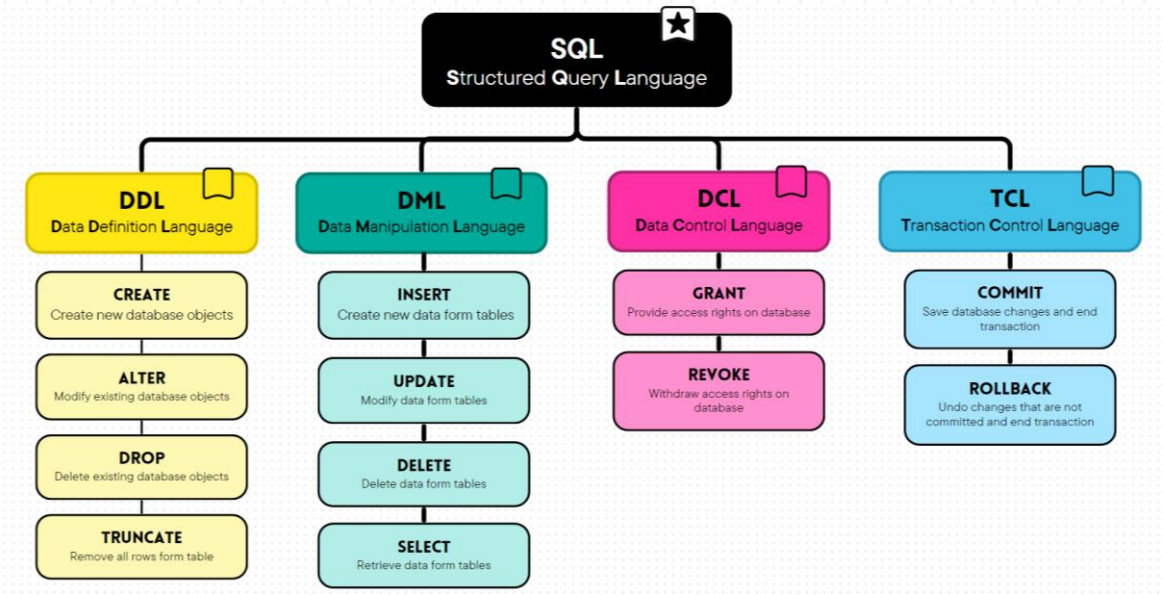
S31: SQL DDL COMMANDS

WHAT IS SQL?

- **SQL (Structured Query Language)** is a **programming language** used for managing and manipulating data in relational databases.
- It allows you to **insert**, **update**, **retrieve**, and **delete** data in a database.
- It is widely **used for data management** in many applications, websites, and businesses. In simple terms, SQL is used to communicate with and control databases.

TYPES OF SQL COMMANDS



DATA DEFINATION LANGUAGE (DDL)

- **"DDL"** stands for **Data Definition Language** in **SQL**.
- It's a subset of SQL statements used to define the structure of a database.

CREATE DATABASE Statement

- Used to create a new database.

Syntax	Example
CREATE { DATABASE } [IF NOT EXISTS] db_name; [create_option] ...	CREATE DATABASE college; or CREATE DATABASE IF NOT EXISTS college;

CREATE TABLE Statement

- Used to create a new table within a database.
- It allows you to define the structure of the table by specifying the columns it will contain, along with their data types and any constraints.

Syntax	Example
CREATE TABLE table_name (Column_name1 datatype [constraint], Column_name2 datatype [constraint], ...);	CREATE TABLE students(student_id INTEGER , name VARCHAR(255) , email VARCHAR(255),);

TRUNCATE TABLE Statement

- Used to quickly delete all rows from a table, effectively emptying it while maintaining the table structure.
- **NOTE:** This statement remove only table contents removed and structure of the table not removed.

Syntax	Example
TRUNCATE TABLE table_name;	TRUNCATE TABLE students;

DROP TABLE Statement

- Used to remove a specific table from the database.
- This statement removes both table data and table structure from database.
- **NOTE:** It permanently deletes the table and its contents.

