Day 5

Constraint

- If we want store accurate data inside table then we should use constraints
- To store accurate data, we need to put/apply some restrictions on column. It is called constraint.
- We can apply constraints during table creation or after table creatiion using ALTER statement.
- SQL supports following constraints
    1. DEFAULT
    2. NOT NULL
    3. UNIQUE
    4. CHECK
    5. PRIMARY KEY
    6. FOREIGN KEY
- We can apply these constraints at two levels:
    1. Column Level
    2. Table Level

Default constraint

```
CREATE TABLE address
(
    city VARCHAR(50),
    state VARCHAR(50),
    pincode CHAR(6),
    country VARCHAR( 50 ) DEFAULT "India"
);
```

```
INSERT INTO address
VALUES
('Pune','MH','411057','India');

SELECT * FROM address
```

```
INSERT INTO address
(city, state,pincode)
VALUES
('Mumbai','MH','411001');
SELECT * FROM address;
```

```
INSERT INTO address
VALUES
('Mumbai','MH','411001',NULL);
SELECT * FROM address;
```

- Remove Default constraint

```
ALTER TABLE address
ALTER country DROP DEFAULT;
```

- SET Default constraint

```
ALTER TABLE address
ALTER country SET DEFAULT "India";
```

```
INSERT INTO address
(city, state,pincode)
VALUES
('Nasik','MH','415001');
SELECT * FROM address;
```

NOT NULL Constraint

- If we dont want to store NULL value inside record/row then we should use NOT NULL constraint on column.

```
CREATE TABLE employees
(
    id INT NOT NULL,
    name VARCHAR(50),
    salary FLOAT
);
```

```
INSERT INTO
employees
VALUES( 1, 'ABC', 10000);
```

```
INSERT INTO
employees
```

```
VALUES( NULL, 'PQR', 20000); -- Error
```

```
INSERT INTO
employees
(name, salary)
VALUES
( 'PQR', 20000); -- Error
```

```
UPDATE employees
SET id = NULL   -- Error
WHERE name='ABC';
```

- How to remove NOT NULL contraint

```
ALTER TABLE employees
MODIFY COLUMN id INT;

DESC employees;

INSERT INTO
employees
(name, salary)
VALUES
( 'PQR', 20000); -- OK
```

- Set NOT NULL constraint on column

```
ALTER TABLE employees
MODIFY COLUMN id INT NOT NULL;
```

UNIQUE Constraint

- If we want store unique value inside record/row then we should use UNIQUE constraint.

```
CREATE TABLE employees
(
    id INT UNIQUE,
    name VARCHAR(50),
    salary FLOAT,
    email VARCHAR(50),
    UNIQUE(email)
```

```
);
DESC employees;
```

```
INSERT INTO employees
VALUES
( 1, 'Sandeep', 45000.50, 'sandeep@gmail.com');
```

```
INSERT INTO employees
VALUES
(2, 'Soham', 65000.50, 'sandeep@gmail.com');
-- NOT OK
```

```
INSERT INTO employees
VALUES
(1, 'Soham', 65000.50, 'soham@gmail.com');
-- NOT OK
```

```
INSERT INTO employees
VALUES
(NULL, 'Soham', 65000.50, 'soham@gmail.com'); -- OK
```

- Table can contain multiple UNIQUE keys.
- Unique key can be NULL.

```
CREATE TABLE student
(
    roll INT,
    `div` CHAR(10),
    name VARCHAR(50),
    marks FLOAT,
    CONSTRAINT UC_STUDENT_ROLL_DIV UNIQUE(roll, `div`)
);
```

```
INSERT INTO
student
VALUES
( 1, 'A', 'Sandeep', 54.0 ),
( 1, 'B', 'Rahul', 80.0 ),
( 2, 'A', 'Ketan', 75.0 ),
( 2, 'B', 'Akash', 60.0 );
```

```
INSERT INTO
student
VALUES
( 1, 'A', 'Nilesh', 65.0 );
```

- Remove Unique Key Constraint

```
ALTER TABLE student
DROP INDEX UC_STUDENT_ROLL_DIV;
```

