**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**IES-IPS ACADEMY, INDORE**



SESSION: 2015-2016

**Subject Code - CS-503**

**Subject Name – Database Management System**

**CONTENTS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S.NO | LIST OF EXPRIEMENTS | PAGE NO. | Date of Experiment | Date of Submission | Remark |
| 1 | SQL & SQL \* PLUS INTRODUCTION |  |  |  |  |
| 2 | data definition LANGUAGE (ddl) Commands |  |  |  |  |
| 3 | data manipulation language (dml) Commands |  |  |  |  |
| 4 | DRL (DATA RETRIEVAL LANGUAGE) Commands |  |  |  |  |
| 5 | TRANSACTIONAL CONTROL LANGUAGE (T.C.L) Commands |  |  |  |  |
| 6 | DATA CONTROL LANGUAGE (D.C.L) Commands |  |  |  |  |
| 7 | INTEGRITY CONSTRAINTS |  |  |  |  |
| 8 | JOINS |  |  |  |  |
| 9 | SuBQUERY, VIEWS |  |  |  |  |
| 10 | PL/SQLINTRODUCTION(PL/SQL TO FIND ADDITION OF TWO NUMBERS, PL/SQL TO FIND AREA OF CIRCLE ) |  |  |  |  |
| 11 | PROCEDURE |  |  |  |  |
| 12 | TRIGGER |  |  |  |  |

**Experiment NUmber- 1.** SQL & SQL \* PLUS INTRODUCTION

Oracle has many tools such as SQL \* PLUS, Oracle Forms, Oracle Report Writer, Oracle Graphics etc.

* SQL \* PLUS: The SQL \* PLUS tool is made up of two distinct parts. These are
  + Interactive SQL: Interactive SQL is designed for create, access and manipulate data structures like tables and indexes.
  + PL/SQL: PL/SQL can be used to developed programs for different applications.
* Oracle Forms: This tool allows you to create a data entry screen along with the suitable menu objects. Thus it is the oracle forms tool that handles data gathering and data validation in a commercial application.
* Report Writer: Report writer allows programmers to prepare innovative reports using data from the oracle structures like tables, views etc. It is the report writer tool that handles the reporting section of commercial application.
* Oracle Graphics: Some of the data can be better represented in the form of pictures. The oracle graphics tool allows programmers to prepare graphs using data from oracle structures like tables, views etc.

SQL (Structured Query Language):

Structured Query Language is a [database](http://en.wikipedia.org/wiki/Database) [computer language](http://en.wikipedia.org/wiki/Programming_language) designed for managing [data](http://en.wikipedia.org/wiki/Data) in [relational database management systems](http://en.wikipedia.org/wiki/Relational_database_management_system)(RDBMS), and originally based upon [Relational Algebra](http://en.wikipedia.org/wiki/Relational_Algebra). Its scope includes data query and update, [schema](http://en.wikipedia.org/wiki/Database_schema) creation and modification, and data access control. SQL was one of the first languages for [Edgar F. Codd](http://en.wikipedia.org/wiki/Edgar_F._Codd)'s [relational model](http://en.wikipedia.org/wiki/Relational_model) in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks"[[3]](http://en.wikipedia.org/wiki/SQL#cite_note-codd-relational-model-2#cite_note-codd-relational-model-2) and became the most widely used language for relational databases.

* IBM developed SQL in mid of 1970’s.
* Oracle incorporated in the year 1979.
* SQL used by IBM/DB2 and DS Database Systems.
* SQL adopted as standard language for RDBS by ASNI in 1989.

**DATA TYPES:**

CHAR (Size): This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can hold. The maximum number of character is 255 characters.

* 1. VARCHAR (Size) / VERCHAR2 (Size): This data type is used to store variable length alphanumeric data. The maximum character can hold is 2000 character.
  2. NUMBER (P, S): The NUMBER data type is used to store number (fixed or floating point). Number of virtually any magnitude may be stored up to 38 digits of precision. Number as large as 9.99 \* 10 124. The precision (p) determines the number of places to the right of the decimal. If scale is omitted then the default is zero. If precision is omitted, values are stored with their original precision up to the maximum of 38 digits.
  3. DATE: This data type is used to represent date and time. The standard format is dd-mm-yy as in 17-SEP-2009. To enter dates other than the standard format, use the appropriate functions. Date time stores date in the 24-Hours format. By default the time in a date field is 12:00:00 am, if no time portion is specified. The default date for a date field is the first day the current month.
  4. LONG: This data type is used to store variable length character strings containing up to 2GB. Long data 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.
  5. RAW: The RAW data type is used to store binary data, such as digitized picture or image. Data loaded into columns of these data types are stored without any further conversion. RAW data type can have a maximum length of 255 bytes. LONG RAW data type can contain up to 2GB.

