Day 2 SQL Constraints, Primary key, Foreign key, SQL functions

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints 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 table.

It Prevents invalid data entry into the table.

Employee Number	First Name	Last Name	Date of Birth	Department Code	Department Name	Department Head
1001	Steave	Jakson	25-09-1985	SA001	Sales	Paul Colgan
1002	Kitty	Mathew	06-04-1998	ACC008	Accounts	Jerry Mathew
1003	Meena	Patel	11-05-1992	SA001	Sales	Paul Colgan
1004	Nancy	Samual	02-12-1996	ACC008	Accounts	Jerry Mathew
1005	Michael	Smith	28-03-1995	SA001	Sales	Paul Colgan
1006	James	Garcia	22-01-1994	SA002	Sales	David Smith
1007	Nancy	Samual	11-02-1996	ACC008	Accounts	Charles Williams

Constraints

1. not null
2. unique
3. primary key
4. foreign key
5. check

Note: check constraint will not be supported by MySQL.

Note: some constraints we can apply at the column level and some constraints we can apply at the table level.

Column Level: where we define the column

1. not null
2. unique
3. primary key

Table Level: after defining all the columns

1. foreign key
2. composite key (multi-column primary key)

not null

a null value is not allowed, that column will be mandatory.

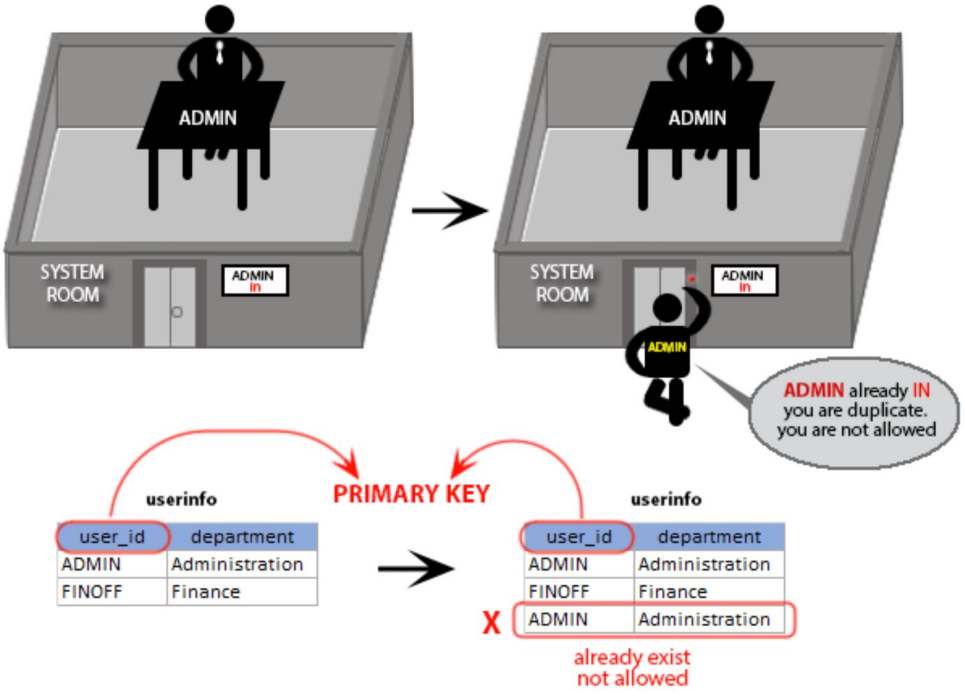
unique

- to that column duplicate values are not allowed.
- here we can insert null values multiple times.

Note: whenever we define a unique constraint on a column then automatically DB engine will create an index on that column.
(Searching based on the unique column is super fast)

primary key

- here also DB engine will create an index for that column.
- value can not be duplicated and value can not be null also.
- another difference between the Primary key and the unique key is inside one table we can have multiple unique constraints but inside one table we can have only one Primary key.
- if we want to apply the PK on multiple columns of a table then it will become a composite key.



Note: with the help of the PK column we can uniquely identify one record inside a table.

```
create table student
(
roll int primary key,
name varchar(12) not null,
address varchar(12) unique not null,
marks int
);
```

Composite key:

teacher (tname, subject, age, address, pincode)

```
create table teacher
(
tname varchar(12) not null,
subject varchar(12) not null,
age int not null,
address varchar(12),
pincode varchar(12),
primary key (tname, subject)
);
```

- here tname and subject will become a composite key, this combination can not be duplicated.

