Primary Key

- A primary key must contain unique values. If the primary key consists of multiple columns, the combination of values in these columns must be unique.
- A primary key column cannot have NULL values. Any attempt to insert or update NULL to primary key columns will result in an error. Note that MySQL implicitly adds a NOT NULL constraint to primary key columns.
- A table can have one and only one primary key
- Primary key column should be the integer e.g., INT, BIGINT.
- A primary key column often has the AUTO_INCREMENT attribute that automatically generates a sequential integer whenever you insert a new row into the table.

Define a PRIMARY KEY constraint in CREATE TABLE

There are 3 ways,

```
1.)CREATE TABLE users(
   user_id INT AUTO_INCREMENT PRIMARY KEY,
   username VARCHAR(40),
   password VARCHAR(255),
   email VARCHAR(255)
);
```

```
2.) CREATE TABLE roles(
   role_id INT AUTO_INCREMENT,
   role_name VARCHAR(50),
   PRIMARY KEY(role_id)
);
```

In case the primary key consists of multiple columns

```
3.) CREATE TABLE user_roles(

   user_id INT,

   role_id INT,

   PRIMARY KEY(user_id, role_id)

);
```

If a table, for some reasons, does not have a primary key, you can use the ALTER TABLE statement to add a primary key to the table

```
ALTER TABLE roles

ADD PRIMARY KEY(employee_id);
```

UNQIUE INDEX:

A UNIQUE index ensures that values in a column must be unique

MySQL allows NULL values in the UNIQUE index.

A table can have multiple UNIQUE indexes

Eg: Suppose that email and username of users in the users table must be unique. To enforce thes rules, you can define UNIQUE indexes for the email and username columns

```
ALTER TABLE users

ADD UNIQUE INDEX  email_unique (email ASC) ;
```

Foreign Key

- A foreign key is a column or group of columns in a table that links to a column or group of columns in another table
- The foreign key places constraints on data in the related tables, which allows MySQL to maintain referential integrity
- A table can have more than one foreign key where each foreign key references to a primary key of the different parent tables
- Once a foreign key constraint is in place, the foreign key columns from the child table must have the corresponding row in the parent key columns of the parent table or values in these foreign key column must be NULL
- (sample db)
- The 'customerNumber' column in the "orders" table links to the 'customerNumber' primary key column in the "customers" table.
- The customers table is called the parent table or referenced table, and the orders table is known as the child table or referencing table.
- Typically, the foreign key columns of the child table often refer to the primary key columns of the parent table.

Self-referencing foreign key

- Sometimes, the child and parent tables may refer to the same table. In this case, the foreign key references back to the primary key within the same table
- (sample db) employees table
- The 'reportTo' column is a foreign key that refers to the 'employeeNumber' column which is the primary key of the "employees" table.
- The foreign key on the column reportTo is known as a recursive or self-referencing foreign key.

FOREIGN KEY syntax (Follow below steps)

1. Specify the name of foreign key constraint that you want to create after the **CONSTRAINT** keyword. If you omit the constraint name, MySQL automatically generates a name for the foreign key constraint.

2. Specify a list of comma-separated foreign key columns after the **FOREIGN KEY** keywords. The foreign key name is also optional and is generated automatically if you skip it

3. Specify the **parent table** followed by a list of comma-separated columns to which the **foreign key columns** reference (**REFERENCES** keyword)

4. Specify how foreign key maintains the referential integrity between the child and parent tables by using the **ON DELETE** and **ON UPDATE** clauses. The reference_option determines action which MySQL will take when values in the parent key columns are deleted (ON DELETE) or updated (ON UPDATE).

MySQL has five reference options:

- **CASCADE**: if a row from the parent table is deleted or updated, the values of the matching rows in the child table automatically deleted or updated.
- **SET NULL**: if a row from the parent table is deleted or updated, the values of the foreign key column (or columns) in the child table are set to NULL.
- **RESTRICT**: if a row from the parent table has a matching row in the child table, MySQL rejects deleting or updating rows in the parent table.
- **NO ACTION:** is the same as RESTRICT.
- **SET DEFAULT**: is recognized by the MySQL parser. However, this action is rejected by both InnoDB and NDB tables.

