DDL COMMANDS

DESCRIPTION:

DDL or Data Definition Language actually consits of the SQL commands that can be used to define the databse schema.It simply deals with description of the database schema and is used to create

and modify the structure of database objects in database.

DDL COMMANDS:

CREATE:

is used to create the database or its objects(like table,index,function,views,store procedure and triggers).

DROP:

is used to delete objects from the databse.

ALTER:

is used to alter the structure of database.

TRUNCATE:

is used to remove all records from a table, including all spaces allocated for the records are removed.

.

RENAME:

is used to rename an object existing in the database.

QUERY

CREATE

CREATE DATABASE STUDENT;

CREATE TABLE STUDENT(R\_NO INT, NAME VARCHAR(10));

INSERT INTO STUDENT(1, ‘ABDUL JAVID’);

INSERT INTO STUDENT(2, ‘ABRAR’);

INSERT INTO STUDENT(34, ‘MICHAEL’);

SELECT \* FROM STUDENT;

|  |
| --- |
| R\_NO NAME  1 ABDULJAVID  2 ABRAR  3 MICHAEL |

ALTER

ALTER TABLE STUDENT ADD (EMAIL VARCHAR(10));

UPDATE STUDENT SET EMAIL=’[123@gmail.c](mailto:123@gmail.com)’ WHERE R\_NO=1;

UPDATE STUDENT SET EMAIL=’[abc@gmail.com](mailto:abc@gmail.com)’ WHERE R\_NO=2;

UPDATE STUDENT SET EMAIL=’[sonybeveira@gmail.com](mailto:sonybeveira@gmail.com)’ WHERE R\_NO=3;

SELECT \* FROM STUDENT;

|  |
| --- |
| R\_NO NAME EMAIL |
| 1 ABDUL JAVID [123@gmail.com](mailto:123@gmail.com)  2 ABRAR [abc@gmail.](mailto:abc@gmail.com)com  34 MICHAEL sonybeveira@gmail.com |

RENAME

ALTER TABLE STUDENT RENAME COLUMN MICHAEL TO MIKE;

SELECT \* FROM STUDENT;

|  |
| --- |
| R\_NO NAME EMAIL |
| 1 ABDUL JAVID [123@gmail.com](mailto:123@gmail.com)  2 ABRAR [abc@gmail.](mailto:abc@gmail.com)com  34 MIKE sonybeveira@gmail.com |

TRUNCATE

TRUNCATE TABLE STUDENT;

|  |
| --- |
|  |
|  |



* **DML(Data Manipulation Language) :** The SQL commands that deals with the manipulation of data present in database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

**Examples of DML:**

* **SELECT** – is used to retrieve data from the a database.
* **INSERT**– is used to insert data into a table.
* **UPDATE** – is used to update existing data within a table.
* **DELETE** – is used to delete records from a database table.

**SELECT:**

**Select is the most commonly used statement in SQL. The SELECT Statement in SQL is used to retrieve or fetch data from a database. We can fetch either the entire table or according to some specified rules. The data returned is stored in a result table. This result table is also called result-set.**

With the SELECT clause of a SELECT command statement, we specify the columns that we want to be displayed in the query result and, optionally, which column headings we prefer to see above the result table.

The select clause is the first clause and is one of the last clauses of the select statement that the database server evaluates. The reason for this is that before we can determine what to include in the final result set, we need to know all of the possible columns that could be included in the final result set.

**INSERT:**

**The INSERT INTO statement of SQL is used to insert a new row in a table. There are two ways of using INSERT INTO statement for inserting rows:**

1. **Only values:** First method is to specify only the value of data to be inserted without the column names.  
   **Syntax:**

**INSERT INTO table\_name VALUES (value1, value2, value3,...);**

**table\_name**: name of the table.

**value1, value2,..** : value of first column, second column,... for the new record

1. **Column names and values both:** In the second method we will specify both the columns which we want to fill and their corresponding values as shown below:  
   Syntax:

**INSERT INTO table\_name (column1, column2, column3,..) VALUES ( value1, value2, value3,..);**

**table\_name**: name of the table.

**column1**: name of first column, second column ...

**value1, value2, value3** : value of first column, second column,... for the new record.

**UPDATE:**

**The UPDATE statement in SQL is used to update the data of an existing table in database. We can update single columns as well as multiple columns using UPDATE statement as per our requirement.**

**Basic Syntax**

**UPDATE table\_name SET column1 = value1, column2 = value2,...**

**WHERE condition;**

**table\_name:** name of the table

**column1**: name of first , second, third column....

**value1**: new value for first, second, third column....

**condition**: condition to select the rows for which the

values of columns needs to be updated.

**DELETE:**

**The DELETE Statement in SQL is used to delete existing records from a table. We can delete a single record or multiple records depending on the condition we specify in the WHERE clause.**

**Basic Syntax:**

DELETE FROM table\_name WHERE some\_condition;

**table\_name**: name of the table

**some\_condition**: condition to choose particular record.

QUESTION:

Performing DML Commands like insertion, deletion, modifying, altering and updating records based on conditions.

