



# BIG DATA TOOLS FOR MANAGERS (N2MBA07)

## Unit -2 : Data Querying and Retrieval using SQL

---

# SQL- Structured Query Language

- SQL commands are instructions for the database. It is used to communicate with the database.
- SQL can perform various tasks like create a database/table, add data to tables, drop the table, modify the table, set permission for users.
- SQL commands are case insensitive, but table and column names are case sensitive.

# SQL- Structured Query Language

SQL statements are divided into two major categories.

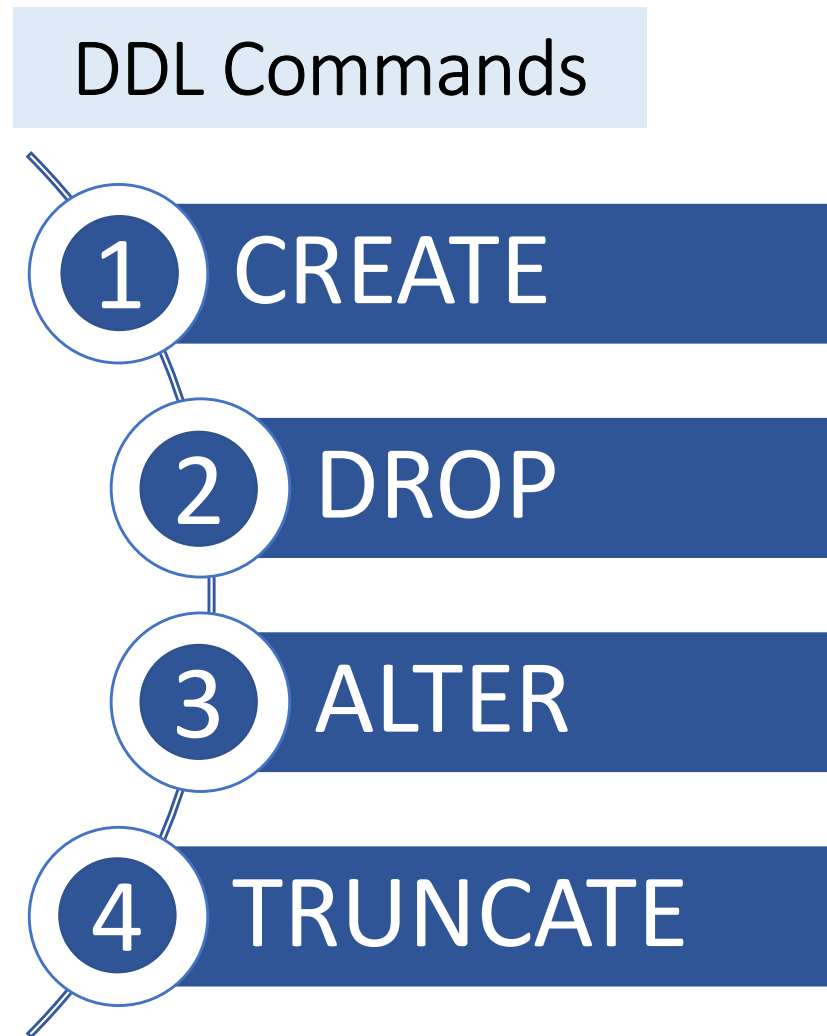
1. Data definition language (DDL)
2. Data manipulation language (DML)

# Data definition language (DDL)

- ✓ DDL statements are used to build and modify the structure of tables in the database.
- ✓ When you execute a DDL statement, it takes effect immediately.
- ✓ It is also known as data descriptive language.

# Data definition language (DDL)

- ✓ DDL statements are used to build and modify the structure of tables in the database.
- ✓ When you execute a DDL statement, it takes effect immediately.
- ✓ It is also known as data descriptive language.



# Data manipulation language (DML)

- ✓ Data Manipulation Language commands allow you to manage the data stored in the database.
- ✓ DML Command is used by the database user/ application programs to retrieve, add, remove or update the information in the database.

# Data manipulation language (DML)

- ✓ Data Manipulation Language commands allow you to manage the data stored in the database.
- ✓ DML Command is used by the database user/ application programs to retrieve, add, remove or update the information in the database.

## DML Commands

INSERT

SELECT

UPDATE

DELETE

# Data definition language (DDL)

## CREATE

Create command is a DDL command used to create a table or a database some objects like tables, Views, indexes, functions.

