NORMALIZATION

Normalization

Database normalization is the process of efficiently organizing data in a database. It is a set or rules/ guidelines / statements that we follow while storing the data.

There are two reasons of the normalization process:

- Eliminating redundant data, for example, storing the same data in more than one tables.
- Ensuring data dependencies make sense.

First Normal Form (1NF)

- Define the data items. This means looking at the data to be stored, organizing the data into columns, defining what type of data each column contains, and finally putting related columns into their own table.

- Ensure that there are no repeating groups of data

- Ensure that there is a primary key.

Second Normal Form (2NF)

- It should meet all the rules for 1NF
- There must be no partial dependences of any of the columns on the primary key

Third Normal Form (3NF)

- It should meet all the rules for 2NF
- Tables should have relationship.

CONSTRAINTS

SQL Constraints:

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Following are commonly used constraints available in SQL:

- PRIMARY Key: Uniquely identified each rows/records in a database table.
- UNIQUE Constraint: Ensures that all values in a column are different.
- NOT NULL Constraint: Ensures that a column cannot have NULL value.
- DEFAULT Constraint : Provides a default value for a column when none is specified. CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.
- FOREIGN Key: Uniquely identified a rows/records in any another database table.

NOT NULL Constraint:

By default, a column can hold NULL values. If we do not want a column to have a NULL value then we need to define such constraint on this column specifying that NULL is now not allowed for that column.

```
CREATE TABLE SALESS(
    ID   INT            NOT NULL,
    EID VARCHAR (20)     NOT NULL,
    AGE  INT            NOT NULL,
    ADDRESS  CHAR (25) ,
    SALARY   DECIMAL (18, 2)
);

ALTER TABLE SALESS
  ALTER COLUMN SALARY  DECIMAL (18, 2) NOT NULL;
```

DEFAULT Constraint:

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value

```
CREATE TABLE SALESS(
    ID   INT            NOT NULL,
    EID VARCHAR (20)     NOT NULL,
    AGE  INT            NOT NULL,
    ADDRESS  CHAR (25) ,
    SALARY   DECIMAL (18, 2) DEFAULT 5000.00
);

ALTER TABLE SALESS
  ADD CONSTRAINT DSAL DEFAULT 5000.00 FOR SALARY;

ALTER TABLE SALESS
 DROP CONSTRAINT DSAL;
```

UNIQUE Constraint:

The UNIQUE constraint provides a unique value to a column.

```
CREATE TABLE SALESS(
    ID   INT            NOT NULL,
    EID VARCHAR (20)     NOT NULL,
    AGE  INT            NOT NULL UNIQUE,
    ADDRESS  CHAR (25) ,
    SALARY   DECIMAL (18, 2) DEFAULT 5000.00
);

ALTER TABLE SALESS
  ADD CONSTRAINT <CONSTRAINT EID > UNIQUE (AGE);

ALTER TABLE SALESS
  ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, SALARY);

ALTER TABLE SALESS
  DROP CONSTRAINT myUniqueConstraint;
```

CHECK Constraint:

The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and it s not entered into the table.

```
CREATE TABLE SALESS(
    ID   INT            NOT NULL,
    EID VARCHAR (20)     NOT NULL,
    AGE  INT            NOT NULL  CHECK (AGE > 18),
    ADDRESS  CHAR (25) ,
    SALARY   DECIMAL (18, 2) DEFAULT 5000.00
);
```

```
ALTER TABLE SALESS
  ADD CONSTRAINT ckAge CHECK (AGE > 18);
```

```
ALTER TABLE SALESS
  DROP CONSTRAINT ckAge;
```

PRIMARY KEY Constraint:

A primary key is a field in a table which uniquely identifies the each rows/records in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.