If you don’t specify the ON DELETE and ON UPDATE clause, the default action is **RESTRICT**.

A.) Example scenario below (with no update or delete)

The 'categoryId' in the "products" table is the foreign key column that refers to the 'categoryId 'column in the "categories" table.

```
CREATE TABLE categories(

    categoryId INT AUTO_INCREMENT PRIMARY KEY,

    categoryName VARCHAR(100) NOT NULL

) ENGINE=INNODB;
```

```
CREATE TABLE products(

    productId INT AUTO_INCREMENT PRIMARY KEY,

    productName varchar(100) not null,

    categoryId INT,

    CONSTRAINT fk_category

    FOREIGN KEY (categoryId)

        REFERENCES categories(categoryId)

) ENGINE=INNODB;
```

Because we don't specify any ON UPDATE and ON DELETE clauses, the default action is RESTRICT for both update and delete operation

You can insert data into the products table only if the value of foreign key exists in the parent table or else it will throw error.

You can't update the existing data too.

B.) With CASACADE

```
CREATE TABLE products(

    productId INT AUTO_INCREMENT PRIMARY KEY,

    productName varchar(100) not null,

    categoryId INT NOT NULL,

    CONSTRAINT fk_category

    FOREIGN KEY (categoryId)

    REFERENCES categories(categoryId)

        ON UPDATE CASCADE

        ON DELETE CASCADE

) ENGINE=INNODB;
```

Automatic update and delete in child class if operation performed in parent class.

C.) SET NULL action

```
CREATE TABLE products(

    productId INT AUTO_INCREMENT PRIMARY KEY,

    productName varchar(100) not null,

    categoryId INT,

    CONSTRAINT fk_category

    FOREIGN KEY (categoryId)

        REFERENCES categories(categoryId)

        ON UPDATE SET NULL

        ON DELETE SET NULL

)ENGINE=INNODB;
```

If parent table is modified then the child table value will not be deleted/updated but instead the foreign key will be set as NULL.

Drop MySQL foreign key constraints

To drop a foreign key constraint, you use the ALTER TABLE statement:

```
ALTER TABLE table_name

DROP FOREIGN KEY constraint_name;
```

To obtain the generated constraint name of a table, you use the SHOW CREATE TABLE statement:

```
SHOW CREATE TABLE table_name;
```

UNIQUE CONSTRAINT

- To ensure values in a column or a group of columns are unique
- A UNIQUE constraint can be either a column constraint or a table constraint.
- you specify the name of the UNIQUE constraint after the CONSTRAINT keyword
- If you define a UNIQUE constraint without specifying a name, MySQL automatically generates a name for it

1. **For 1 column**

```
CREATE TABLE table_name(

    ...,

    column_name data_type UNIQUE,

    ...

);
```

2. **For multiple columns**

```
CREATE TABLE table_name(

   ...

   column_name1 column_definition,

   column_name2 column_definition,

   ...,

   CONSTRAINT constraintName

   UNIQUE(column_name1,column_name2)

);
```

You can drop an unique constraint anytime

```
ALTER TABLE table_name

DROP INDEX constraint_name;
```

add it to an existing column in a table

```
ALTER TABLE table_name

ADD CONSTRAINT constraint_name

UNIQUE (column_list);
```

NOT NULL Constraint

- The NOT NULL constraint is a column constraint that ensures values stored in a column are not NULL.
- IF a column has the PRIMARY KEY constraint, therefore, it implicitly includes a NOT NULL constraint.

```
column_name data_type NOT NULL;
```

It's a good practice to have the NOT NULL constraint in every column of a table unless you have a good reason not to do so

Generally, the NULL value makes your queries more complicated because you have to use functions such as ISNULL(), IFNULL(), and NULLIF() for handling NULL.

Steps to add a NOT NULL constraint to an existing column:-

1. Check the current values of the column if there is any NULL.

2. Update the NULL to non-NULL if NULLs exist.

3. Modify the column with a NOT NULL constraint.

Drop a NOT NULL constraint:-

To drop a NOT NULL constraint for a column, you use the ALTER TABLE..MODIFY statement:

```
ALTER TABLE table_name

MODIFY column_name column_definition;
```

Note that the column definition (column_definition) must restate the original column definition without the NOT NULLconstraint.