**Connect to database**

**To show the database**

**SYNTAX:**

SHOW DATABASES;

**To use the database**

**SYNTAX:**

USE <database\_name>;

**EXAMPLE:**

**SQL>** USE emp\_details;

**To show the tables**

**SYNTAX:**

SHOW TABLES;

**DDL Commands**

1. **CREATE**

Create command is used to create a table in the database. It defines each column of the table uniquely. Each column has minimum of three attributes, a name, data type and size.

**To crate the database**

**SYNTAX:**

CREATE DATABASE <database\_name>;

**EXAMPLE:**

**SQL>**CREATE DATABASE emp\_details;

**To crate the table**

**SYNTAX:**

CREATE TABLE <table\_name> (<col1> <datatype>(<size>),<col2> <datatype><size>));

**EXAMPLE:**

**SQL>**CREATE TABLE Emp(EmpNo int, EmpId char(10));

1. **DESC**

To describe the structure (column and data types) of an existing database, table, index, or view.

**SYNTAX:**

DESC table\_name;

**EXAMPLE:**

**SQL>** DESC Emp;

1. **RENAME**

To change the name of the table

**SYNTAX:**

RENAME TABLE <Old\_Table\_Name> TO <New\_Table\_Name>

**EXAMPLE:**

**SQL>** RENAME TABLE Emp to employee;

1. **DROP**

If a table is dropped, all records held within it are lost and cannot be recovered

**SYNTAX:**

DROP TABLE <table\_name>;

**EXAMPLE:**

**SQL>** DROP TABLE emp;

1. **ALTER**

To modify an existing database object. The structure of the table can be changed using this command like add new column, change the width of a datatype, change the datatype of a column.

**Adding New Columns**

**SYNTAX:**

ALTER TABLE <table\_name> ADD(<new col><datatype(size),<new col>datatype(size));

**EXAMPLE:**

**SQL>** ALTER TABLE emp ADD sal int;

**Dropping a columns from a table**

**SYNTAX:**

ALTER TABLE <table\_name> DROP <column\_Name>:

**EXAMPLE:**

**SQL>** ALTER TABLE emp DROP sal;

**Modifying Existing columns**

**SYNTAX:**

ALTER TABLE <table\_name> MODIFY columnName1 Newdatatype(Newsize);

**EXAMPLE:**

**SQL>** ALTER TABLE emp MODIFY empno char(10);

1. **TRUNCATE**

To destroy the data in an existing database, table, index, or view. If a table is truncated all records held within it are lost and cannot be recovered but the table structure is maintained.

**SYNTAX:**

TRUNCATE TABLE <table\_name>;

**EXAMPLE:**

**SQL>** TRUNCATE TABLE emp;

**DML Commands**

1. **INSERT**

INSERT statement adds one or more records to any single table in a relational database. It is used to insert data into a table.

**SYNTAX:**

INSERT INTO <table\_name> VALUES (expr1,expr2……..);

**EXAMPLE:**

**SQL>** INSERT INTO emp VALUES(100,’Smith’);

1. **UPDATE**

UPDATE statement that changes the data of one or more records in a table. It is used to update a row of a table. Either all the rows can be updated, or a subset may be chosen using a condition.

**SYNTAX:**

UPDATE <table\_name> SET <column\_name> = value [, column\_name = value ...] [WHERE

condition]

**EXAMPLE:**

**SQL>** UPDATE emp SET empid=’John’ WHERE empno=100;

1. **DELETE**

DELETE statement removes one or more records from a table. A subset may be defined for deletion using a condition, otherwise all records are removed.

**SYNTAX:**

DELETE FROM <table\_name> [WHERE condition]

**EXAMPLE:**

**SQL>** DELETE FROM emp;

1. **SELECT**

SELECT statement is used to query or retrieve data from a table in the database.

**To get all the elements**

**SYNTAX:**

SELECT \* FROM <table\_name>;

**EXAMPLE:**

**SQL>** SELECT \* FROM emp;

**To get a particular column**

**SYNTAX:**

SELECT <column\_name> FROM <table\_name>;

**EXAMPLE:**

**SQL>** SELECT empid FROM emp;

**Aggregate Functions**

Aggregate functions perform a calculation on a set of values and return a single value.

Consider the schema

EMPLOYEE (name, salary)

1. **COUNT(\*)**

Returns the number of rows

**EXAMPLE:**

**SQL>** SELECT COUNT(\*) FROM employee;

1. **AVG**

Returns the average value of a given column

**EXAMPLE:**

**SQL>** SELECT AVG(salary) FROM employee;

1. **MIN**

Returns the smallest value

**EXAMPLE:**

**SQL>** SELECT MIN(salary) FROM employee;

1. **MAX**

Returns the largest value

**EXAMPLE:**

**SQL>** SELECT MAX(salary) FROM employee;

1. **SUM**

Returns the sum of the values

**EXAMPLE:**

**SQL>** SELECT SUM(salary) FROM employee;

**Constraints**

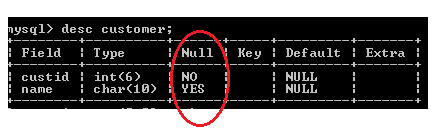
1. **NOT NULL**

While creating tables, by default the rows can have null value. The enforcement of not null constraint in a table ensures that the table contains values.