**interactive SQL:**

syntax : verb(Parameter\_1,Parameter\_2,Parameter\_3,........Parameter\_n);

SQL language is sub-divided into several language elements, including:

* + Clauses, which are in some cases optional, constituent components of statements and queries.
  + Expressions, which can produce either [scalar](http://en.wikipedia.org/wiki/Scalar_(computing)) values or [tables](http://en.wikipedia.org/wiki/Table_(database)) consisting of [columns](http://en.wikipedia.org/wiki/Column_(database)) and [rows](http://en.wikipedia.org/wiki/Row_(database)) of data.
  + Predicates which specify conditions that can be evaluated to SQL [three-valued logic (3VL)](http://en.wikipedia.org/wiki/Ternary_logic) Boolean truth values and which are used to limit the effects of statements and queries, or to change program flow.
  + Queries which retrieve data based on specific criteria.
  + Statements which may have a persistent effect on schemas and data, or which may control transactions, program flow, connections, sessions, or diagnostics.
  + SQL statements also include the [semicolon](http://en.wikipedia.org/wiki/Semicolon) (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.
  + [Insignificant white space](http://en.wikipedia.org/wiki/Whitespace_(computer_science)) is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

There are five types of SQL statements. They are:

1. data definition LANGUAGE (ddl)

2. data manipulation language (dml)

3. DATA RETRIEVAL LANGUAGE (DRL)

4. TRANSACTIONAL CONTROL LANGUAGE (TCL)

5. DATA CONTROL LANGUAGE (DCL)

**Experiment NUmber- 2.**

data definition LANGUAGE (ddl) Commands

data definition LANGUAGE (ddl):The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project. Let's take a look at the structure and usage of four basic DDL commands:

1. CREATE 2. ALTER 3. DROP 4. RENAME

**1. CREATE:**

**(a)create table:** This is used to create a new relation and the corresponding

**Syntax:** create table relation\_name

(field\_1 data\_type(Size), field\_2 data\_type(Size) , .. …………….. );

**Example:**

SQL>create table Student (sno NUMBER(3), sname char(10), class char(5));

**(b)create TABLE..as select....:** This is used to create the structure of a new relation from the structure of an existing relation.

**Syntax:** create table (relation\_name\_1, field\_1,field\_2,.....field\_n) AS SELECT field\_1,field\_2,...........field\_n from relation\_name\_2;

**Example:** SQL>create table std(rno, sname) as select sno, sname from student;

**2. ALTER:**

**(a)ALTER TABLE ...ADD...:** This is used to add some extra fields into existing relation.

**Syntax:** ALTER TABLE relation\_name ADD(new field\_1 data\_type(size), new field\_2 data\_type(size),…………);

**Example :**  SQL>ALTER TABLE std ADD(Address CHAR(10));

**(b)ALTER table...modify...:** This is used to change the width as well as data type of fields of existing relations.

**Syntax:** alter table relation\_name modify (field\_1 newdata\_type(Size), field\_2 newdata\_type(Size),..............field\_newdata\_type(Size));

**Example:**SQL>alter table student modify(sname varchar(10),class varchar(5));

**3. drop table:** This is used to delete the structure of a relation. It permanently deletes the records in the table.

**Syntax:**  drop table relation\_name;

**Example:** SQL>drop table std;

**4. Rename:** It is used to modify the name of the existing database object.

**Syntax:**  RENAME table old\_relation\_name TO new\_relation\_name;

**Example:** SQL>rename table std to std1;

**5. TRUNCATE:** This command will remove the data permanently. But structure will not be removed.

**Syntax:** TRUNCATE TABLE <Table name>

**Example** TRUNCATE TABLE student;

**Difference between Truncate & Delete:**-

* By using truncate command data will be removed permanently & will not get back where as by using delete command data will be removed temporally & get back by using roll back command.
* By using delete command data will be removed based on the condition where as by using truncate command there is no condition.
* Truncate is a DDL command & delete is a DML command.