QUERIES:

CREATE DATABASE STUDENT;

USE STUDENT;

CREATE TABLE STDNT(RNO INT,RNAME VARCHAR(20),AGE INT);

INSERT INTO STDNT VALUES((3,’ADITHYA’,20),(4,’AISWARYA’,21),(36,’NABEEL’,21),(37,’NAZEEB’,21));

**$$**SELECT \* FROM STDNT;

ALTER TABLE STDNT MODIFY RNO VARCHAR(20);

UPDATE STDNT SET RNO=’5’ WHERE RNAME=’ADITHYA’;

UPDATE STDNT SET RNO=’8’ WHERE RNAME=’AISWARYA’;

UPDATE STDNT SET RNO=’40’ WHERE RNAME=’NABEEL’;

UPDATE STDNT SET RNO=’45’ WHERE RNAME=’NAZEEB’;

**\*\***SELECT \* FROM STDNT;

**##**SELECT RNAME FROM STDNT WHERE RNO=’8’;

RENAME TABLE STDNT TO SDNT;

DELETE FROM SDNT WHERE RNO=’45’;

**^^**SELECT \* FROM SDNT;

**%%**TRUNCATE TABLE SDNT;

OUTPUT:

$$

|  |  |  |
| --- | --- | --- |
| RNO | RNAME | AGE |
| 3 | ADITHYA | 20 |
| 4 | AISWARYA | 21 |
| 36 | NABEEL | 21 |
| 37 | NAZEEB | 21 |

**\*\***

|  |  |  |
| --- | --- | --- |
| RNO | RNAME | AGE |
| 5 | ADITHYA | 20 |
| 8 | AISWARYA | 21 |
| 40 | NABEEL | 21 |
| 45 | NAZEEB | 21 |

**##**

|  |
| --- |
| RNAME |
| AISWARYA |

**^^**

|  |  |  |
| --- | --- | --- |
| **RNO** | **RNAME** | **AGE** |
| **5** | **ADITHYA** | **20** |
| **8** | **AISWARYA** | **21** |
| **40** | **NABEEL** | **21** |

**%%**

**empty set**

***NESTED QUERIES***

In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query.

There are mainly two types of nested queries:

* **Independent Nested Queries:** In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc. are used in writing independent nested queries.
* **Co-related Nested Queries:** In co-related nested queries, the output of inner query depends on the row which is being currently executed in outer query.

**QUESTIONS**

Consider the following schema

supplier (sid integer, sname string, address string)

products ( pid integer, pname string, color string)

catalog( sid integer, pid integer, cost real)

**QUERY**

* CREATE TABLE SUPPLIER(SID INT PRIMARY KEY,SNAME VARCHAR(200),ADDRESS VARCHAR(200));
* INSERT INTO SUPPLIER VALUES(1,’GOKUL’,’MUMBAI’),(2,’DAVID’, ’CHENNAI’),(3,’FADIK’,’PUNE’);
* CREATE TABLE PRODUCTS(PID INT PRIMARY KEY,PNAME VARCHAR(90),COLOR VARCHAR(90));
* INSERT INTO PRODUCTS VALUES(100,’FAN’,’BLACK’),(101,’BULB’, ’WHITE’);
* CREATE TABLE CATALOG(SID INT, PID INT,COST FLOAT,FOREIGN KEY(SID) REFERENCES SUPPLIER(SID),FOREIGN KEY(PID) REFERENCES PRODUCT(PID));
* INSERT INTO CATALOG VALUES(1,100,1000.00),(2,101,10.00),(3,102, 1000.00),(1,100,800.00),(1,104,2099.99);
* SELECT \* FROM SUPPLIER;
* SELECT \* FROM PRODUCTS;
* SELECT \* FROM CATALOG;

OUTPUT:

SUPPLIER

|  |  |  |
| --- | --- | --- |
| SID | SNAME | ADDRESS |
| 1 | GOKUL | MUMBAI |
| 2 | DAVID | CHENNAI |
| 3 | DADIK | PUNE |

PRODUCTS

|  |  |  |
| --- | --- | --- |
| PID | PNAME | COLOR |
| 100 | FAN | BLACK |
| 101 | BULB | WHITE |
| 102 | PEN | GREEN |
| 103 | SHIRT | RED |
| 104 | TABLE | BROWN |

CATLOG:

|  |  |  |
| --- | --- | --- |
| SID | PID | COST |
| 1 | 103 | 100.00 |
| 1 | 100 | 800.00 |
| 1 | 104 | 2099.99 |
| 2 | 101 | 100.00 |
| 2 | 102 | 10.00 |
| 3 | 103 | 1000.00 |

QUESTIONS:

1. Find the name of the supplier who supplies a red product that cost less than 100 rupees ?

QUERY:

SELECT SNAME FROM SUPPLIER AS S,PRODUCT AS P,CATLOG AS C WHERE PCOLOUR=’RED’ AND C.COST <100 AND C.SID=S.SID AND C.PID=P.PID

OUTPUT:

|  |
| --- |
| EMPTY |

b)Find the sid of supplier who supply some red or green products?

QUERY:

SELECT C.SID FROM PRODUCT AS P,CATLOG AS C WHERE (P.COLOR=’RED’ OR P.COLOR=’GREEN’) AND C.PID=P.PID

OUTPUT:

|  |
| --- |
| 1 |
| 2 |
| 3 |

c)Find the name of the supplier who supply a red product that cost less than 100 rupees and a green product that cost less than 100 rupees

QUERY:

SELECT S.SNAME FROM SUPPLIER AS S,PRODUCT AS P,CATLOG AS C

WHERE(P.COLOR=’RED’AND C.COST < 100) AND (P.COLOR=’GREEN’ AND C.COST < 100) AND C.PID=P.PID AND P.SID=S.SID

OUTPUT:

|  |
| --- |
| DAVID |

d)Find the sid of the supplier who supply every red or green product?

QUERY:

SELECT C.SID FROM PRODUCT AS P ,CATLOG AS C WHERE(P.CCOLOR=’RED’ OR P.COLOR=’GREEN’) AND C.PID=P.PID;

OUTPUT:

|  |
| --- |
| 1 |
| 2 |
| 3 |

e)Find the pid of the supplier who sell the cheapest and expensive product?

QUERY:

SELECT SID FROM CATLOG

WHERE (COST=(SELECT MAX(COST) FROM CATLOG)

OR COST=(SELECT MIN(COST) FROM CATLOG));

OUTPUT:

|  |
| --- |
| 1 |
| 2 |

f)Find the pid of the product supplied by at least two different suppliers?

