use of Cursor for Update

(FOR UPDATE CURSOR, WHERE CURRENT of clause and CURSOR variable is used).

We can use update cursors for update operation only. The following example shows change of rating (Already we have written this program without using Update Cursor) using Update Cursor.

```
DECLARE
CURSOR C1 IS SELECT * FROM SAILORS FOR UPDATE;
BEGIN
FOR Z IN C1 LOOP
/* It‘s not compulsory to define variable using rowtype for update cursor as well as for
simple cursors */
IF(Z.RATING <=5) THEN
UPDATE SAILORS SET RATING= RATING+2 WHERE CURRENT OF
C1; ELSIF(Z.RATING>5 AND Z.RATING<10)
UPDATE SAILORS SET RATING= RATING+1 WHERE CURRENT OF C1;
END IF;
END LOOP;
END;
/
```

Output :