**Experiment NUmber- 3.**

data manipulation language (dml) Commands

data manipulation language (dml): The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

**1. INSERT 2. UPDATE 3. DELETE**

**1. insert into:**  This is used to add records into a relation. These are three type of insert into queries which are as

**a) Inserting a single record**

**Syntax:** insert into relationname(field\_1,field\_2,………field\_n)

values (data\_1,data\_2,........data\_n);

**Example:** SQL>insert into student (sno,sname,class,address)VALUES

(1,’Ravi’,’M.Tech’,’Palakol’);

**b) Inserting all records from another relation**

**Syntax:** insert into relation\_name\_1 select field\_1,field\_2,field\_n

FROM relation\_name\_2 WHERE field\_x=data;

**Example:** SQL>insert into std select sno,sname from student

where name = ‘Ramu‘;

**c) Inserting multiple records**

**Syntax:** insert into relation\_name field\_1,field\_2,.....field\_n) values

(&data\_1,&data\_2,........&data\_n);

**Example:** SQL>insert into student(sno,sname,class,address)

VALUES(&sno,’&sname’,’&class’,’&address’);

Enter value for sno: 101

Enter value for name: Ravi

Enter value for class: M.Tech

Enter value for name: Palakol

**2. update-set-WHERE:** This is used to update the content of a record in a relation.

**Syntax:**  SQL>update relation name

set field\_name1=data,field\_name2=data,……..

where field\_name=data;

**Example:** SQL>update student set sname = ‘kumar’ WHERE sno=1;

**3. delete-from**: This is used to delete all the records of a relation but it will retain the structure of that relation.

**a) delete-from**: This is used to delete all the records of relation.

**Syntax:** SQL>delete from relation\_name;

**example:** SQL>delete from std;

**b) delete -from-WHERE:** This is used to delete a selected record from a relation.

**Syntax:** SQL>delete from relation\_name WHERE condition;

**Example:** SQL>delete from student WHERE sno = 2;

**Experiment NUmber- 4.**

DRL (DATA RETRIEVAL LANGUAGE) Commands

DRL(DATA RETRIEVAL LANGUAGE): Retrieves data from one or more tables.

**1. select from:** To display all fields for all records.

**Syntax :** select \* from relation\_name;

**Example :** SQL> select \* from dept;

DEPTNO DNAME

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

10 ACCOUNTING

20 RESEARCH

30 SALES

40 OPERATIONS

**2. Select from:** To display a set of fields for all records of relation.

**Syntax:** select a set of fields FROM relation\_name;

**Example:** SQL> select deptno, dname from dept;

DEPTNO DNAME

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

10 ACCOUNTING

20 RESEARCH

30 SALES

**3. select - from -WHERE:** This query is used to display a selected set of fields for a selected set of records of a relation.

**Syntax:** select a set of fields from relation\_name where condition;

**Example:** SQL> select \* FROM dept WHERE deptno<=20;

DEPTNO DNAME

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

10 ACCOUNTING

20 RESEARCH

**4. select - from -group BY:** This query is used to group to all the records in a relation together for each and every value of a specific key(s) and then display them for a selected set of fields the relation.

**Syntax:** select a set of fields FROM relation\_name GROUP BY field\_name;

**Example:** SQL> SELECT EMPNO, SUM (SALARY) FROM EMP GROUP BY EMPNO;

EMPNO SUM (SALARY)

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

1 3000

2 4000

3 5000

4 6000

4 rows selected.

**5. select - from -order by:** This query is used to display a selected set of fields from a relation in an ordered manner base on some field (ascending and descending order).

**Syntax:** select a set of fields FROM relation\_name

order by field\_name;

**Example:** SQL> SELECT empno,ename,job FROM emp ORDER BY salary;

EMPNO ENAME SALARY

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

4 RAVI 3000

2 aravind 4000

1 sagar 5000

3 Laki 6000

4rows selected.