SELECT PID FROM CATLOG GROUP BY PID HAVING

COUNT(\*)>1;

OUTPUT:

|  |
| --- |
| 103 |

g)Find the pid of the most expensive products supplied by supplier name David?

SELECT C.PID FROM CATLOG AS C,SUPPLIER AS S

WHERE C.SNAME =’DAVID’ AND C.SID=S.SID

AND C.COST >= ALL (SELECT C2.COST FROM CATLOG AS C1,SUPPLIER AS S2

WHERE S2.NAME=’DAVID’ AND C2.SID=S2.SID);

|  |
| --- |
| 102 |

OUTPUT:

**GROUPING IN SQL**

**DESCRIPTION:**

**The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions.**

**That is if a particular column has same values in different rows then it will arrange these rows in a group.**

**Important Points:**

**· GROUP BY clause is used with the SELECT statement.**

**· In the query, GROUP BY clause is placed after the WHERE clause.**

**· In the query, GROUP BY clause is placed before ORDER BY clause if used any.**

**Syntax:**

**SELECT column1, function \_ name (column2)**

**FROM table \_ name**

**WHERE condition**

**GROUP BY column1, column2**

**ORDER BY column1, column2;**

**Function \_ name: Name of the function used for example, SUM (), AVG ().**

**Table \_ name: Name of the table.**

**Condition: Condition used.**

**Parameters or Arguments**

**Expression 1, expression 2, ... expression \_ n**

**Expressions that are not encapsulated within an aggregate function and must be included in the GROUP BY clause at the end of the SQL statement.**

**Aggregate \_ function**

**This is an aggregate function such as the SUM, COUNT, MIN, MAX, or AVG functions.**

**Aggregate \_ expression**

**This is the column or expression that the aggregate \_ function will be used on.**

**Tables**

**The tables that you wish to retrieve records from. There must be at least one table listed in the FROM clause.**

**Q1: CONSIDER THE SCHEMA**

**DEPARTMENT (DEPTID INT, NO OF EMPLOYEE INT)**

**EMPLOYEE (EMPID INT, NAME STRING, SALARY FLOAT, DEPTID FLOAT)**

**F0R EACH DEPARTMENT RETRIVE THE DEPARTMENT NO, NO OF EMPLOYEES IN A DEPARTMENT AND THEIR AVERAGE SALARIES**

**QUERIES:**

CREATE DATABASE EMPLOYEE;

USE EMPLOYEE;

CREATE TABLE DEPT VALUES ((1, 2), (2, 2), (3, 3));

CREATE TABLE EMPLOYEE (EMPID INT, NAME VARCHAR (200), SALARY FLOAT, DEPTID INT);

INSERT INTO EMPLOYEE VALUES ((100,’AMSHA’, 1000, 1), (101,’AMAYA’, 2000, 1), (102, ‘AMRUTHA’, 3000, 2), (103,’NADEER’, 3000, 2), (104, ‘SAFWAN’, 1000, 3));

SELECT E.DEPTID, D.NOE, AVG (SALARY) FROM DEPT AS D.EMPLOYEE AS E WHERE D.DEPTID = E.DEPTID GROUP BY E.DEPTID;

SELECT \* FROM EMPLOYEE;

SELECT \*FROM DEPT;

**DEPARTMENT**

|  |  |
| --- | --- |
| DEPTID | NOE |
| 1 | 2 |
| 2 | 2 |
| 3 | 3 |

**EMPLOYEE**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMPID** | **NAME** | **SALARY** | **DEPTID** |
| 100 | **AMSHA** | **1000** | **1** |
| **101** | **AMAYA** | **2000** | **1** |
| **102** | **AMRUTHA** | **3000** | **2** |
| **103** | **NADEER** | **3000** | **2** |
| 104 | SAFWAN | 1000 | 3 |

OUTPUT

|  |  |  |
| --- | --- | --- |
| DEPTID | NOE | AVG(SALARY) |
| 1 | 2 | 1500.00 |
| 2 | 2 | 3000.00 |
| 3 | 3 | 1333.33 |

**Q2: CONSIDER THE SCHEMA**

**PROJECT ( PROJECT NO INT, PROJECT NAME STRING)**

**EMPLOYEE (EMPID INT, NAME STRING, PROJECT NO INT)**

**FOR EACH PROJECT RETRIVE THE PROJECT NO, PROJECT NAME AND THE NO OF EMPLOYEE WHO WORK IN THEIR PROJECT**

**QUERIES:**

**CREATE TABLE PROJECT (PRONO INT, PRONAME VARCHAR (200));**

**INSERT INTO PROJECT VALUES ((1,’ALEXA’),(2,’ROBOTICS’),(3,’ORENDA’));**

**SELECT \* FROM PROJECT;**

**CREATE TABLE EMPLOYEE (EMPID INT, ENAME VARCHAR(200), PRONO INT);**

**INSERT INTO EMPLOYEE VALUES ((1,’AMSHA’, 1),(2,’AMAYA’,2),(3,’AMRUTHA’,3),(4,’NADEER’,3)(5, ‘SAFWAN’,3));**

**SELECT \* FROM EMPLOYEE;**

**SELECT P.PRONO, P.PRONAME, COUNT (E.PRONO) FROM PROJECT AS P, EMPLOYEE AS E WHERE P.PRONO = E.PRONO GROUP BY EMPLOYEE.PRONO;**

**PROJECT**

|  |  |
| --- | --- |
| **PRONO** | **PRONAME** |
| **1** | **ALEXA** |
| **2** | **ROBOTICS** |
| **3** | **ORENDA** |

**EMPLOYEE**

|  |  |  |
| --- | --- | --- |
| **EMPID** | **ENAME** | **PRONO** |
| **1** | **AMSHA** | **1** |
| **2** | **AMAYA** | **1** |
| **3** | **AMRUTHA** | **2** |
| **4** | **NADEER** | **3** |
| **5** | **SAFWAN** | **3** |