Foreign key:

- with the help of the Foreign key, we enforce referential integrity.
- with the help of a Foreign key, we establish the relationship between the two tables.
- The second table(child table) Foreign column must refer to the PK column of the parent/first table.
- Primary key-related Foreign key columns must belong to the same datatype but the column names can be different.

- FK can accept duplicate and null values also.

Note: with the help of FK we can establish the parent and child relationship between 2 tables.

```
create table dept
(
did int primary key,
dname varchar(12) not null,
location varchar(12)
);
```

```
create table emp
(
eid int primary key,
ename varchar(12),
salary int,
deptId int
);
```

- let's achieve referential integrity:

```
drop table emp;
```

```
create table emp
(
eid int primary key,
ename varchar(12),
salary int,
deptId int,
foreign key (deptId) references dept(did)
);
```

- the table which contains the FK column will be considered a child table.

Note: whenever we try to establish a relationship using FK then DB violates the following 2 rules:

1. insertion inside the child table. (we can not insert data that is not there inside the parent table)
2. deletion or updation in the parent table (even if we can not drop the parent table also.)

- so, in order to drop the parent table, we need to drop all the child tables then only we can drop the parent table?

```
delete from dept where did = 12; // error
```

```
update dept set did = 18 where did =12; // error
```

- to overcome this updation and deletion problem we should use :

ON DELETE CASCADE
or
ON DELETE SET NULL

similarly, we can use it for updates and also

ON UPDATE CASCADE
or
ON UPDATE SET NULL

- while creating the child table.

```
create table emp
(
eid int primary key,
ename varchar(12),
salary int,
deptId int,
foreign key (deptId) references dept(did) ON UPDATE CASCADE ON DELETE SET NULL
);
```

Adding constraints to the existing table:

```
create table a1(id int, name varchar(12));
```

```
alter table a1 modify id int primary key;
```

Adding FK to the existing table:

```
create table b1(bid int);
```

```
alter table b1 add foreign key (bid) references a1(id) on delete cascade;
```

Functions in MySQL

- It is used to solve a particular task.
- A sql function must return a value.

1. Predefined functions
2. User-defined functions (PL/SQL)

Predefined functions:

It is categorized into the following categories:

1. number functions
2. character functions
3. date functions
4. group functions or aggregate functions
5. special functions
6. number functions:

---

a. abs(): it returns the absolute number.

Example

```
select abs(-10) from dual;
```

Here dual is a sudo table, in mysql it is optional whereas in oralce db it is mandatory.

```
select 10+10 from dual;
or
select 10+10;
```

b. mod(m,n) : it retunrs the reminder of m/n;

Example

```
select mod(10,2) from dual;
```

c. round(m,n)
d. truncate(m,n)

Example:

```
select round(12.4248243,3) from dual;  // 12.425

e. ceil()
f. floor()
```

greatest() and least():

It will return the biggest and smallest number from the list of argument

Example:

```
select greatest(10,20,40,60,12,15) from dual;
```

Note: from a single column if we want to get the max and min values then we should use group functions like max() min();

Example

```
select max(marks) from student;  // Ok
select greatest(marks) from student;  // error
```

character functions:

a. upper()
b. lower()
c. length()
d. replace()
e. concat()
f. substr()

etc

```
select name, length(name) len from student;
select substr('ratan',3,2) from dual; // ta
```

date functions:

1. **sysdate():** it will return the current date and time.

```
select sysdate() from dual;
```

2. **date_format()**

```
select date_format(sysdate(), '%d-%m-%y');
```

3. adddate()

```
adddate(date, INTERVAL value unit);
```

DAY
HOUR
YEAR
MONTH
WEEK

```
select adddate(sysdate(), INTERVAL 10 DAY) from dual;
```

Group functions/ aggregate function:

- these functions operate over the several values of a single column and then results in in a single value.

1. max()
2. min()
3. sum()
4. avg()
5. count(*) // it will count the record, and consider the null value also
6. count(columnName) // it will not count the null value

```
select max(marks), min(marks), sum(marks), avg(marks) from student;
select count(marks) from student;  // null value will not be considers
select count(*) from student;
select count(DISTINCT marks) from student;
```

Group By clause:

- The main purpose of the group by clause is to group the records/ rows logically.
- This clause is mostly used with group functions only.
- It is used to divide similar data items into a set of logical groups.

short syn:

```
select col_name(s) from table group by  col_name(s);
```

long syn:

```
select col_name(s)
from
tablename(s)
[where condition] ---opt
```

```
group by col_name(s)
[having <cond>] ---opt
```