Syntax :

```
CREATE TABLE table_name (  
    col-name-1 data-type ,  
    col-name-2 data-type ,  
    col-name-3 data-type ,  
    ....  
);
```



# Data definition language (DDL)

## CREATE

```
CREATE TABLE table_name (  
    col-name-1    data-type ,  
    col-name-2    data-type ,  
    col-name-3    data-type ,  
    ....  
);
```

```
CREATE TABLE emp_details (  
    EMP_ID  TEXT,  
    EMP_NAME TEXT,  
    EMP_POST TEXT  
);
```

# Data definition language (DDL)

## DATA TYPE

The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

### Most common data types:

- String/Text
- Numeric
- Datetime

```
CREATE TABLE table_name (  
    col-name-1    data-type ,  
    col-name-2    data-type ,  
    col-name-3    data-type ,  
    ....  
);
```

# Data definition language (DDL)

DATA TYPE

STRING

## **CHAR(size)**

A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored

Example:

```
CREATE TABLE student(  
    usn CHAR(10)  
);
```

# Data definition language (DDL)

## DATA TYPE

## STRING TYPES

### CHAR(size)

A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length.

Example:

```
CREATE TABLE student(  
    usn CHAR(10)  
);
```

1	S	I	2	3	M	B	A	0	1
1	S	I	2	3	M	B	A	2	(space)
1	S	I	2	3	B	A	3	(space)	(space)

# Data definition language (DDL)

## DATA TYPE

## STRING

### **VARCHAR(size)**

A variable-length string between 1 and 255

Example:

```
CREATE TABLE student(  
    usn VARCHAR(10)  
);
```

1	S	I	2	3	M	B	A	0	1
1	S	I	2	3	M	B	A	2	
1	S	I	2	3	B	A	3		
1	S	I	2	3	M	B	A	0	4

# Data definition language (DDL)

## DATA TYPE

## STRING

---

### **TINYTEXT**

- TEXT column with a maximum length of 255 characters

### **MEDIUMTEXT**

- TEXT column with a maximum length of 16777215 characters

### **LONGTEXT**

- TEXT column with a maximum length of 4294967295 or 4 GB of characters

# Data definition language (DDL)

DATA TYPE

STRING

Example:

```
CREATE TABLE student (  
    USN          CHAR(10),  
    NAME         VARCHAR(50),  
    ADDRESS      TINYTEXT,  
    CITY         VARCHAR(20),  
    STATE        VARCHAR(20),  
    PINCODE      CHAR(5)  
);
```

# Data definition language (DDL)

## DATA TYPE

## NUMERIC

Data Type	Signed range(Number with Sign)	Unsigned range
TINYINT	-128 to 127	0 to 255
SMALLINT	-32768 to 32767	0 to 65535
MEDIUMINT	-8388608 to 8388607	0 to 16777215
INT	-2147483648 to 2147483647	0 to 4294967295
BIGINT	-9223372036854775808 to 9223372036854775807	0 to 18446744073709551615
FLOAT	Decimal precision can go to 24 places for a float type	
DOUBLE	Decimal precision can go to 53 places for a double	



# Data definition language (DDL)

DATA TYPE

DATETIME

Data Type	Maximum Size
DATE	Values range from '1000-01-01' to '9999-12-31'.
TIME	Values range from '-838:59:59' to '838:59:59'.
DATETIME	Values range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
TIMESTAMP	Values range from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.

# Data definition language (DDL)

## DATA TYPE

---

### String Type

CHAR

VARCHAR

TINYTEXT

MEDIUMTEXT

LONGTEXT

### Numeric Type

TINYINT

SMALLINT

MEDIUMINT

INT

BIGINT

FLOAT

DOUBLE

### Date Type

DATE

TIME

DATETIME

TIMESTAMP

# Data definition language (DDL)

## DATA TYPE

---

Example : String, Int, and DateTime

```
CREATE TABLE student_details (  
    USN          CHAR(10),  
    NAME         VARCHAR(50),  
    DOB          DATE,  
    AGE          TINYINT,  
    CGPA         FLOAT,  
    ADDRESS      MEDIUMTEXT  
);
```

# Data definition language (DDL)

## CREATE with 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.

Syntax :

```
CREATE TABLE table_name (  
    col-name-1 data-type constraint ,  
    col-name-2 data-type constraint ,  
    col-name-3 data-type constraint ,  
    ....  
);
```

# Data definition language (DDL)

## CREATE with Constraints

---

Some basic SQL constraints:

1. NOT NULL
2. UNIQUE
3. PRIMARY KEY
4. DEFAULT

# Data definition language (DDL)

## CREATE with Constraints

### NOT NULL Constraint

- By default, a column can hold NULL values
- The NOT NULL constraint enforces a column to NOT accept NULL values.