Syntax	Example
DROP { TABLE } [IF EXISTS] table_name;	DROP TABLE students; or DROP TABLE IF EXISTS students;

DATA INTEGRITY

- Data integrity in databases refers to the accuracy, completeness, and consistency of the data stored in a database. It is a measure of the reliability and trustworthiness of the data and ensures that the data in a database is protected from errors, corruption, or unauthorized changes.
- There are various methods used to ensure data integrity, including:
  - **Constraints:**
    Constraints in databases are rules or conditions that must be met for data to be inserted, updated, or deleted in a database table. They are used to enforce the integrity of the data stored in a database and to prevent data from becoming inconsistent or corrupted.
  - **Transactions:**
    A sequence of database operations that are treated as a single unit of work.
  - **Normalization:**
    A design technique that minimizes data redundancy and ensures data consistency by organizing data into separate tables.

CONSTRAINTS IN MYSQL

- Constraints in databases are rules or conditions that must be met for data to be inserted, updated, or deleted in a database table.
- They are used to enforce the integrity of the data stored in a database and to prevent data from becoming inconsistent or corrupted.
- Following are list of constraints:

Constraints	Description

NOT NULL	The **NOT** logical operator is used to test for Boolean conditions & **NULL** is simply a place holder for data that does not exist.
	By default, a column can hold **NULL** values & the **NOT NULL** constraint ensure that a column in a table cannot have a **NULL** value, meaning that every row must contain a value for that column.
	Example:
	`CREATE TABLE users(`
	`user_id INTEGER NOT NULL,`
	`name VARCHAR(255) NOT NULL,`
	`password VARCHAR(255) NOT NULL,`
	`);`

UNIQUE

Ensures that all values in a column or combination of columns are unique across the entire table.
This constraint can be applied to a single column or a combination of columns (composite unique constraint).

Examples:

1st method to define **UNIQUE** constraint

```
CREATE TABLE users(
  user_id INTEGER NOT NULL,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL
);
```

2nd method to define **UNIQUE** constraint

```
CREATE TABLE users(
  user_id INTEGER NOT NULL,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  password VARCHAR(255) NOT NULL,
  CONSTRAINT unique_col UNIQUE (name, email)
);
```

PRIMARY KEY

A combination of the **NOT NULL** and **UNIQUE** constraints.
It uniquely identifies each record in a table and ensures that no duplicate values are allowed.

NOTE: In case, in future if you want to delete the primary key you can use 2nd method to delete without losing primary key columns.

Examples:

1st method to define **PRIMARY KEY** constraint

```
CREATE TABLE users(
  user_id INTEGER NOT NULL PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE
);
```

2nd method to define **PRIMARY KEY** constraint

```
CREATE TABLE users(
  user_id INTEGER NOT NULL,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE,
  CONSTRAINT users_pk PRIMARY KEY (user_id)
);
```

AUTO INCREMENT
(or IDENTITY)

Automatically generates a unique value for each new row inserted into the table.
It's often used in conjunction with the **PRIMARY KEY** constraint to create a unique identifier for each row.

Examples:

```
CREATE TABLE users(
  user_id INTEGER PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL
);
```

CHECK

Allows you to specify a condition that must be satisfied for the values in a column.
It's commonly used to enforce domain integrity by limiting the range of allowable values.

Example: Condition to checking the age of the user before inserting the data in column.

1st method to define **CHECK** constraint

```
CREATE TABLE users(
  user_id INTEGER PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  age INTEGER CHECK (age >= 18 AND age <= 30)
);
```

2nd method to define **CHECK** constraint

```
CREATE TABLE users(
  user_id INTEGER PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  age INTEGER ,
  CONSTRAINT age_check CHECK (age >= 18 AND
  age <= 30)
);
```

DEFAULT

Specifies a default value for a column.
If no value is explicitly provided when inserting a new row, the default value is used.