**Example:** SQL> SELECT empno,ename,job FROM emp ORDER BY salary desc ;

EMPNO ENAME SALARY

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

4 Laki 6000

2 sagar 5000

1 aravind 4000

3 RAVI 3000

**6**. **HAVING clause**: HAVING clause is used to specify which groups are to be displayed, and thus, you further restrict the groups on the basis of aggregate information. In the syntax:

*group\_condition* restricts the groups of rows returned to those groups for whichthe specified condition is true

**Example:** SQL> SELECT department\_id, MAX(salary) FROM employees

GROUP BY department\_id **HAVING** **MAX(salary)>10000;**

Department\_id MAX(SALARY)

20 5000

30 8000

50 7500

60 5900

**7. join using select - from - order by:** This query is used to display a set of fields from two relations by matching a common field in them in an ordered manner based on some fields.

**Syntax:** select a set of fields from both relations from relation\_1, relation\_2 WHERE relation\_1.field\_x = relation\_2.field\_y order by field\_z;

**Example:** SQL>SELECT empno,ename,job,dname FROM emp,dept

WHERE emp.deptno = 20 ORDER BY job;

EMPNO ENAME JOB DNAME

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

7788 SCOTT ANALYST ACCOUNTING

7902 FORD ANALYST ACCOUNTING

7566 JONES MANAGER OPERATIONS

7566 JONES MANAGER SALES

20 rows selected.

**8. join using select - from - group by:** This query is used to display a set of fields from two relations by matching a common field in them and also group the corresponding records for each and every value of a specified key(s) while displaying.

**Syntax:** select a set of fields from both relations FROM relation\_1,relation\_2 WHERE relation\_1.field-x=relation\_2.field-y group by field-z;

**Example:** SQL> SELECT empno,SUM(SALARY) FROM emp,dept

WHERE emp.deptno =20 GROUP BY JOB;

EMPNO SUM (SALARY)

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

7369 3200

7566 11900

7788 12000

7876 4400

**9. union:** This query is used to display the combined rows of two different queries, which are having the same structure, without duplicate rows.

**Syntax:** SELECT field\_1,field\_2,....... FROM relation\_1 WHERE (Condition) UNION SELECT field\_1,field\_2,....... FROM relation\_2 WHERE (Condition);

**Example:**

SQL> SELECT \* FROM STUDENT; SQL> SELECT \* FROM STD;

SNO SNAME SNO SNAME

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

1 kumar 3 ramu

2 ravi 5 lalitha

3 ramu 9 devi

SQL> SELECT \* FROM student UNION SELECT \* FROM std;

SNO SNAME

---- ------

1 kumar

2 ravi

3 ramu

5 lalitha

9 devi

**10. interseCt:** This query is used to display the common rows of two different queries, which are having the same structure, and to display a selected set of fields out of them.

**Syntax:** select field\_1,field\_2,.. FROM relation\_1 WHERE

(Condition) INTERSECT SELECT field\_1,field\_2,.. FROM relation\_2 WHERE(Condition);

**Example :** SQL> SELECT \* FROM student INTERSECT SELECT \* FROM std;

SNO SNAME

---- -------

1. Kumar

3 ramu

**11. minus:** This query is used to display all the rows in relation\_1,which are not having in the relation\_2.

**Syntax:** select field\_1,field\_2,......FROM relation\_1

WHERE(Condition) MINUS SELECT field\_1,field\_2,.....

FROM relation\_2 WHERE(Conditon);

**SQL>** SELECT \* FROM student MINUS SELECT \* FROM std;

SNO SNAME

---- -------

2 RAVI

Note: Different Where Condition

1. Comparison Condition

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| = | Equal to |
| > | Greater than |
| >= | Greater than or Equal to |
| < | Less than |
| <= | Less than or Equal to |
| <> | Not Equal to |
| Between ……AND…….. | Between two values |
| IN (set) | Match any of a list of values |
| LIKE | Match a Character Pattern |
| IS NULL | Is a null Value |

1. Logical Conditions

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| AND | Return TRUE if both component condition are true |
| OR | Return TRUE if either component condition is true |
| NOT | Return TRUE if the following condition is false |

**Experiment NUmber- 5.**

TRANSATIONAL CONTROL LANGUAGE (T.C.L) Commands

TRANSACTIONAL CONTROL LANGUAGE (T.C.L): A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be mode permanent to the database only if they are committed a transaction begins with an executable SQL statement & ends explicitly with either role back or commit statement.