- Set Unique Key Constraint

```
ALTER TABLE student
ADD CONSTRAINT UC_STUDENT_ROLL_DIV UNIQUE(roll, `div`);
```

```
CREATE TABLE employee
(
    id INT,
    name VARCHAR(50),
    salary FLOAT,
    CONSTRAINT UC_EMPLOYEE_ID UNIQUE( ID )
);
```

CHECK Constraint

- It is introduced in MySQL 8.0.16.
- It is used to check range of values.

```
CREATE TABLE employees
(
    id INT NOT NULL UNIQUE,
    name VARCHAR(50) UNIQUE,
    salary FLOAT CHECK( salary > 0 ),
);
```

```
CREATE TABLE employees
(
    id INT NOT NULL UNIQUE,
    name VARCHAR(50) UNIQUE,
```

```
    salary FLOAT,
    CHECK( salary > 0 )
);
```

```
CREATE TABLE employees
(
    id INT NOT NULL UNIQUE,
    name VARCHAR(50) UNIQUE,
    salary FLOAT,
    CONSTRAINT CHK_EMPLOYEES_SALARY CHECK( salary > 0 )
);
```

```
INSERT INTO employees
VALUES( 1, 'ABC', 5000 );

INSERT INTO employees
VALUES( 2, 'PQR', -5000 ); --ERROR

INSERT INTO employees
VALUES( 2, 'PQR', NULL );
```

- Remove Check Constraint

```
ALTER TABLE employees
DROP CHECK CHK_EMPLOYEES_SALARY;

INSERT INTO employees
VALUES( 3, 'XYZ', -5000 );
```

- Set Check Constraint

```
ALTER TABLE employees
ADD CONSTRAINT CHK_EMPLOYEES_SALARY
CHECK( salary BETWEEN 8000 AND 20000 );
```

```
INSERT INTO employees
VALUES
(1, 'Sandeep', 15000 );
-- (1, 'Sandeep', 25000 ); -- NOT OK
-- (1, 'Sandeep', 5000 ); --NOt OK
```

PRIMARY KEY CONSTRAINT

- PRIMARY KEY = UNIQUE + NOT NULL
- Per table we can declare only one primary key.
- It can be single value or composite value.
- Primary key do not contain NULL value.
- If we want to idenitfy record in a table uniquely then we should delare primary key.
- Types of primary Key
    1. Natural Primary key
    2. Composite Primary Key
    3. Surrogate Primary Key
- If single value is sufficient to identify record in a table uniquely then it is called Natural primary key.
- Example
    - Customers
        - name
        - address
        - contact_number
        - email[PK]
    - Employee
        - name
        - empid[PK]
        - salary
        - dept
        - designation
- When single value is insufficient to identify record in a table uniquely then we can use combination of values as a primary key. It is called composite primary key.
- Example
    - Student
        - name
        - roll
        - div
        - marks
        - primary key[roll + div]
- If we use AUTO_GENERATED value from column as a primary key then it is called Surrogate primary key
- Example
    - Student
        - regid [AUTO_INCREMENT] [PK]
        - name
        - email
- Primary Key declaration at column level

```
CREATE TABLE employees
(
    empid INT PRIMARY KEY,
    name VARCHAR(50),
    salary FLOAT
);
```

- Primary Key declaration at table level

```
CREATE TABLE employees
(
    empid INT,
    name VARCHAR(50),
    salary FLOAT,
    -- PRIMARY KEY( empid ) -- OK
    CONSTRAINT PK_EMPLOYEES_EMPID PRIMARY KEY( empid )
);
```

```
INSERT INTO employees
VALUES( 1, 'ABC', 10000); -- OK

INSERT INTO employees
VALUES( 1, 'PQR', 20000); -- NOT OK

INSERT INTO employees
VALUES( NULL, 'PQR', 20000); -- NOT NULL
```

- Composite Primary Key declaration at table level

```
CREATE TABLE student
(
    roll INT,
    division VARCHAR(50),
    name VARCHAR(50),
    marks FLOAT,
    CONSTRAINT PK_STUDENT_ROLL_DIVISION PRIMARY KEY( roll, division )
);
```

```
INSERT INTO student
VALUES
( 1, 'A', 'AAA', 50 ),
( 2, 'A', 'BBB', 50 ),
( 3, 'B', 'CCC', 50 ),
( 4, 'B', 'DD', 50 );
```

```
INSERT INTO student
VALUES
( 1, 'A', 'EEE', 60 ); --Not OK
```

```
INSERT INTO student
VALUES
( 1, NULL, 'EEE', 60 ); --Not Ok
```

```
INSERT INTO student
VALUES
( NULL, 'A', 'EEE', 60 ); --Not Ok
```