Example: In registration If user not insert the date in column, then default date is current date.

```
CREATE TABLE ticket(
  ticket_id INTEGER PRIMARY KEY ,
  name VARCHAR(255) NOT NULL,
  travel_date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

FOREIGN KEY (or REFERENCING KEY)	Establishes a relationship between two tables by referencing the primary key or a unique key of another table.
	It enforces referential integrity by ensuring that the values in the foreign key column(s) match values in the referenced column(s) of the related table.
	Syntax:
	`CREATE TABLE table_name(`
	`  col1 datatype,`
	`  col2 datatype, ...`
	`  CONSTRAINT [name_of_the_fk_column]  FOREIGN KEY [primary_key_column name]`
	`  REFERNECES (primary_key_table_name_with_primary_key_column_name)`
	`);`
	Example:
	`CREATE TABLE customers(`
	`  c_id INTEGER PRIMARY KEY AUTO_INCREMENT,`
	`  name VARCHAR(255) NOT NULL,`
	`  email VARCHAR(255) NOT NULL UNIQUE`
	`);`
	Orders table contains foreign key
	`CREATE TABLE orders(`
	`  order_id INTEGER PRIMARY KEY AUTO_INCREMENT,`
	`  c_id INTEGER NOT NULL,`
	`  name VARCHAR(255) NOT NULL,`
	`  order_date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,`
	`  CONSTRAINT orders_pk  FOREIGN KEY (c_id) REFERENCES customers(c_id)`
	`);`

REFERENTIAL ACTIONS OR REFERENTIAL INTEGRITY CONSTRAINTS

- When two tables are related via a foreign key, the actions taken on the referenced (or parent) table can affect the referencing (or child) table.
  OR
- When two tables are related through a foreign key constraint, referential actions define what happens in the related table when certain operations are performed on the primary key column of the referenced table.
- These constraints and actions are fundamental for maintaining data integrity and consistency in a database.
- There are typically four main types of referential actions:

1. RESTRICT

- Prevents the deletion of a row in the parent table if there are matching rows in the child table.
- **ON DELETE** and **ON UPDATE** are clauses used in foreign key constraints to specify what actions should be taken in the child table (the table containing the foreign key) when a corresponding record in the parent table (the table containing the primary key being referenced) is either deleted or updated.
- **Example:** To set a **RESTRICT** delete & update action

```
CREATE TABLE orders(
    order_id INTEGER PRIMARY KEY AUTO_INCREMENT
     c_id INTEGER NOT NULL,
    name VARCHAR  255   NOT NULL,
    order_date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT orders_pk  FOREIGN KEY (c_id) REFERENCES customers(c_id)
    ON DELETE RESTRICT,
    ON UPDATE RESTRICT
);
```

2. CASCADE

- If a user tries to delete the statement(s) which will affect the rows in the foreign key table, then those rows will be deleted when the primary key record is deleted.
- Similarly, if an update statement affects rows in the foreign key table, then those rows will be updated with the value from the primary key record after it has been updated.
- **Example:**

```
CREATE TABLE orders(
    order_id INTEGER PRIMARY KEY AUTO_INCREMENT
     c_id INTEGER NOT NULL,
    name VARCHAR(255) NOT NULL,
    order_date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT orders_pk  FOREIGN KEY (c_id) REFERENCES customers(c_id),
    ON DELETE CASCADE,
    ON UPDATE CASCADE
);
```

3. SET NULL

- Sets the foreign key column(s) in the child table to **NULL** when the corresponding row in the parent table is deleted or updated.
- **Example:** To set a **SET NULL** delete & update action

```
CREATE TABLE orders(
    order_id INTEGER PRIMARY KEY AUTO_INCREMENT
     c_id INTEGER NOT NULL,
    name VARCHAR(255) NOT NULL,
    order_date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT orders_pk  FOREIGN KEY (c_id) REFERENCES customers(c_id),
    ON DELETE SET NULL,
    ON UPDATE SET NULL
);
```

4. SET DEFAULT

- Similar to **SET NULL**, but instead of setting the foreign key column(s) to **NULL**, it sets them to their default values specified by the **DEFAULT** constraint
- **Example:** To set a **SET DEFAULT** delete & update action

```
CREATE TABLE orders(
    order_id INTEGER PRIMARY KEY AUTO_INCREMENT
     c_id INTEGER NOT NULL,
    name VARCHAR(255) NOT NULL,
    order_date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT orders_pk  FOREIGN KEY (c_id) REFERENCES customers(c_id),
    ON DELETE SET DEFAULT,
    ON UPDATE SET DEFAULT
);
```

ALTER TABLE Statement

- The **ALTER TABLE** statement is used to add, modify, or drop/delete columns in a table.
- The **ALTER TABLE** statement is also used to rename a table.
- **Syntax:**