**1. COMMIT:** This command is used to end a transaction only with the help of the commit command transaction changes can be made permanent to the database.

**Syntax:** SQL>COMMIT;

**Example:** SQL>COMMIT;

**2. SAVE POINT**: Save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter role back. Thus, save point is used in conjunction with role back.

**Syntax:** SQL>SAVE POINT ID;

**Example:** SQL>SAVE POINT xyz;

**3. ROLE BACK:** A role back command is used to undo the current transactions. We can role back the entire transaction so that all changes made by SQL statements are undo (or) role back a transaction to a save point so that the SQL statements after the save point are role back.

**Syntax:**  ROLE BACK( current transaction can be role back)

ROLE BACK to save point ID;

**Example:** SQL>ROLE BACK;

SQL>ROLE BACK TO SAVE POINT xyz;

**Experiment NUmber- 6.**

DATA CONTROL LANGUAGE (D.C.L) Commands

DATA CONTROL LANGUAGE (D.C.L): DCL provides users with privilege commands the owner of database objects (tables), has the soul authority ollas them. The owner (data base administrators) can allow other data base users to access the objects as per their requirement

**1. GRANT:** The GRANT command allows granting various privileges to other users and allowing them to perform operations within their privileges

**For Example**, if a uses is granted as ‘SELECT’ privilege then he/she can only view data but cannot perform any other DML operations on the data base object GRANTED privileges can also be withdrawn by the DBA at any time

**Syntax:** SQL>GRANT PRIVILEGES on object\_name To user\_name;

**Example**: SQL>GRANT SELECT, UPDATE on emp To hemanth;

**2. REVOKE:** To with draw the privileges that has been GRANTED to a uses, we use the REVOKE command

**Syntax:** SQL>REVOKE PRIVILEGES ON object-name FROM user\_name;

**Example:** SQL>REVOKE SELECT, UPDATE ON emp FROM ravi;

**Experiment NUmber- 7.**

INTEGRITY CONSTRAINTS

Integrity Constraints are used to apply business rules for the database tables.

The constraints available in SQL are Primary Key, **Foreign Key, Not Null, Unique, Check.**

Constraints can be defined in two ways   
1) The constraints can be specified immediately after the column definition. This is called column-level definition.   
2) The constraints can be specified after all the columns are defined. This is called table-level definition.

## I) SQL Primary key:

This constraint defines a column or combination of columns which uniquely identifies each row in the table.

**Syntax to define a Primary key at column level:**

column name datatype [CONSTRAINT constraint\_name] PRIMARY KEY

**Syntax to define a Primary key at table level:**

[CONSTRAINT constraint\_name] PRIMARY KEY (column\_name1,column\_name2,..)

* **column\_name1, column\_name2** are the names of the columns which define the primary Key.
* The syntax within the bracket i.e. [CONSTRAINT constraint\_name] is optional.

**For Example:** To create an employee table with Primary Key constraint, the query would be like.

**Primary Key at column level:**

CREATE TABLE employee   
( id number(5) PRIMARY KEY,   
name char(20),   
dept char(10),   
age number(2),   
salary number(10),   
location char(10)   
);

or

CREATE TABLE employee  
( id number(5) CONSTRAINT emp\_id\_pk PRIMARY KEY,   
name char(20),  
dept char(10),  
age number(2),  
salary number(10),  
location char(10)  
);

**Primary Key at column level:**

CREATE TABLE employee   
( id number(5),   
name char(20),  
dept char(10),  
age number(2),  
salary number(10),  
location char(10),  
CONSTRAINT emp\_id\_pk PRIMARY KEY (id)  
);

**Primary Key at table level:**

CREATE TABLE employee   
( id number(5), NOT NULL,  
name char(20),  
dept char(10),  
age number(2),  
salary number(10),  
location char(10),  
ALTER TABLE employee ADD CONSTRAINT PK\_EMPLOYEE\_ID PRIMARY KEY (id)  
);

## II) SQL Foreign key or Referential Integrity :

This constraint identifies any column referencing the PRIMARY KEY in another table. It establishes a relationship between two columns in the same table or between different tables. For a column to be defined as a Foreign Key, it should be a defined as a Primary Key in the table which it is referring. One or more columns can be defined as Foreign key.