**OUTPUT**

|  |  |  |
| --- | --- | --- |
| PRONO | PRONAME | COUNT(PRONO) |
| 1 | ALEXA | 2 |
| 2 | ROBOTICS | 1 |
| 3 | ORENDA | 2 |

***AGGREGATE FUNCTIONS***

***IN DATABASE MANAGEMENT SYSTEM AN AGGREGATE FUNCTION WHERE THE VALUES OF MULTIPLE ROWS ARE GROUPED TOGETHER TO FORM A SINGLE VALUE OF MORE SIGNIFICANT MEANING OR MEASURMENT SUCH AS A SET, A BAG OR A LIST***

*SYNTAX*

*SELECT <FUNCTION NAME> (<PARAMETER>) FROM<TABLENAME>*

*TYPES*

*1.AVG FUNCTION*

*THIS FUNCTION RETURNS THE AVERAGE VALUES OF THE NUMERIC COLUMN THAT IS SUPPLIED AS A PARAMETER.*

*2.COUNT FUNCTION*

*THE COUNT FUNCTION RETURNS THE NUMBER OF ROWS IN THE RESULT. IT DOES NOT COUNT THE NULL VALUES.*

*3.MAX FUNCTION*

*IT IS USED TO FIND THE MAXIMUM VALUE IN THE COLUMN THAT IS SUPPLIED AS A PARAMETER. IT CAN BE USED ON ANY TYPE OF DATA.*

*4.SUM FUNCTION*

*THIS FUNCTION SUMS UP THE VALUES IN THE COLUMN SUPPLIED AS PARAMETR.*

*5.STDDEV*

*THE STDDEV FUNCTION IS USED TO FIND STANDARD DEVIATION OF THE COLUMN SPECIFIED AS ARGUMENT.*

*6.VARIANCE FUNCTION*

*THIS FUNCTION IS USED TO FIND VARIANCE OF THE COLUMN SPECIFIED AS ARGUMENT.*

*QUEISTION:*

*1 CONSIDER THE SCHEMA EMPLOYEE(ID INT,NAME STRING,SALARY FLOAT)*

*? FIND MAX,MIN,AVG SALARY OF ALL EMPLOYEE*

*2 COSIDER THE SCHEMA EMPLOYE(ID INT,NAME STRING,SALARY FLOAT,DEPARTMANT STRING)*

*? RETRIVE THE TOTAL NO.OF EMPLOYEES IN THE SEARCH DEPARTMENT*

*QUERIES*

*1 \*CREATE DATABASE EMP;*

*\*USE EMP;*

*\*CREATE TABLE EMPLOYEE(ID INT,NAME VARCHAR(200),SALARY INT);*

*\*INSERT INTO EMPLOYEE VALUES((1,’JACK’,25000),(2,’JOHN’,35000),(3,’GEORGE’,20000));*

*$ SELECT \* FROM EMPLOYEE;*

*@ SELECT MAX(SALARY),MIN(SALARY),AVG(SALARY) FROM EMPLOYEE;*

*2 \* CREATE TABLE EMPLOYE(ID INT,NAME VARCHAR(200),SALARY FLOAT,DEPARTMENT VARCHAR(200) );*

*\* INSERT INTO EMPLOYE VALUES((1,’JACK’,25000,RESEARCH),(2,’JOHN’,35000,ELECTRICAL),(3,’GEORGE’,20000,RESEARCH));*

*& SELECT \* FROM EMPLOYE;*

*! SELECT COUNT(\*) FROM EMPLOYE WHERE DEPARTMENT = ‘RESEARCH’;*

*OUTPUT*

*$*

|  |  |  |
| --- | --- | --- |
| **ID** | **NAME** | **SALARY** |
| 1 | JACK | 25000 |
| 2 | JOHN | 35000 |
| 3 | GEORGE | 20000 |

*@*

|  |  |  |
| --- | --- | --- |
| **MAX(SALARY)** | **MIN(SALARY)** | **AVG(SALARY)** |
| 35000 | 20000 | 2666.66 |

*&*

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **NAME** | **SALARY** | **DEPARTMENT** |
| 1 | JACK | 25000 | RESEARCH |
| 2 | JOHN | 35000 | ELECTRICAL |
| 3 | GEORGE | 20000 | RESEARCH |

*!*

|  |
| --- |
| **COUNT(\*)** |
| 2 |

**JOIN OPERATION**

The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

There are different types of joins available in SQL

1.Cross Join

2.Inner Join

3.Left Outer Join

4.Right Outer Join

5.Full Outer Join

**1.CROSS JOIN**

The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to either True or where the join-condition is absent from the statement.

**2.INNER JOIN**

The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

**3.NATURAL JOIN**

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.