### Syntax :

```
CREATE TABLE student_details (  
    USN          CHAR(10)    NOT NULL,  
    NAME         VARCHAR(50) NOT NULL,  
    DOB          DATE,  
    AGE          TINYINT,  
    CGPA         FLOAT,  
    ADDRESS      MEDIUMTEXT  
);
```

# Data definition language (DDL)

## CREATE with Constraints

### UNIQUE Constraint

- The UNIQUE constraint ensures that all values in a column are different.

Syntax :

```
CREATE TABLE student_details (  
    USN          CHAR(10)    UNIQUE,  
    NAME         VARCHAR(50),  
    DOB          DATE,  
    AGE          TINYINT,  
    CGPA         FLOAT,  
    ADDRESS      MEDIUMTEXT  
);
```

# Data definition language (DDL)

## CREATE with Constraints

---

### PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).



# Data definition language (DDL)

## CREATE with Constraints

---

### PRIMARY KEY Constraint

Syntax :

```
CREATE TABLE student_details (  
    USN          CHAR(10) PRIMARY KEY,  
    NAME         VARCHAR(50),  
    DOB          DATE,  
    AGE          TINYINT,  
    CGPA         FLOAT,  
    ADDRESS      MEDIUMTEXT  
);
```

# Data definition language (DDL)

## CREATE with Constraints

---

### **DEFAULT Constraint**

- The DEFAULT constraint is used to set a default value for a column.
- The default value will be added to all new records, if no other value is specified.

# Data definition language (DDL)

## CREATE with Constraints

---

### DEFAULT Constraint

```
CREATE TABLE student_details (  
    USN          CHAR(10) UNIQUE,  
    NAME         VARCHAR(50) NOT NULL,  
    DOB          DATE,  
    AGE          TINYINT,  
    CGPA         FLOAT,  
    ADDRESS      MEDIUMTEXT DEFAULT 'TUMKUR,INDIA'  
);
```

# View Table Structure

## DESCRIBE

**DESCRIBE** or **DESC** command used to view the structure of the table.

Syntax :

```
DESCRIBE table_name ;  
OR  
DESC table_name ;
```

Example:

```
DESC student_details;
```

# Data definition language (DDL)

## TRUNCATE

---

- TRUNCATE TABLE statement is used to delete the data inside a table, but not the table.
- It permanently removes records/observation from the table.

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE student_details;
```

# Data definition language (DDL)

## DROP

---

- The DROP TABLE statement is used to drop an existing table in a database.
- It permanently removes objects from the database or MySQL server.

Syntax:

**DROP TABLE** table\_name;

Example:

**DROP TABLE** student\_details;

# Data definition language (DDL)

## ALTER

---

Alter command is used to alter the structure of the tables in the database.

For Example :

- ✓ To add a column to existing table.
- ✓ To rename any existing Column.
- ✓ Alter is also used to drop a column.
- ✓ To Change datatype of any Column or to modify its size

# Data definition language (DDL)

## Example:

---

Create employee table with following columns :  
emp\_id, ename, post, country

```
CREATE TABLE employee (  
    emp_id  text,  
    ename   text,  
    post    text,  
    country text  
);
```



# Data manipulation language (DML)

- ✓ Data Manipulation Language commands allow you to manage the data stored in the database.
- ✓ DML Command is used by the database user/ application programs to retrieve, add, remove or update the information in the database.

## DML Commands

INSERT

SELECT

UPDATE

DELETE

# Data manipulation language (DML)

## INSERT

---

INSERT INTO statement is used to insert new records in a table.

Syntax:

**INSERT INTO**

**table\_name**

**VALUES** ('value-1', 'value-2',.....);

# Data manipulation language (DML)

## INSERT

---

INSERT INTO statement is used to insert new records in a table.

Syntax:

**INSERT INTO**

**table\_name**

**VALUES** ('value-1', 'value-2',.....);

# Data manipulation language (DML)

## INSERT

---

INSERT INTO statement is used to insert new records in a table.

INSERT INTO for selected columns

Syntax:

**INSERT INTO**

**table\_name** (col-1, col-2,...)

**VALUES** ('value-1', 'value-2',.....);

# Data manipulation language (DML)

## INSERT

---

Example :