**Syntax to define a Foreign key at column level:**

[CONSTRAINT constraint\_name] REFERENCES Referenced\_Table\_name(column\_name)

**Syntax to define a Foreign key at table level:**

[CONSTRAINT constraint\_name] FOREIGN KEY(column\_name) REFERENCES referenced\_table\_name(column\_name);

**For Example:**

a) Lets use the "product" table and "order\_items".   
  
**Foreign Key at column level:**

CREATE TABLE product   
( product\_id number(5) CONSTRAINT pd\_id\_pk PRIMARY KEY,   
product\_name char(20),  
supplier\_name char(20),  
unit\_price number(10)  
);

CREATE TABLE order\_items  
( order\_id number(5) CONSTRAINT od\_id\_pk PRIMARY KEY,  
product\_id number(5) CONSTRAINT pd\_id\_fk REFERENCES, product(product\_id),  
product\_name char(20),  
supplier\_name char(20),  
unit\_price number(10)  
);

**Foreign Key at table level:**

CREATE TABLE order\_items  
( order\_id number(5) ,  
product\_id number(5),  
product\_name char(20),  
supplier\_name char(20),  
unit\_price number(10)  
CONSTRAINT od\_id\_pk PRIMARY KEY(order\_id),  
CONSTRAINT pd\_id\_fk FOREIGN KEY(product\_id) REFERENCES product(product\_id)  
);

b) If the employee table has a 'mgr\_id' i.e, manager id as a foreign key which references primary key 'id' within the same table, the query would be like,

CREATE TABLE employee  
( id number(5) PRIMARY KEY,  
name char(20),  
dept char(10),  
age number(2),  
mgr\_id number(5) REFERENCES employee(id),  
salary number(10),  
location char(10)   
);

## III) SQL Not Null Constraint :

This constraint ensures all rows in the table contain a definite value for the column which is specified as not null. Which means a null value is not allowed.

**Syntax to define a Not Null constraint:**

[CONSTRAINT constraint name] NOT NULL

**For Example:** To create a employee table with Null value, the query would be like

CREATE TABLE employee  
( id number(5),  
name char(20) CONSTRAINT nm\_nn NOT NULL,  
dept char(10),  
age number(2),  
salary number(10),  
location char(10)   
);

## IV) SQL Unique Key:

This constraint ensures that a column or a group of columns in each row have a distinct value. A column(s) can have a null value but the values cannot be duplicated.

**Syntax to define a Unique key at column level:**

[CONSTRAINT constraint\_name] UNIQUE

**Syntax to define a Unique key at table level:**

[CONSTRAINT constraint\_name] UNIQUE(column\_name)

**For Example:** To create an employee table with Unique key, the query would be like,

**Unique Key at column level:**

CREATE TABLE employee  
( id number(5) PRIMARY KEY,  
name char(20),  
dept char(10),  
age number(2),  
salary number(10),  
location char(10) UNIQUE   
);

or

CREATE TABLE employee  
( id number(5) PRIMARY KEY,  
name char(20),  
dept char(10),  
age number(2),  
salary number(10),  
location char(10) CONSTRAINT loc\_un UNIQUE   
);

**Unique Key at table level:**

CREATE TABLE employee  
( id number(5) PRIMARY KEY,  
name char(20),  
dept char(10),  
age number(2),  
salary number(10),  
location char(10),  
CONSTRAINT loc\_un UNIQUE(location)   
);

## V) SQL Check Constraint :

This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.

**Syntax to define a Check constraint:**

[CONSTRAINT constraint\_name] CHECK (condition)

**For Example:** In the employee table to select the gender of a person, the query would be like

**Check Constraint at column level:**

CREATE TABLE employee   
( id number(5) PRIMARY KEY,   
name char(20),   
dept char(10),   
age number(2),   
gender char(1) CHECK (gender in ('M','F')),   
salary number(10),   
location char(10)   
);

**Check Constraint at table level:**

CREATE TABLE employee   
( id number(5) PRIMARY KEY,   
name char(20),   
dept char(10),   
age number(2),   
gender char(1),   
salary number(10),   
location char(10),   
CONSTRAINT gender\_ck CHECK (gender in ('M','F'))   
);

**Experiment NUmber- 8.**

JOINS

**JOINS:**

Join is a query in which data is returned from two or more tables.