**4.LEFT OUTER JOIN**

All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

**5.RIGHT OUTER JOIN**

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

**6.FULL OUTER JOIN**

All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

**QUESTIONS:**

Consider the schemas

class (id integer,name string)

class\_info(id integer, name string)

Query:

CREATE TABLE CLASS(ID INT,NAME VARCHAR(20));

INSERT INTO CLASS VALUES ((1,’ATHIRA’),(2,’CHANCHAL’),(3,’DILNA’));

CREATE TABLE CLASS\_INFO(ID INT,NAME VARCHAR(20)); INSERT INTO CLASS\_INFO VALUES ((4,’PRASILA’),(5,’PRAJEESH’));

**OUTPUT**

CLASS:

|  |  |
| --- | --- |
| ID | NAME |
| 1 | ATHIRA |
| 2 | CHANCHAL |
| 3 | DILNA |

CLASS\_INFO:

|  |  |
| --- | --- |
| ID | NAME |
| 4 | PRASILA |
| 5 | PRAJEESH |

1. Find the cross join product of these two tables

Query:

SELECT CLASS.ID , CLASS.NAME , CLASS\_INFO.ID , CLASS\_INFO.NAME FROM CLASS,CLASS\_INFO ;

**OUTPUT**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | NAME | ID | NAME |
| 1 | ATHIRA | 4 | PRASILA |
| 1 | CHANCHAL | 4 | PRAJEESH |
| 1 | DILNA | 5 | PRASILA |
| 2 | ATHIRA | 5 | PRAJESSH |
| 2 | CHANCHAL |  |  |
| 2 | DILNA |  |  |
| 3 | ATHIRA |  |  |
| 3 | CHANCHAL |  |  |
| 3 | DILNA |  |  |

1. Find the inner join product of these two tables

QUERY:

SELECT \* FROM CLASS INNER JOIN CLASS\_INFO ON CLASS.ID = CLASS\_INFO.ID;

**OUTPUT:**

EMPTY SET

1. Find the natural join product of these two tables

QUERY:

INSERT INTO CLASS\_INFO VALUES(2,’CHANCHAL’);

SELECT \* FROM CLASS NATURAL JOIN CLASS\_INFO;

**OUTPUT:**

|  |  |
| --- | --- |
| ID | NAME |
| 2 | CHANCHAL |

d.1)Find left outer join

Query:

SELECT \* FROM CLASS LEFT OUTER JOIN CLASS\_INFO ON CLASS.ID=CLASS\_INFO.ID;

|  |  |  |  |
| --- | --- | --- | --- |
| ID | NAME | ID | NAME |
| 1 | ATHIRA | NULL | NULL |
| 2 | CHANCHAL | 2 | CHANCHAL |
| 3 | DILNA | NULL | NULL |

2. Find right outer join

Query:

SELECT \* FROM CLASS RIGHT OUTER JOIN CLASS\_INFO ON CLASS.ID=CLASS\_INFO.ID;

**OUTPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | NAME | ID | NAME |
| NULL | NULL | 4 | PRASILA |
| NULL | NULL | 5 | PRAJEESH |
| 2 | CHANCHAL | 2 | CHANCHAL |

3.Find full outer join

Query:

SELECT \* FROM CLASS LEFT OUTER JOIN CLASS\_INFO ON CLASS.ID=CLASS\_INFO.ID UNION ALL SELECT \* FROM CLASS RIGHT OUTER JOIN CLASS\_INFO ON CLASS.ID=CLASS\_INFO.ID;

**OUTPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | NAME | ID | NAME |
| NULL | NULL | NULL | NULL |
| 2 | CHANCHAL | 2 | CHANCHAL |
| NULL | NULL | NULL | NULL |

**CONSTRAINTS**

# Constraints in DBMS

Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table. The whole purpose of constraints is to maintain the **data integrity** during an update/delete/insert into a table. In this tutorial we will learn several types of constraints that can be created in RDBMS.

## Types of constraints

* NOT NULL
* UNIQUE
* DEFAULT
* CHECK
* Key Constraints – PRIMARY KEY, FOREIGN KEY
* Domain constraints
* Mapping constraints

#### NOT NULL:

NOT NULL constraint makes sure that a column does not hold NULL value. When we don’t provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

#### UNIQUE:

UNIQUE Constraint enforces a column or set of columns to have unique values. If a column has a unique constraint, it means that particular column cannot have duplicate values in a table.

#### DEFAULT:

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

#### CHECK:

This constraint is used for specifying range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

## Key constraints:

#### PRIMARY KEY:

Primary key uniquely identifies each record in a table. It must have unique values and cannot contain nulls. In the below example the ROLL\_NO field is marked as primary key, that means the ROLL\_NO field cannot have duplicate and null values.

#### FOREIGN KEY:

Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.  
Read more about it.

#### Domain constraints:

Each table has certain set of columns and each column allows a same type of data, based on its data type. The column does not accept values of any other data type.  
 Domain constraints are **user defined data type** and we can define them like this:

Domain Constraint = data type + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT)

#### Mapping constraints

Mapping constraints can be explained in terms of mapping cardinality:

**Mapping Cardinality**:  
**One to One**: An entity of entity-set A can be associated with at most one entity of entity-set B and an entity in entity-set B can be associated with at most one entity of entity-set A.

**One to Many**: An entity of entity-set A can be associated with any number of entities of entity-set B and an entity in entity-set B can be associated with at most one entity of entity-set A.

**Many to One**: An entity of entity-set A can be associated with at most one entity of entity-set B and an entity in entity-set B can be associated with any number of entities of entity-set A.

**Many to Many**: An entity of entity-set A can be associated with any number of entities of entity-set B and an entity in entity-set B can be associated with any number of entities of entity-set A.

We can have these constraints in place while creating tables in database.

**QUESTIONS**

(A).Create a table of person and order (person\_ID int, Lname string, Fname string, age int)

(order\_ID int, order\_no int, person\_ID int). The following constraints should be satisfied.

1.Lname , Fname and person\_ID columns will not accept NULL values.

2.Create a unique constraints on the ID column when the person table is created.

3.Create primary key - ‘person\_ID’ for person table and order\_ID for order table.

4.Set person\_ID as the foreign key of order table.

5.Create a check constraint on the age column check constraint should ensure that you cannot have any person below 18 years.

6.Set default value for city column when the person table is created.

7.Create an index for person table using create index constraints.