Surrogate Primary Key declaration at table level

```
CREATE TABLE orders
(
    id INT AUTO_INCREMENT,
    status VARCHAR( 50 ),
    CONSTRAINT PK_ORDERS_ID PRIMARY KEY( id)
);
```

```
INSERT INTO orders
(status)
VALUES('Pending');
```

```
INSERT INTO orders
(status)
VALUES('Delivered');
```

```
ALTER TABLE orders AUTO_INCREMENT = 100;
INSERT INTO orders
(status)
VALUES('Pending');
```

- Drop primary key

```
ALTER TABLE employees
DROP PRIMARY KEY;
```

- SET primary key

```
ALTER TABLE employees
ADD CONSTRAINT PK_EMPLOYEES_EMPID PRIMARY KEY( empid );
```

FOREIGN KEY Constraint

- If parent-child relationship exist between the table then we need to use foreign key.
- column in child table which referes to column in parent table is called foreign key.
- A table, which contains, foreign key, is called child table. A table which contains candidate key is called parent table.

```
CREATE TABLE dept
(
    dno INT,
    dname VARCHAR(50),
    loc VARCHAR( 50 ),
    CONSTRAINT PK_DEPT_DNO PRIMARY KEY(dno)
);

INSERT INTO dept
VALUES
( 10, 'DEV', 'PUNE'),
( 20, 'QA', 'MUMBAI');
```

```
CREATE TABLE emp
(
    eno INT,
    ename VARCHAR( 50 ),
    sal FLOAT,
    dno INT,
    -- FOREIGN KEY dno REFERENCES dept(dno)
    CONSTRAINT FK_EMP_DEPT_DNO FOREIGN KEY ( dno)  REFERENCES dept( dno )
);

INSERT INTO emp
VALUES
( 1, 'Rahul', 10000, 10),
( 2, 'Amit', 15000, 10),
( 3, 'Sandeep', 5000, 20);
```

```
INSERT INTO emp
VALUES
(4,'Ketan',7000,30);
```

```
INSERT INTO emp
VALUES
(4,'Ketan',7000,NULL);
```

- Remove foreign key

```
ALTER TABLE emp
DROP FOREIGN KEY FK_EMP_DEPT_DNO;

DESC emp;

INSERT INTO emp
VALUES
(4,'Ketan',7000,30);
```

- Set foreign key

```
ALTER TABLE emp
ADD CONSTRAINT FK_EMP_DEPT_DNO FOREIGN KEY ( dno)  REFERENCES dept( dno );
```

```
DELETE FROM dept
WHERE dno=20; -- NOT OK
```

```
DELETE FROM emp
WHERE eno=3; --OK
DELETE FROM dept
WHERE dno=20; -- NOT OK
```

- In case of foreign key, if we want to delete record from parent table then it is necessary to delete all records associated with primary key from child table.

```
ALTER TABLE emp
DROP FOREIGN KEY FK_EMP_DEPT_DNO;

ALTER TABLE emp
ADD CONSTRAINT FK_EMP_DEPT_DNO FOREIGN KEY ( dno)  REFERENCES dept( dno )
ON UPDATE CASCADE ;
```

```
UPDATE dept
SET dno=1
WHERE dname='DEV';

SELECT * FROM DEPT;
SELECT * FROM EMP;
```

```
ALTER TABLE emp
DROP FOREIGN KEY FK_EMP_DEPT_DNO;

ALTER TABLE emp
ADD CONSTRAINT FK_EMP_DEPT_DNO FOREIGN KEY ( dno)  REFERENCES dept( dno )
ON DELETE CASCADE ;
```

```
DELETE FROM dept
WHERE dno=1;
SELECT * FROM DEPT;
SELECT * FROM EMP;
```

```
ALTER TABLE emp
DROP FOREIGN KEY FK_EMP_DEPT_DNO;

ALTER TABLE emp
ADD CONSTRAINT FK_EMP_DEPT_DNO FOREIGN KEY ( dno)  REFERENCES dept( dno )
ON DELETE CASCADE ON UPDATE CASCADE ;
```