```
ALTER TABLE table_name
ADD COLUMN column_name data_type [constraints]
```

```
ALTER TABLE table_name
ADD COLUMN column_name data_type [constraints]
[BEFORE | AFTER existing_column_name];
```

  - **table_name:** The name of the table to modify.
  - **new_column_name:** The name of the new column to add to the table.
  - **column_name datatype [constraints]:** The datatype and definition of the column (**NULL** or **NOT NULL**, etc).
  - **BEFORE | AFTER column_name** (Optional)
    - It tells **MySQL** where in the table to create the column. If this parameter is not specified, the new column will be added to the end of the table.
    - **BEFORE existing_column_name:** This adds the new column before the specified existing column.
    - **AFTER existing_column_name:** This adds the new column after the specified existing column.

Queries related to ADD COLUMNS:

- Here are some queries related to adding columns in **MySQL**:

customers table:

```
CREATE TABLE customers(
  c_id INTEGER NOT NULL,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255)
);
```

c_id	name	email
1	rohit	rohit@gmail.com

Adding 'password' column using ADD COLUMN:

```
ALTER TABLE customers
ADD COLUMN password VARCHAR(255) NOT NULL
```

c_id	name	email	password
1	rohit	rohit@gmail.com	

Add column at specific location using AFTER:

```
ALTER TABLE customers
ADD COLUMN surname VARCHAR(255) NOT NULL AFTER name
```

c_id	name	surname	email	password
1	rohit	sharma	rohit@gmail.com	

Add multiple columns in table:

```
ALTER TABLE customers
ADD COLUMN pan_number VARCHAR(255) NOT NULL AFTER name,
ADD COLUMN join_date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
```

c_id	name	pan_number	surname	email	password	join_date
1	rohit		sharma	rohit@gmail.com		2024-05-05 18:57:59

Queries related to DELETE COLUMNS:

- Here are some queries related to deleting columns in **MySQL**:

customers table:

c_id	name	pan_number	surname	email	password	join_date
1	rohit	ABCD12KA4	sharma	rohit@gmail.com	123!@#	2024-05-05 18:57:59

1.Deleting 'email' column using DROP COLUMN:

```
ALTER TABLE customers
DROP COLUMN email;
```

c_id	name	surname	pan_number	join_date
1	rohit	sharma	ABCD12KA4	2024-05-05 18:57:59

2.Delete multiple columns in table:

```
ALTER TABLE customers
DROP COLUMN password, DROP COLUMN join_date;
```

c_id	name	surname
1	rohit	sharma

Queries related to MODIFY COLUMNS:

- Here are some queries related to modifying columns in **MySQL** using the **MODIFY** clause:

customers table:

c_id	name	pan_number	surname	email	password	join_date
1	rohit	ABCD12KA4	sharma	rohit@gmail.com	123!@#	2024-05-05 18:57:59

1.Modify the columns in table:

```
ALTER TABLE customers MODIFY COLUMN pan_number INTEGER NOT NULL;
```

c_id	name	pan_number	surname	email	password	join_date
1	rohit	0	sharma	rohit@gmail.com	123!@#	2024-05-05 18:57:59

Editing & Deleting constraints:

- **Editing Constraints:**
  You cannot directly edit a constraint in **MySQL**. Instead, you need to drop the existing constraint and then add a new one with the modified definition.
- **Deleting Constraints:**
  To delete a constraint, you use the **DROP** clause followed by the constraint type and name. For example, to drop a constraint
- Queries related to deleting the constraints:

customers table:

c_id	name	pan_number	surname
1	rohit	ABCD12K4	sharma

1.Add new column with constraints in customers table:

```
ALTER TABLE customers ADD CONSTRAINT  check_customers_age INTEGER CHECK (age > 18);
```

c_id	name	pan_number	surname	check_customers_age
1	rohit	ABCD12K4	sharma	19

2.Delete(drop) the constraints in customers table:

```
ALTER TABLE customers DROP CONSTRAINT  check_customers_age;
```

c_id	name	pan_number	surname
1	rohit	ABCD12K4	sharma

3.After drop, add required constraint in customers table:

```
ALTER TABLE customers ADD CONSTRAINT check_customers_age INTEGER CHECK (age > 13);
```

c_id	name	pan_number	surname	check_customers_age
1	rohit	ABCD12K4	sharma	12