How the join will be performed:

Step 1: Make the Cartesian product of the given tables.

Step 2: Check for the equality on common attributes for the given tables.

**1) Natural join:**

It returns the matching rows from the table that are being joined.

Syntax:

select <attribute> from TN1,TN2 where TN1.attribute=TN2.attribute.

**2) Inner join:**

It returns the matching rows from the table that are being joined.

Syntax:

select <attribute> from TN1 innerjoin TN2 on TN1.attribute=TN2.attribute.

**3) Left outer join:**

It returns all the rows from the table1 even when they are unmatched.

Syntax:

select <attribute> from TN1 left outer join TN2 on TN1.attribute=TN2.attribute.

OR

select <attribute> from TN where TN1.attribute(+)=TN2.attribute.

**4) Right outer join:**

It returns all the rows from the table2 even when they are unmatched.

Syntax:

select <attribute> from TN1 right outer join TN2 on TN1.attribute=TN2.attribute.

OR

select <attribute> from TN where TN1.attribute=(+)TN2.attribute.

**5) Full join:**

It is the combination of both left outer and right outer join.

Syntax:

select <attribute> from TN1 full join TN2 on TN1.attribute=TN2.attribute.

**Experiment NUmber- 9**

Subqueries

**Subqueries**

A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. You can build powerful statements out of simple ones by using subqueries. They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

You can place the subquery in a number of SQL clauses, including:

* The WHERE clause
* The HAVING clause
* The FROM clause

In the syntax:

*operator* includes a comparison condition such as >, =, orIN

**Note:** Comparison conditions fall into two classes: single-row operators (>, =, >=, <, <>, <=) and multiple-row operators (IN, ANY, ALL).

The subquery is often referred to as a nested SELECT, sub-SELECT, or inner SELECT statement. The subquery generally executes first, and its output is used to complete the query condition for the main or outer query.

Subquery Syntax

Select Select\_list from table where expr operator (select Select\_list from table);

Example

SQL> SELECT last\_name, job\_id FROM employees

WHERE job\_id = (SELECT job\_id FROM employees WHERE employee\_id = 141);

Last\_Name Job\_id

Shayam Manager

Mohan Manager

Rahul Manager

Prince Manager

VIEWS

A view is a virtual table based on the result-set of an SQL statement.

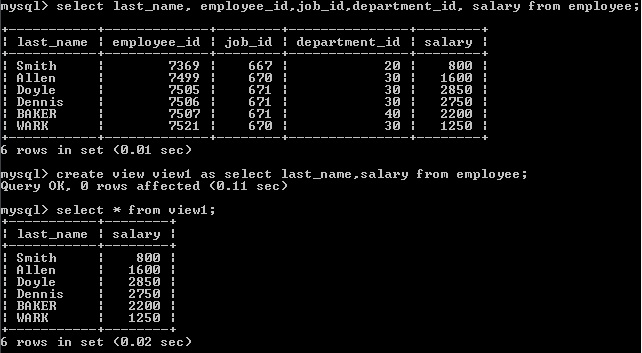
A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

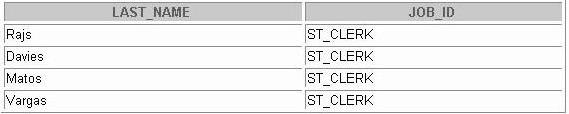
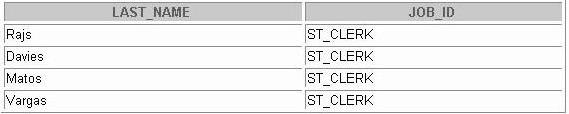
You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

VIEW Syntax

CREATE VIEW view\_name AS  
SELECT column\_name(s)  
FROM table\_name

Example





**Experiment NUmber- 10**

PL/SQLINTRODUCTION

* **PL/SQL** stands for **Procedural Language** extension of SQL.
* PL/SQL is a combination of SQL along with the procedural features of programming languages.
* It was developed by Oracle Corporation in the early 90’s to enhance the capabilities of SQL.
* Oracle uses a PL/SQL engine to processes the PL/SQL statements. A PL/SQL code can be stored

in the client system (client-side) or in the database (server-side).