Example:

```
+------+---------+--------+--------+
| eid  | ename   | salary | deptId |
+------+---------+--------+--------+
| 1000 | ravi    |  80000 |     10 |
| 1001 | amit    |  82000 |     10 |
| 1002 | mukesh  |  84000 |     12 |
| 1003 | dinesh  |  78000 |     10 |
| 1004 | manoj   |  72000 |     12 |
| 1005 | chandan |  62000 |     14 |
| 1006 | rakesh  |  82000 |     11 |
+------+---------+--------+--------+
```

- the above data is called as detailed data and after performing the group by operation, we will get summarized data which is useful for the analysis.

```
select sum(salary) from emp; // it will calculate the salary of total emp.
```

- to calculate the sum of salaries dept wise.

```
select dept, sum(salary) from emp group by deptid;
+--------+-------------+
| deptid | sum(salary) |
+--------+-------------+
|     10 |      240000 |
|     11 |       82000 |
|     12 |      156000 |
|     14 |       62000 |
+--------+-------------+
```

```

```

```
select deptid, sum(salary), max(salary), min(salary), avg(salary) from emp group by deptid;
```

Rules of using group by

1. we should not use group functions, with normal columns without using group by clause, it will not give the correct result.

```
select deptid, sum(salary) from emp;
```

2. group functions we can not use inside the where clause.

Example:

```
select * from emp where avg(salary) > 80000;
ERROR 1111 (HY000): Invalid use of group function
```

3. other than the group function all the columns mentioned inside the select clause should be there after the group by clause otherwise (oracle DB will give an error and MySQL DB will not give the expected result.)

Example

```
select deptid, ename, sum(salary) from emp group by deptid, ename;
```

- here we have used deptid and ename other than group function sum(), so these both column name should be there inside the group by clause.
- if 2 emp with the same name works in the same deptid then their salaried will be grouped together.

Having clause:

- after group by clause, we are not allowed to use where condition, in place of where condition we should use having clause after the group by clause.
- having clause is always used with the group by clause.
- in where clause we can not use group functions, where as inside the having clause we can use group functions.

```
select deptid, min(salary), max(salary), avg(salary), sum(salary) from emp group by deptid having sum(salary) > 100000;
```

Using where clause and having clause also:

```
select deptid, sum(salary) from emp where deptid IN(10,12,14) group by deptid having sum(salary) > 100000;
```

Special functions:

1. IF() function:

- if we want to generate some data/column logically based on some decision then we can use IF() function.

syn:

IF(condition, value-IF_true, value_IF_false);

Example:

```
select IF(10 % 2 = 1, 'correct', 'incorecet') result from dual;
select *, if(salary > 60000 , 'High Sal', 'Low sal') desciption from emp;
select *, if(salary < 70000, salary, salary+5000) result from emp;
```

CASE() function:

- It is similar to the switch case in any programming language.
- The CASE statements goes through the multiple conditions and return a value when the condition is met, otherwise it will return the value defined inside the ELSE par**t.**

Syntax:

```
CASE
WHEN condition1 THEN result1
WHEN condition2 THEN result2
WHEN condition3 THEN result3
WHEN condition4 THEN result4
ELSE result;
END;
```

```
select *, (CASE WHEN salary <= 50000 THEN 'LOW' WHEN salary > 50000 AND salary < 70000 THEN 'medium' ELSE 'HIGH' END) description from emp;
```

```
create table product
(
pid int primary key,
pname varchar(12),
price int,
quantity int
);
```

```
mysql> insert into product values(1, 'pen' ,10, 100);
Query OK, 1 row affected (0.16 sec)

mysql> insert into product values(2, 'pencil' ,5, 200);
Query OK, 1 row affected (0.15 sec)

mysql> insert into product values(3, 'notebook' ,20, 10);
Query OK, 1 row affected (0.11 sec)

mysql> insert into product values(4, 'rubber' ,12, 0);
Query OK, 1 row affected (0.17 sec)
```

mysql> select * from product;

```
+-----+----------+-------+----------+
| pid | pname    | price | quantity |
+-----+----------+-------+----------+
|   1 | pen      |    10 |      100 |
|   2 | pencil   |     5 |      200 |
|   3 | notebook |    20 |       10 |
|   4 | rubber   |    12 |        0 |
+-----+----------+-------+----------+
```

4 rows in set (0.00 sec)

```
select *, (
CASE
WHEN quantity >= 100 THEN 'Sufficient Stock'
WHEN quantity < 100  AND quantity > 0 THEN 'Selling Fast'
ELSE 'sold out'
END
) description  from product;
```