- foreign_key_checks is system variable that we can use to enable or disable foreign key.

```
SHOW VARIABLES;

SHOW VARIABLES LIKE 'f%';

SELECT @@foreign_key_checks from dual;

INSERT INTO emp
VALUES
( 1, 'Rahul', 10000, 10),
( 2, 'Amit', 15000, 10),
( 3, 'Sandeep', 5000, 20); -- NOt OK
```

```
SET @@foreign_key_checks = 0;
SELECT @@foreign_key_checks from dual;
```

```
INSERT INTO emp
VALUES
( 1, 'Rahul', 10000, 10),
( 2, 'Amit', 15000, 10),
( 3, 'Sandeep', 5000, 20);

SET @@foreign_key_checks = 1;
SELECT @@foreign_key_checks from dual;

INSERT INTO dept
VALUES
( 10, 'DEV', 'PUNE'),
( 20, 'QA', 'MUMBAI');
```

Joins

- If we insert duplicate records inside table then it takes more space on HDD. To minimize this wastage we should use normalization.
- In case of normalization, we can create multiple tables and we can maintain relationship between them using constraints(PK , FK).
- If we want to get data from multiple tables then we should join.
- Types of joins
    1. Cross join
    2. Inner Join
    3. Outer Join
        1. Left Outer Join
        2. Right Outer Join
        3. Full Join
    4. Self Join

Cross Join

- In case cross join, a table which is having maximum records is called "driving table" and another table is called "driven" table.
- If we want to combine/join each record of driving table with all the records of driven table then we should use cross join.
- It is also called as cartesion join.

```
emplist : collection of emp objects
deptlist : collection of dept objects
for( emp : emplist )
{
    for( dept : deptlist )
    {
        print( emp & dept );
    }
}
```

```
SELECT * FROM
EMP e CROSS JOIN DEPT d;
```

```
SELECT e.ename, d.dname FROM
EMP e CROSS JOIN DEPT d;
```

```
SELECT e.ename, d.dname FROM
DEPT d CROSS JOIN EMP e;
```

```
SELECT e.ename, d.dname FROM
EMP e, DEPT d;
```

- It is a join without condition, hence it faster than all other joins.

Inner Join

- If we want fetch matching records from both tables then we should use inner join

```
for( e : emplist )
{
    for( dept d : deptlist )
    {
        if( e.deptno == d.deptno )
            print( e & d );
    }
}
```

```
SELECT e.ename, d.dname
FROM emp e INNER JOIN dept d
ON e.deptno=d.deptno;
```

```
SELECT e.ename, d.dname
FROM emp e , dept d
WHERE e.deptno=d.deptno;
```

LEFT OUTER JOIN

- If we want to get all the records (matching & non matching) from left table and matching records from right table then we should use left outer join.

```
for( e : emplist )
{
    flag = 0;
    for( d : deptlist )
    {
        if( e.dno == d.dno )
        {
            flag = 1;
            print( e & d )
        }
    }
    if( flag == 0 )
    print( e & NULL );
}
```

```
SELECT e.ename, d.dname
FROM emp e LEFT OUTER JOIN dept d
ON e.deptno = d.deptno;
```

```
SELECT e.ename, d.dname
FROM emp e LEFT JOIN dept d
ON e.deptno = d.deptno;
```

RIGHT OUTER JOIN

- If we want to get all the records (matching & non matching) from right table and matching records from left table then we should use right outer join.

```
for(d  : deptlist )
{
    flag = 0;
    for( e : emplist )
    {
        if( d.dno == e.dno )
        {
            flag = 1;
            print( e, d );
        }
    }
    if( flag == 0 )
     print( NULL, d );
}
```

```
SELECT e.ename, d.dname
FROM dept d LEFT JOIN  emp e
ON d.deptno = e.deptno;
```

```
SELECT e.ename, d.dname
FROM emp e RIGHT JOIN dept d
ON e.deptno = d.deptno;
```

View

Temporary Table