## A Simple PL/SQL Block:

Each PL/SQL program consists of SQL and PL/SQL statements which from a PL/SQL block.

### PL/SQL Block consists of three sections:

* The Declaration section (optional).
* The Execution section (mandatory).
* The Exception (or Error) Handling section (optional).

### Declaration Section:

The Declaration section of a PL/SQL Block starts with the reserved keyword DECLARE. This section is optional and is used to declare any placeholders like variables, constants, records and cursors, which are used to manipulate data in the execution section. Placeholders may be any of Variables, Constants and Records, which stores data temporarily. Cursors are also declared in this section.

### Execution Section:

The Execution section of a PL/SQL Block starts with the reserved keyword BEGIN and ends with END. This is a mandatory section and is the section where the program logic is written to perform any task. The programmatic constructs like loops, conditional statement and SQL statements form the part of execution section.

### Exception Section:

The Exception section of a PL/SQL Block starts with the reserved keyword EXCEPTION. This section is optional. Any errors in the program can be handled in this section, so that the PL/SQL Blocks terminates gracefully. If the PL/SQL Block contains exceptions that cannot be handled, the Block terminates abruptly with errors.   
  
Every statement in the above three sections must end with a semicolon **;** . PL/SQL blocks can be nested within other PL/SQL blocks. Comments can be used to document code.

### How a Sample PL/SQL Block Looks

|  |
| --- |
| DECLARE       Variable declaration BEGIN       Program Execution  EXCEPTION       Exception handling END; |

## PL/SQL Program

## pl 1.jpg

**Experiment NUmber- 11**

PROCEDURE

A procedure is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages.

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

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

Procedures: Passing Parameters

We can pass parameters to procedures in three ways.  
1) IN-parameters  
2) OUT-parameters  
3) IN OUT-parameters

A procedure may or may not return any value.

General Syntax to create a procedure :

CREATE [OR REPLACE] PROCEDURE proc\_name [list of parameters]

IS

Declaration section

BEGIN

Execution section

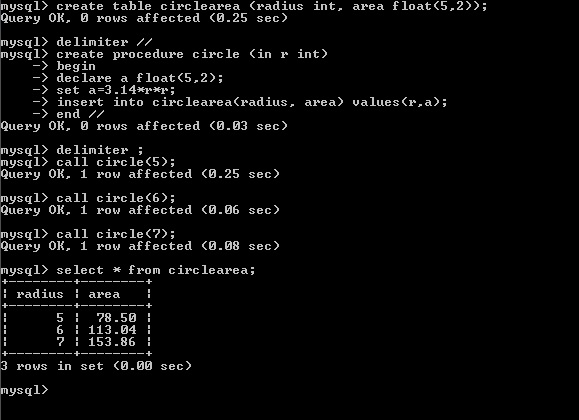
EXCEPTION

Exception section

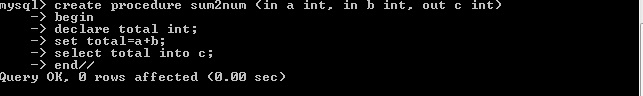
END;

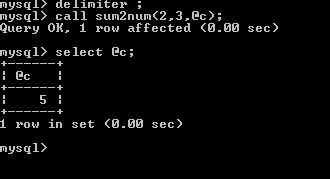
Procedure Example

PL/SQL TO FIND AREA OF CIRCLE



PL/SQL TO FIND ADDITION OF TWO NUMBERS





**Experiment NUmber- 12**

TRIGGER

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events:

* A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
* A database definition (DDL) statement (CREATE, ALTER, or DROP).
* A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

The syntax for creating a trigger is:

CREATE

[DEFINER = { ***user*** | CURRENT\_USER }]

TRIGGER ***trigger\_name***

***trigger\_time*** ***trigger\_event***

ON ***tbl\_name*** FOR EACH ROW

***trigger\_body***

***trigger\_time***: { BEFORE | AFTER }

***trigger\_event***: { INSERT | UPDATE | DELETE }

* CREATE [OR REPLACE] TRIGGER trigger\_name: Creates or replaces an existing trigger with the *trigger\_name*.
* {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
* {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
* [OF col\_name]: This specifies the column name that would be updated.
* [ON table\_name]: This specifies the name of the table associated with the trigger.
* [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
* [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
* WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