**QUERIES**

1. CREATE TABLE PERSON(PID INT NOT NULL,LNAME VARCHAR(200) NOT NULL,FNAME VARCHAR(200) NOT NULL,AGE INT);

2. ALTER TABLE PERSON ADD UNIQUE(PID);

3. ALTER TABLE PERSON ADD PRIMARY KEY(PID);

CREATE TABLE ORDER(OID INT NOT NULL,ONO INT, PID INT);

ALTER TABLE ORDER ADD PRIMARY KEY(OID);

4. ALTER TABLE ORDER ADD FOREIGN KEY(PID) REFERENCES PERSON(PID);

5. ALTER TABLE PERSON ADD CHECK(AGE < 18);

6.ALTER TABLE PERSON ADD (CITY VARCHAR(200) DEFAULT ‘MANJERI’);

7.CREATE INDEX I ON PERSON(PID);

INSERT INTO PERSON VALUES ((1,’ADHI’,’CHANDHRAN’,20),(2,’ANUPAMA’,’KTK’,21),(3,’FARHA’,’JAFAR’,21));

SELECT \* FROM PERSON;

**OUTPUT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PID | LNAME | FNAME | AGE | CITY |
| 1 | CHANDRAN | ADHI | 20 | MANJERI |
| 2 | KTK | ANUPAMA | 21 | MANJERI |
| 3 | JAFAR | FARHA | 21 | MANJERI |

(B) 1. Ensure that child records in related table have a parent record.

2. Ensure that a column contains a value within a specific set range.

3. Delete a child record when the parent record is deleted.

**QUERIES**

CREATE TABLE PARENT(RNO INT PRIMARY KEY, NAME VARCHAR(200), AGE INT ON DELETE CASCADE);

INSERT INTO PARENT VALUES((1,’ADHI’,20),(2,’ANU’,21),(3,’FARHA’,21),(4,’ABHI’,22),(5,’ANJU’,20));

CREATE TABLE CHILD(DEP\_ID INT, DEP\_NAME VARCHAR(200), RNO INT FOREIGN KEY(RNO) REFERENCES PARENT (RNO));

INSERT INTO CHILD VALUES((100,’CSE’,1),(200,’EEE’,2),(300,’ECE’,3),(400,’CIVIL’,4),(500,’MECH’,5));

SELECT \* FROM PARENT;

SELECT \* FROM CHILD;

DELETE FROM PARENT WHERE RNO=3;

SELECT \* FROM CHILD;

SELECT RNO, NAME, AGE FROM PARENT WHERE AGE BETWEEN 18 AND 20;

SELECT \* FROM PARENT;

**OUTPUT**

1. PARENT

|  |  |  |
| --- | --- | --- |
| RNO | NAME | AGE |
| 1 | ADHI | 20 |
| 2 | ANU | 21 |
| 3 | FARHA | 21 |
| 4 | ABHI | 22 |
| 5 | ANJU | 20 |

CHILD

|  |  |  |
| --- | --- | --- |
| DEP\_ID | DEP\_NAME | RNO |
| 100 | CSE | 1 |
| 200 | EEE | 2 |
| 300 | ECE | 3 |
| 400 | CIVIL | 4 |
| 500 | MECH | 5 |

2. CHILD

|  |  |  |
| --- | --- | --- |
| RNO | DEP\_NAME | RNO |
| 100 | CSE | 1 |
| 200 | EEE | 2 |
| 400 | CIVIL | 4 |
| 500 | MECH | 5 |

3. PARENT

|  |  |  |
| --- | --- | --- |
| RNO | NAME | AGE |
| 1 | ADHI | 20 |
| 5 | ANJU | 20 |

Group 8

Topic:Set operations

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

**UNION** is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its resultset. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.

**UNION** **ALL**-This operation is similar to Union. But it also shows the duplicate rows

**Intersect** operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements

The **Minus** operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.

Questions:

1.consider the following schema.emptest(empid int,empname string,empaddr string,empssn string),empdesign(empid int,empname string,empaddr string,empssn string).perform the following operations on the table.

1.union

2.Union all

3.Intersect

4.Minus

Create statement

create table emptest(empid int,empname varchar(20),empaddr

varchar(20),empssn varchar(20));

create table empdesign(empid int,empname varchar(20),empaddr

varchar(20),empssn varchar(20));

insert into emptest values((1,’akshay’,’bnglr’,11)(2,’anandhu’,’bnglr’,22)(3,’hadi’,bnglr’,33)(4,’jithin’,’bnglr’,44));

insert into empdesign values((1,’akshay’,’bnglr’,11)(4,’jithin’,’bnglr’,44));

query:

1.select from emptiest union empdesign;

2.select from emptiest union all empdesign;

3.mysql does not support intersect,intersect done using join

Select emptest.empid,emptest.empname from emptest on emptest.empid=empdesign.empid and emptest.empname=empdesign.empname;

4.mysql does not support minus,minus done using join.

Select emptest.empid,emptest.empname from emptest join empdesign using (empid,empname)

emptest

|  |  |  |  |
| --- | --- | --- | --- |
| Empid | empname | empaddr | Empssn |
| 1 | Akshay | bnglr | 11 |
| 2 | Anandhu | Bnglr | 22 |
| 3 | Hadi | Bnglr | 33 |
| 4 | jithin | bnglr | 44 |

Empdesign

|  |  |  |  |
| --- | --- | --- | --- |
| empid | empname | empaddr | Empssn |
| 1 | Akshay | Bnglr | 11 |
| 4 | jithin | Bnglr | 44 |

Output

1.

|  |  |  |  |
| --- | --- | --- | --- |
| Empid | empname | empaddr | Empssn |
| 1 | Akshay | bnglr | 11 |
| 2 | Anandhu | Bnglr | 22 |
| 3 | Hadi | Bnglr | 33 |
| 4 | jithin | bnglr | 44 |

2.