A table can have only one primary key which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key.

```
CREATE TABLE SALESS(
    ID   INT            NOT NULL,
    EID VARCHAR (20)     NOT NULL,
    AGE  INT            NOT NULL,
    ADDRESS  CHAR (25) ,
    SALARY   DECIMAL (18, 2) ,
    PRIMARY KEY (ID)
);

ALTER TABLE SALESS
  ADD CONSTRAINT pkID  PRIMARY KEY (ID);

ALTER TABLE SALESS
  DROP CONSTRAINT pkID;
```

FOREIGN KEY Constraint:

A foreign key is a key used to link two tables together. This is sometimes called a referencing key.

```
CREATE TABLE SALESS(
    ID   INT            NOT NULL,
    EID VARCHAR (20)     NOT NULL,
    AGE  INT            NOT NULL,
    ADDRESS  CHAR (25) ,
    PRIMARY KEY (ID)
);

CREATE TABLE ORDERS(
    OID   INT            NOT NULL,
     CUST_ID INT REFERENCES SALESS (ID),
     ODATE  DATE,
     QTY INT,
     PRICE INT
);

ALTER TABLE ORDERS
ADD CONSTRAINTS  FKID  FOREIGN KEY (CUST_ID) REFERENCES  SALESS (ID);
```

ASSIGNMENT

ASSIGNMENT – 3

CREATE TWO TABLES EMP & EMP_SAL AS PER THE BELOW STRUCTURE:

EMP	
Field EID	Constraints
EMPID	Primary Key
NAME	NOT NULL
ADDR	No employee from UTTAM NAGAR
CITY	DEL, GGN, FBD, NOIDA
PHNO	UNIQUE
EMAIL	Should be on Gmail / Yahoo Domain
DOB	<= '1-Jan-2000'

EMP_SAL	
Field EID	Constraints
EMPID	Foreign Key
DEPT	HR, MIS, OPS , IT ADMIN, TEMP
DESI	ASSO, MGR, VP, DIR
BASIC	>=20000
DOJ	-

By default DEPT should be TEMP

CLAUSES

SQL CLAUSES

SQL BETWEEN Clause

SELECT column1, column2....columnN FROM table_EID WHERE column_EID BETWEEN val-1 AND val-2;

SQL IN Clause

SELECT column1, column2....columnN

FROM table_EID

WHERE column_EID IN (Val1, Val2... Valn);

SQL Like Clause

SELECT column1, column2....columnN FROM table_EID WHERE column_EID LIKE { PATTERN}

SQL COUNT Clause

SELECT COUNT(column_EID) FROM table_EID WHERE CONDITION;

SQL DISTINCT Clause

SELECT DISTINCT (column) FROM table_EID;

SQL CLAUSES

SQL ORDER BY Clause

```
SELECT column1, column2....columnN
FROM   table_EID
WHERE  CONDITION
ORDER BY column_EID {ASC|DESC};
```

SQL GROUP BY Clause

```
SELECT SUM(column_EID)
FROM   table_EID
WHERE  CONDITION
GROUP BY column_EID;
```

SQL HAVING Clause

```
SELECT SUM(column_EID)
FROM table_EID
WHERE  CONDITION GROUP BY column_EID
HAVING (arithematic function condition);
```

ASSIGNMENT – 4

In the EMP table display :

CITY WISE COUNT OF EMPLOYEES ARRANGED IN DESCENDING ORDFER

DETAILS OF THE EMPLOYEES WHO DOES NOT HAVE AN ACCOUNT ON YAHOO DOMAIN

From the Emp_Sal table display:

DESIGNATION WISE TOTAL COST AND NUMBER OF MEMBERS ARRANGED IN DESCENDING ORDER OF THE TOTAL COST

JOINS

SQL Joins

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.

SQL Join Types:

- INNER JOIN: returns rows when there is a match in both tables.
- LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN: returns rows when there is a match in one of the tables.
- CARTESIAN JOIN: returns the cartesian product of the sets of records from the two or more joined tables.
- SELF JOIN: is used to join a table to itself, as if the table were two tables, temporarily renaming at least one table in the SQL statement.

