Structure Query Language (SQL)

6.12.2 OR operator

OR operator is also used to combine multiple conditions with *Where* clause. The only difference between AND and OR is their behavior. When we use AND to combine two or more than two conditions, records satisfying all the condition will be in the result. However, in case of OR, at least one condition from the conditions specified must be satisfied by any record to be in the result.

Example of OR

Consider the following **Emp** table

Eid	Name	Age	Salary
401	Anu	22	5000
402	Shane	29	8000
403	Rohan	34	12000
404	Scott	44	10000
405	Tiger	35	9000

```
SELECT * from Emp WHERE salary > 10000 OR age > 25
```

The above query will return records where either salary is greater than 10000 or age greater than 25.

402	Shane	29	8000
403	Rohan	34	12000
404	Scott	44	10000
405	Tiger	35	9000

6.13 SQL Constraints

SQl Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into following two types,

- **Column level constraints :** limits only column data
- **Table level constraints :** limits whole table data

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

6.13.1 NOT NULL Constraint

NOT NULL constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value. One important point to note about NOT NULL constraint is that it cannot be defined at table level.

Example using NOT NULL constraint

```
CREATE table Student(s_id int NOT NULL, Name varchar(60), Age int);
```

The above query will declare that the **s_id** field of **Student** table will not take NULL value.

6.13.2 UNIQUE Constraint

UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. UNIQUE constraint can be applied at column level or table level.

Example using UNIQUE constraint when creating a Table (Table Level)

```
CREATE table Student(s_id int NOT NULL UNIQUE, Name varchar(60), Ag
e int);
```

The above query will declare that the **s_id** field of **Student** table will only have unique values and won't take NULL value.

Example using UNIQUE constraint after Table is created (Column Level)

```
ALTER table Student add UNIQUE(s_id);
```

The above query specifies that **s_id** field of **Student** table will only have unique value.

6.13.3 Primary Key Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

Example using PRIMARY KEY constraint at Table Level

```
CREATE table Student (s_id int PRIMARY KEY, Name varchar(60) NOT NU
LL, Age int);
```

The above command will creates a PRIMARY KEY on the `s_id`.

Example using PRIMARY KEY constraint at Column Level

```
ALTER table Student add PRIMARY KEY (s_id);
```

The above command will creates a PRIMARY KEY on the `s_id`.

6.13.4 Foreign Key Constraint

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see it using two table.

Customer_Detail Table :

c_id	Customer_Name	Address
101	Adam	Noida
102	Alex	Delhi
103	Stuart	Rohtak

Order_Detail Table :

Order_id	Order_Name	c_id
10	Order1	101
11	Order2	103
12	Order3	102

In **Customer_Detail** table, c_id is the primary key which is set as foreign key in **Order_Detail** table. The value that is entered in c_id which is set as foreign key in **Order_Detail** table must be present in **Customer_Detail**table where it is set as primary key. This prevents invalid data to be inserted into c_id column of **Order_Detail**table.

Example using FOREIGN KEY constraint at Table Level

```
CREATE table Order_Detail(order_id int PRIMARY KEY,
order_name varchar(60) NOT NULL,
c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id));
```

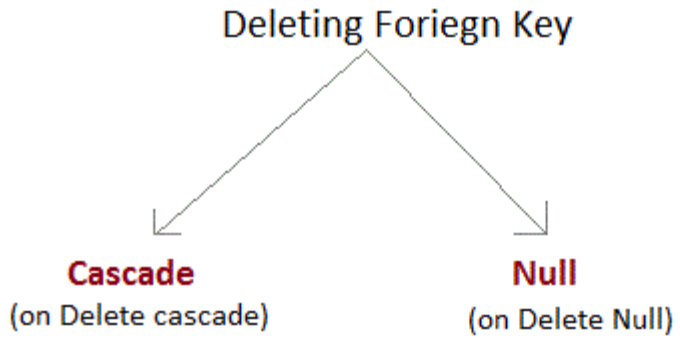
In this query, c_id in table Order_Detail is made as foriegn key, which is a reference of c_id column of Customer_Detail.

Example using FOREIGN KEY constraint at Column Level

```
ALTER table Order_Detail add FOREIGN KEY (c_id) REFERENCES Custome
r_Detail(c_id);
```

Behavior of Foreign Key Column on Delete

There are two ways to maintain the integrity of data in Child table, when a particular record is deleted in main table. When two tables are connected with Foreign key, and certain data in the main table is deleted, for which record exit in child table too, then we must have some mechanism to save the integrity of data in child table.



- **On Delete Cascade :** This will remove the record from child table, if that value of foreign key is deleted from the main table.

- **On Delete Null :** This will set all the values in that record of child table as NULL, for which the value of foreign key is selected from the main table.
- If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.

```
ERROR : Record in child table exist
```

6.13.4 CHECK Constraint

CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

Example using CHECK constraint at Table Level

```
create table Student(s_id int NOT NULL CHECK(s_id > 0),
Name varchar(60) NOT NULL,
Age int);
```

The above query will restrict the s_id value to be greater than zero.

Example using CHECK constraint at Column Level

```
ALTER table Student add CHECK(s_id > 0);
```

6.14 SQL Functions

SQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc. SQL functions are divided into two categories,

- Aggregate Functions

- Scalar Functions

6.14.1 Aggregate Functions

These functions return a single value after calculating from a group of values. Following are some frequently used Aggregate functions.

1) AVG( )

Average returns average value after calculating from values in a numeric column.

Its general Syntax is,

```
SELECT AVG(column_name) from table_name
```

Example using A

VG( )

Consider following **Emp** table

Eid	Name	Age	Salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find average of salary will be,

```
SELECT avg(salary) from Emp;
```

Result of the above query will be,

avg(salary)
8200

2) COUNT()

Count returns the number of rows present in the table either based on some condition or without condition.

Its general Syntax is,

```
SELECT COUNT(column_name) from table-name
```

Example using COUNT()

Consider following **Emp** table

Eid	Name	Age	Salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to count employees, satisfying specified condition is,

```
SELECT COUNT(name) from Emp where salary = 8000;
```

Result of the above query will be,

count(name)
2