|  |  |  |  |
| --- | --- | --- | --- |
| Empid | empname | empaddr | Empssn |
| 1 | Akshay | bnglr | 11 |
| 2 | Anandhu | Bnglr | 22 |
| 3 | Hadi | Bnglr | 33 |
| 4 | jithin | bnglr | 44 |
| 1 | akshay | bnglr | 11 |
| 4 | jithin | bnglr | 44 |

3.

|  |  |
| --- | --- |
| Empid | Empname |
| 1 | Akshay |
| 4 | Jtihin |

4.

|  |  |
| --- | --- |
| empid | Empname |
| 1 | Akshay |
| 4 | Jtihin |

2.consider the following schema supplier(supply id int,supply\_name varchar(20)) company(companyid int,company name varchar(20))

Retrieve the id value and name of the supplier whose id greater than 1000 and companies whose id value is >1000,rename resultant table to as id value and name value

**Create**

Create table supplier(sid int,sname varchar(20));

Create table company(cid int,cname varchar(20));

Insert into company values((500,’intel’)(2000,’dell’)(3000,’hp’));

Insert into supplier values((1000,’kt’)(3000,’hadi’)(5000,’jithin’));

Create table in(idvalue cid,namevalue varchar(20),id\_value int,name\_value varchar(20));

**Query**

Insert into in select sid,snamecid,cid,cname from supplier,company where supply.sid in(select sid from supply where sid>2000 and company.cid in (select cid from company whose cid>1000);

OUTPUT

Supply company

|  |  |
| --- | --- |
| sname | Sid |
| Kt | 500 |
| Hadi | 3000 |
| jithin | 5000 |
| cname | cid |
| Intel | 500 |
| Hp | 3000 |
| dell | 2000 |

|  |  |  |  |
| --- | --- | --- | --- |
| idvalue | namevalue | Id\_value | Name\_value |
| 3000 | Hadi | 3000 | Hp |
| 5000 | jithin | 2000 | Dell |

BUILT IN FUNCTIONS

Create an employee table with eid, ename, salary, hire date.

CREATE TABLE EMPLOYEE(EID INT,ENAME VARCHAR(20),SALARY FLOAT,HIREDATE DATE);

INSERT INTO EMPLOYEE VALUES(1,'JOHN',23000,'2018-10-22'),(2,'JACK',2000,'2018-03-03'),(3,'FRANCIS',2000,'2018-10-27'),(4,'JIMMY',29045,'2018-10-27');

SELECT \*FROM EMPLOYEE;

|  |  |  |  |
| --- | --- | --- | --- |
| EID | ENAME | SALARY | HIREDATE |
| 1 | JOHN | 23000 | 2018-10-22 |
| 2 | JACK | 25000 | 2018-03-03 |
| 3 | FRANCIS | 2000 | 2018-05-20 |
| 4 | JIMMY | 29045 | 2018-10-27 |

**1.NUMERIC FUNCTIONS:**

a. Find out per day salary of each employee .use 'ROUND' function to round off to two decimal places.

QUERY:

SELECT ROUND (SALARY/30,2) FROM EMPLOYEE;

|  |
| --- |
| ROUND(SALARY/30,2) |
| 766.67 |
| 833.33 |
| 66.67 |
| 968.16 |

OUTPUT:

b. Find out whether the salary of each employee is odd or even using 'MOD' function.

QUERY:

SELECT MOD (SALARY,2) FROM EMPLOYEE;

OUTPUT:

|  |
| --- |
| MOD(SALARY,2) |
| 0 |
| 0 |
| 0 |
| 1 |

**2.CHARACTER FUNCTION:**

a. Display name of each employee is uppercase and lowercase.

QUERY:

SELECT UPPER(ENAME) AS ENAME FROM EMPLOYEE;

|  |
| --- |
| ENAME |
| JOHN |
| JACK |
| FRANCIS |
| JIMMY |

SELECT LOWER(ENAME) AS ENAME FROM EMPLOYEE;

|  |
| --- |
| ENAME |
| john |
| jack |
| francis |
| jimmy |

b. Display name of each employee with first letter in uppercase.

QUERY:

UPDATE EMPLOYEE SET ENAME = CONCAT(UCASE(LEFT(ENAME,1)),LCASE(SUBSTRING(ENAME,2)));

SELECT ENAME FROM EMPLOYEE;

OUTPUT:

|  |
| --- |
| ENAME |
| John |
| Jack |
| Francis |
| Jimmy |

c. Add space to the names of each employee using 'RPAD' function.

QUERY:

SELECT ENAME,RPAD(ENAME,10," ") FROM EMPLOYEE;

OUTPUT:

|  |  |
| --- | --- |
| ENAME | RPAD(ENAME,10," ") |
| John | John |
| Jack | Jack |
| Francis | Francis |
| Jimmy | Jimmy |

d. Remove the white space from ename using 'TRIM' function.

QUERY:

SELECT ENAME,TRIM(ENAME) FROM EMPLOYEE;

OUTPUT:

|  |  |
| --- | --- |
| ENAME | TRIM(ENAME) |
| John | John |
| Jack | Jack |
| Francis | Francis |
| Jimmy | Jimmy |

e. Select substring of each ename from character position 1 to 3.

QUERY:

SELECT ENAME,SUBSTRING(ENAME,1,3) FROM EMPLOYEE;

OUTPUT:

|  |  |
| --- | --- |
| ENAME | SUBSTRING(ENAME,1,3) |
| John | Joh |
| Jack | Jac |
| Francis | Fra |
| Jimmy | Jim |

f. Display eid, ename and length of ename from emp.

QUERY:

SELECT EID,ENAME,LENGTH(ENAME) FROM EMPLOYEE;

OUTPUT:

|  |  |  |
| --- | --- | --- |
| EID | ENAME | LENGTH(ENAME) |
| 1 | John | 4 |
| 2 | Jack | 4 |
| 3 | Francis | 7 |
| 4 | Jimmy | 5 |

g. Replace the name of employee John with Ram.