**EXAMPLE:**

**SQL>** CREATE TABLE CUSTOMER(custid int(6) NOT NULL, name char(10));

**(use desc command to see the structure of the table and see the change)**

****

**SQL>** INSERT INTO CUSTOMER VALUES (100, ‘RUAS’); valid

**SQL>** INSERT INTO CUSTOMER VALUES (100, ‘NULL’); valid

**SQL>** INSERT INTO CUSTOMER VALUES (NULL, ‘RUAS’); not valid – column custid cannot be null

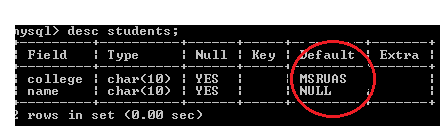
1. **DEFAULT**

The DEFAULT constraint inserts a default value into a column of a table. When you insert a new row into the table, no need to specify the value for the column.

**EXAMPLE:**

**SQL>** CREATE TABLE STUDENTS (college char(10) DEFAULT “MSRUAS”, name char(10));

**(Use desc command to see the structure of the table and see the change)**

****

**SQL>** INSERT INTO STUDENTS VALUES (‘RUAS’, ’John’); the row contains (RUAS, John)

**SQL>** INSERT INTO STUDENTS (name) values (‘Smith’); the row contains (MSRUAS, Smith)

1. **UNIQUE**

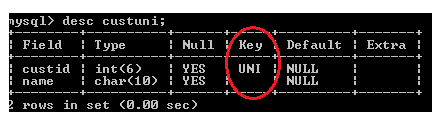
It is used to ensure that information in the column for each record is unique, as with telephone or drivers license numbers. It prevents the duplication of value with rows of a specified column in a set of column. A column defined with the constraint can allow null value.

**Unique constraint at the column level**

**EXAMPLE:**

**SQL>** CREATE TABLE CUSTUNI (custid int(6) UNIQUE, name char(10));

**(Use desc command to see the structure of the table and see the change)**



**SQL>** INSERT INTO CUSTUNI VALUES (100, RUAS); valid

**SQL>** INSERT INTO CUSTUNI VALUES (100, MSRUAS); not valid – duplicate entry for custid (100)

**Unique constraint at the table level**

**EXAMPLE:**

**SQL>** CREATE TABLE CUSTUNIT (Id int(6), Name char(10),unique(Id,Name));

**SQL>** INSERT INTO CUSTUNIT VALUES (100, RUAS); valid

**SQL>** INSERT INTO CUSTUNIT VALUES (100, MSRUAS); valid

**SQL>** INSERT INTO CUSTUNIT VALUES (101, MSRUAS); valid

**SQL>** INSERT INTO CUSTUNIT VALUES (100, MSRUAS); invalid - duplicate entry for CUSTUNIT (100, MSRUAS)

1. **PRIMARY KEY**

A primary key avoids duplication of rows and does not allow null values. It can be defined on one or more columns in a table and is used to uniquely identify each row in a table. These values should never be null. Better to use only one primary key at the table level.

**EXAMPLE:**

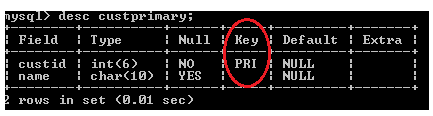
**To add a primary key at the column level**

**SQL>** CREATE TABLE CUSTPRIMARY (custid int(6) PRIMARY KEY, name char(10));

or

**SQL>** CREATE TABLE CUSTPRIMARY (custid int(6), name char(10), PRIMARY KEY (custid))

**(Use desc command to see the structure of the table and see the change)**

****

**SQL>** INSERT INTO CUSTPRIMARY VALUES (100, RUAS); valid

SQL> INSERT INTO CUSTPRIMARY VALUES (100, MSRUAS); not valid – duplicate entry for custid

**To delete a primary key**

**SQL>** ALTER TABLE CUSTPRIMARY DROP PRIMARY KEY;

1. **FOREIGN KEY**

Referential integrity constraint enforces relationship between tables. Foreign key is a column or combination of column included in the definition of referential integrity, which would refer to a referenced key. They create a parent child relationship between two tables. Referenced key is a unique or primary key upon which is defined on a column belonging to the parent table.

**EXAMPLE:**

**SQL>** CREATE TABLE STUDENTS(Name char(10), Reg\_No int, PRIMARY KEY (Reg\_No))

**SQL>** CREATE TABLE EMP (Ename CHAR(15), RegNo INT(11), FOREIGN KEY (RegNo) REFERENCES stud(Reg\_No));

**BUILT-IN FUNCTIONS**

**CHARACTER/STRING FUNCTION:**

SQL> select upper('welcome');

-----------

WELCOME

SQL> select lower('HAI') ;

---

hai

SQL> select concat('MSR',' UAS');

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

MSRUAS

SQL> select length('MSR‟);

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

12

SQL> select replace('MSR UAS', 'MSR,'Ramaiah');

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

Ramaiah university

**DATE & TIME FUNCTION**

SQL> select sysdate ;

---------

07-APR-10

SQL> select add\_months(sysdate,3);

---------

07-JUL-10

**NUMERIC FUNCTION**

SQL> select round(15.6789);

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

16

SQL> select ceil(23.20);

-----------

24

SQL> select floor(34.56);

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

34

SQL> select sign(-345);

----------

-1

**MATH FUNCTION:**

SQL> select abs(-45) ;

---------

45

SQL> select power(2,3) ;

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

8

SQL> select mod(11,5) ;

---------

1

SQL> select sqrt(225) ;

---------

15

**VIEWS**

Q1: The organization wants to display only the details of the employees those who are ASP.

Solution:

1. Create a view on emp table named managers

2. Use select from clause to do horizontal portioning

Ans: SQL> create view empview as select \* from emp where job='ASP';

View created.

SQL> select \* from empview;

EMPNO ENAME JOB DEPTNO SAL

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

2 Arjun ASP 2 12000 3 Gugan ASP 2 20000