Constrains

```
CREATE TABLE employee(
        eno int,
        name varchar(20),
        age int,
        department varchar(20),
        salary decimal(8,2)
);
```

Constraints

- Data integrity is a process that ensures only valid data should be stored in the database table.
- Constraints are used to prevent invalid data entry into our table.
- We have following types of constraints:
    - Not null
    - Unique
    - Primary key
    - Foreign key
    - Check
- All the above constraints are created in two level
    - column level
    - table level

Column Level

- In this method we are creating constraints on individual columns.
- Whenever we are creating a column at that time we can specify the constraints type.

  Syntax

  ```
  CREATE TABLE table_name(
          col1 datatype(size) constrainttype,
          ....
          coln datatype(size) constrainttype
  );
  ```

Table Level

- In this method we are creating constraints on group of column.

- In this method, first we should define all columns then at last we must specify constraints type along with group of columns.
- Syntax:

```
CREATE TABLE table_name(
        col1 datatype(size),
        ....
        coln datatype(size),
        constrainttype(col1,col2...,coln)
);
```

Different Ways of Creating Constraints

	Column Level	Table Level
Not null	Yes	No
Unique	Yes	Yes
Primary Key	Yes	Yes
Foreign Key	Yes	Yes
Check	Yes	Yes

NOT NULL

- It does not accept null values but it accepts duplicate values.
- It will support only column level syntax.

Column Level Syntax

```
CREATE TABLE employee(
        eno int not null,
        name varchar(20)
);
```

Insert Record

```
INSERT INTO employee values(1,'Ram');

INSERT INTO employee values(null,'Ram');
```

Error: Column 'eno' cannot be null

```
INSERT INTO employee(name) values('Ram');
```

Error: Field 'eno' doesn't have a default value

```
INSERT INTO employee values(1,'Swam');
```

Table Level Syntax

```
CREATE TABLE employee(
        eno int not null,
        name varchar(20),
        not null(eno,name)
);
```

Error: not null not support table level

Adding NOT NULL constraint to existing column

```
ALTER TABLE employee MODIFY name VARCHAR(20) NOT NULL;
```

Removing NOT NULL constraint from existing column

```
ALTER TABLE employee MODIFY name VARCHAR(20);
```

Note

- If existing column contains null values then we will not able to add not null constraint to that column.

UNIQUE

- It does not accept duplicate values but it will accept NULL values.
- It will support both column level and table level syntax.

Example

Column Level

```
CREATE TABLE employee(
        eno int unique,
        name varchar(20)
);
SELECT * FROM employee;

INSERT INTO employee values(1,'Ram');

INSERT INTO employee values(null,'Ram');

INSERT INTO employee(name) values('Ram');

INSERT INTO employee values(1,'Swam');
```

Table Level

```
CREATE TABLE employee(

            eno int,

            name varchar(20),

            unique(eno, name)

);

INSERT INTO employee values(1,'Ram');

INSERT INTO employee values(null,'Ram');

INSERT INTO employee(name) values('Ram');

INSERT INTO employee values(1,'Swam');

INSERT INTO employee values(1,'Swam');
```

```
Error:  Duplicate entry '1-Swam' for key 'employee.eno'
```

Assignment

```
CREATE TABLE employee(
            eno int,
            name varchar(20),
            unique(eno),
            unique(name)
);
SELECT * FROM employee;
INSERT INTO employee values(1,'Ram');
INSERT INTO employee values(null,'Ram');
INSERT INTO employee(name) values('Ram');
INSERT INTO employee values(1,'Swam');
INSERT INTO employee(name,eno) values('Ram',2);
```

Adding Unique constraint to existing column

```
ALTER TABLE employee ADD CONSTRAINT UNIQUE(name);
```

Removing Unique constraint from existing column

ALTER TABLE employee DROP CONSTRAINT name;

Note

- If existing column contains duplicate values then we will not able to add unique constraint to that column.
- If we are not giving name to unique constraint then default name will be the column name.

Check Constraint name

SELECT constraint_name FROM information_schema.KEY_COLUMN_USAGE WHERE TABLE_SCHEMA = avd AND TABLE_NAME = 'employee';

Giving User Defined Name to Unique Constraints

Syntax:

ALTER TABLE table_name ADD CONSTRAINT constraint_name constraint_type (column_name);

Example:

ALTER TABLE employee ADD CONSTRAINT name_unique UNIQUE(name);

Assigning Multiple Constraint to Same Column

```
CREATE TABLE employee(

        eno int not null unique,

  name varchar(20)

);
```

DESC employee;

SELECT * FROM employee;

INSERT INTO employee values(1,'Ram');

INSERT INTO employee values(1,'Raj');

INSERT INTO employee values(null,'Raj');

ALTER TABLE employee DROP CONSTRAINT eno;

ALTER TABLE employee MODIFY eno int;

PRIMARY KEY

- It does not accept duplicate and null values.
- It will uniquely identify a record in a table.
- There can be only one primary key in a table.

Example: Column Level

```
CREATE TABLE employee(
        eno int primary key,
        name varchar(20)
);
DESC employee;
SELECT * FROM employee;
INSERT INTO employee values(1,'Ram');
INSERT INTO employee values(null,'Ram');
INSERT INTO employee(name) values('Ram');
INSERT INTO employee values(1,'Swam');
```

Example: Table Level (Composite Primary Key)

- Composite primary key is the combination of columns as a single primary key.

```
CREATE TABLE employee(
        eno int,
        name varchar(20),
        primary key(eno, name)
);
DESC employee;
SELECT * FROM employee;
INSERT INTO employee values(1,'Ram');
INSERT INTO employee values(null,'Ram');
INSERT INTO employee values(2,null);
INSERT INTO employee values(1,'Swam');
INSERT INTO employee values(2,'Swam');
```

Adding Unique constraint to existing column

ALTER TABLE employee ADD CONSTRAINT primary key(eno);

<u>Removing Unique constraint from existing column</u>

ALTER TABLE employee DROP PRIMARY KEY;

ALTER TABLE employee MODIFY eno int;