INNER JOIN

The most frequently used and important of the joins is the INNER JOIN. They are also referred to as an EQUIJOIN..

```
SELECT table1.column1, table2.column2...
FROM table1
INNER JOIN table2
ON table1.common_filed = table2.common_field;
```

LEFT JOIN

The SQL Left Join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching.

```
SELECT table1.column1, table2.column2...
FROM table1
LEFT JOIN table2
ON table1.common_filed = table2.common_field;
```

RIGHT JOIN

The SQL Right Join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching.

```
SELECT table1.column1, table2.column2...
FROM table1
RIGHT JOIN table2
ON table1.common_filed = table2.common_field;
```

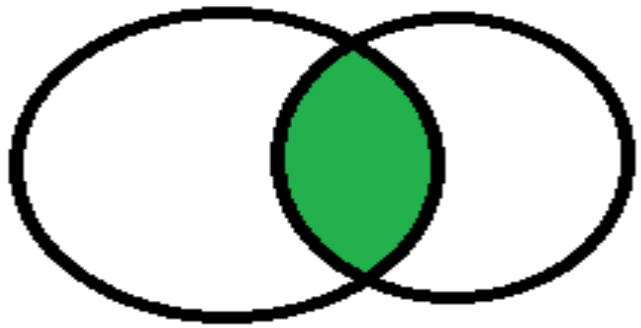
FULL JOIN

The SQL FULL JOIN combines the results of both left and right outer joins.

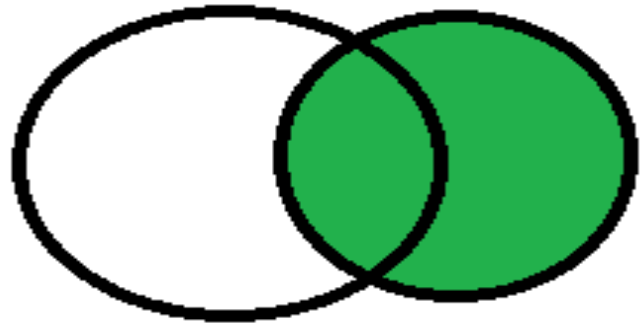
```
SELECT table1.column1, table2.column2...
FROM table1
FULL JOIN table2
ON table1.common_filed = table2.common_field;
```

JOINS

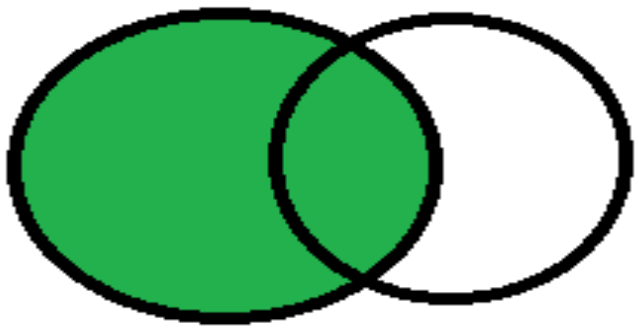
Inner Join



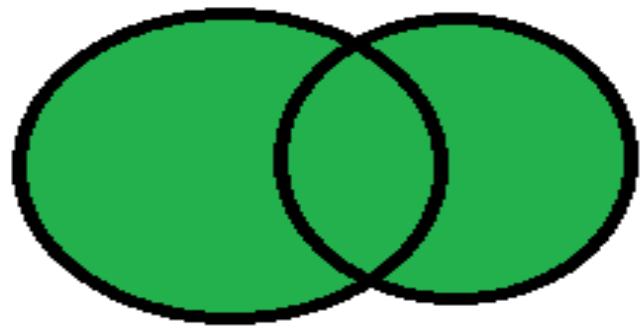
Right Join



Left Join



Full Join



CARTESIAN JOIN

- The CARTESIAN JOIN or CROSS JOIN returns the cartesian product of the sets of records from the two or more joined tables.
- It produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table if no WHERE clause is used along with CROSS JOIN.
- If WHERE clause is used with CROSS JOIN, it functions like an INNER JOIN.

```
SELECT table1.column1, table2.column2...
FROM table1
CROSS JOIN table2
```

```
SELECT table1.column1, table2.column2...
FROM table1
CROSS JOIN table2
WHERE table1.common_filed = table2.common_field;
```

SELF JOIN

The SQL SELF JOIN is used to join a table to itself, as if the table were two tables, temporarily renaming at least one table in the SQL statement.

```
SELECT a.column_EID, b.column_EID...
FROM table1 a, table1 b
 WHERE a.common_filed = b.common_field;
```

SELF JOIN