QUERY:

SELECT REPLACE (ENAME,'JOHN','RAM') FROM EMPLOYEE;

OUTPUT:

|  |
| --- |
| ENAME |
| Ram |
| Jack |
| Francis |
| Jimmy |

**3.DATE FUNCTIONS:**

a. Display days employed for each employee using 'DATEDIFF' function.

QUERY:

SELECT DATEDIFF('2019-03-01',HIREDATE) AS DAYS-EMPLOYED

OUTPUT:

|  |
| --- |
| DAYS-EMPLOYED |
| 72 |
| 305 |
| 227 |
| 66 |

b. Extract and display year hired for each employee from employee table.

QUERY:

SELECT YEAR(HIREDATE) AS YEAR FROM EMPLOYEE;

OUTPUT:

|  |
| --- |
| YEAR |
| 2018 |
| 2018 |
| 2018 |
| 2018 |

c. Extract and concatenate the year, month and day from 'SYSDATE'.

QUERY:

SELECT YEAR(HIREDATE),NOW() FROM EMPLOYEE;

OUTPUT:

|  |  |
| --- | --- |
| YEAR(HIREDATE) | NOW() |
| 2018 | 2018-11-07 10:55:42 |
| 2018 | 2018-11-07 10:55:42 |
| 2018 | 2018-11-07 10:55:42 |
| 2018 | 2018-11-07 10:55:42 |

**4.CONVERTION FUNCTION:**

a. Convert the system date to the format

DD-MM-YYYY --> HH24:M1:SS

QUERY:

SELECT DATE\_FORMAT(NOW(),"%H:%I:%S");

OUTPUT:

|  |
| --- |
| DATE\_FORMAT(NOW(),"%H:%I:%S") |
| 11:21:13 |

CREATING VIEW

53 -SHAMNA HIBA K.P

54 -SRUTHI BABU

55 -SURYA MANOHARAN K

56- VAISHNAVI N P

57- VIDHURAM N

**Create table employee:**

create table employee(eno int(4),ename varchar(20),job varchar(20),mgr int(4),hiredate date,salary float(7,2),deptno int(2));

insert into employee values(1,'john','engineer',22,'2017-07-09',398888,1), (2,'liya','designer',76,'2018-07-22',8777,2),(5,'martin','professor',11,'2017-01-04',1000,1),(9,'marx','engineer',18,'2016-01-03',45000,2);

select \* from employee;

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

| eno | ename | job | mgr | hiredate | salary | deptno |

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

| 1 | john | engineer | 22 | 2017-07-09 | 99999.99 | 1 |

| 2 | liya | designer | 76 | 2018-07-22 | 8777.00 | 2 |

| 5 | martin | professor | 11 | 2017-01-04 | 1000.00 | 1 |

| 9 | marx | engineer | 18 | 2016-01-03 | 45000.00 | 2 |

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

**Create table department:**

create table department(dno int(40),dname varchar(20),loc varchar(30));

insert into department values(1,'cse','floorone'),(2,'eee','floortwo'),(3,'mech','floorthree');

select \* from department;

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

| dno | dname | loc |

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

| 1 | cse | floorone |

| 2 | eee | floortwo |

| 3 | mech | floorthree |

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

**Q.1) Create a view of the table employee as empview which shows the employee in department 2 and their annual salary:**

create view empview as select ename,salary\*12 from employee where deptno=2;

select \* from empview;

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

| ename | salary\*12 |

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

| liya | 105324.00 |

| marx | 540000.00 |

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

**Create table order details:**

create table order\_details(orderno int(4),productcode int(4),quantityorder int(4),priceeach float,orderlino int(4));

insert into order\_detais values(22,11,2,100,66),(33,22,3,200,88),(44,44,2,500,11);

insert into order\_details values(22,11,2,100,66),(33,22,3,200,88),(44,44,2,500,11);

select \* from order\_details;

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

| orderno | productcode | quantityorder | priceeach | orderlino |

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

| 22 | 11 | 2 | 100 | 66 |

| 33 | 22 | 3 | 200 | 88 |

| 44 | 44 | 2 | 500 | 11 |

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

select orderno,(quantityorder\*priceeach) total\_sales from order\_details;

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

| orderno | total\_sales |

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

| 22 | 200 |

| 33 | 600 |

| 44 | 1000 |

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

**Q.2) Create a view that represents total sales per order:**

create view sales\_per\_order as select orderno,(quantityorder\*priceeach) as total\_sales from order\_details;

select \* from sales\_per\_order;

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

| orderno | total\_sales |

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

| 22 | 200 |

| 33 | 600 |

| 44 | 1000 |

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

**Q.3) Create a view called 'big sales order' based on the 'sales per order' view to show every sales order whose total is greater than 200:**

create view big\_sales\_order as select orderno,total\_sales from sales\_per\_order where total\_sales>200;

select \* from big\_sales\_order;

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

| orderno | total\_sales |

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

| 33 | 600 |

| 44 | 1000 |

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

**Q.4) Create a view to show associated department to employee:**

create view show\_dept as select ename,dname from employee,department where deptno=dno;

select \* from show\_dept;

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

| ename | dname |

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

| john | cse |

| liya | eee |

| martin | cse |

| marx | eee |

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

**Q.5) Update the view to show that employee name in uppercase:**

alter view show\_dept as select upper(ename),dname from employee,department where deptno=dno;

select \* from show\_dept;

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

| upper(ename) | dname |

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

| JOHN | cse |

| LIYA | eee |

| MARTIN | cse |

| MARX | eee |

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

**Q.6) Delete the constructed view:**

drop view show\_dept;