```
INSERT INTO  
    employee  
VALUES ( 'E1', 'John', 'Manager', 'USA');
```

```
INSERT INTO  
    employee  
VALUES ( 'E2', 'Nick', 'AVP', 'UK');
```

```
INSERT INTO  
    employee  
VALUES ( 'E3', 'John', 'VP', 'INDIA');
```

# Data manipulation language (DML)

## SELECT

---

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

Syntax:

**SELECT**

*column1, column2, column3,...*

**FROM** *table\_name*;

# Data manipulation language (DML)

SELECT

---

SELECT \* FROM table\_name;

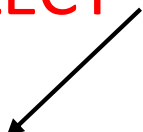
 (asterisk) represent all the columns from table

# Data manipulation language (DML)

SELECT

---

SELECT \* FROM table\_name;

 (asterisk) represent all the columns from table

SELECT \* FROM employee;

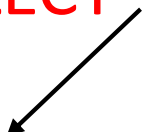


# Data manipulation language (DML)

## SELECT

---

**SELECT** \* **FROM** table\_name;

 (asterisk) represent all the columns from table

**SELECT** \* **FROM** employee;

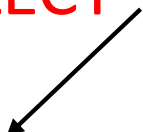
**SELECT** **ename** **FROM** employee;

# Data manipulation language (DML)

## SELECT

---

**SELECT** \* **FROM** table\_name;

 (asterisk) represent all the columns from table

**SELECT** \* **FROM** employee;

**SELECT** ename **FROM** employee;

**SELECT** ename, country **FROM** employee;

# Data manipulation language (DML)

## SELECT DISTINCT

---

The SELECT DISTINCT statement is used to return only distinct (different) values.

Syntax:

```
SELECT  
DISTINCT column1, column2...  
FROM table_name;
```

```
SELECT DISTINCT ename FROM employee;
```

```
SELECT DISTINCT ename, country FROM employee;
```

# Data manipulation language (DML)

## SELECT with WHERE

---

WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

Syntax:

**SELECT**

*column1, column2, column3,...*

**FROM** *table\_name*

**WHERE** *condition;*

# SQL Operators

## SQL WHERE

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. <b>Note:</b> In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

# SQL Operators

## SQL WHERE

---

Condition can be applied in multiple columns using AND OR operators.

- AND operator displays a record if all the conditions separated by AND are TRUE.
- OR operator displays a record if any of the conditions separated by OR is TRUE.

# SQL Operators

## SQL WHERE

---

Example:

```
SELECT * FROM employee WHERE emp_id = 'E1';
```

```
SELECT * FROM employee  
WHERE emp_id = 'E2' AND country = 'UK';
```

```
SELECT * FROM employee  
WHERE post = 'AVP' OR country = 'UK';
```

# GROUP BY

## Aggregation

---

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country."

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.



# GROUP BY

## Aggregation

---

### Syntax :

```
SELECT col-1, col-2, function(col-3)  
FROM table_name  
WHERE condition  
GROUP BY col-1, col-2;
```

### Example:

```
SELECT USN, Sections, max(CGPS)  
FROM student_details  
GROUP BY USN, Sections;
```

# GROUP BY with HAVING

## Aggregation

---

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

HAVING clause is equivalent to WHERE clause but HAVING used with only GROUP BY clause.

# GROUP BY

## Aggregation

---

### Syntax :

```
SELECT col-1, col-2, function(col-3)  
FROM table_name  
WHERE condition  
GROUP BY col-1, col-2  
HAVING condition;
```

### Example:

```
SELECT USN, Sections, max(CGPA)  
FROM student_details  
GROUP BY USN, Sections  
HAVING max(CGPA) > 7.5;
```

# ORDER BY

## Sorting

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

Syntax:

**SELECT**

*column1, column2, column3,...*

**FROM** *table\_name*

**ORDER BY** col-1, col-2...**ASC|DESC;**

# ORDER BY

## Sorting

### Example:

```
SELECT * FROM student_details  
ORDER BY USN ASC;
```

### Syntax:

```
SELECT  
    column1, column2,,...  
FROM table_name  
ORDER BY col-1, col-2...ASC|DESC;
```

# ORDER BY

## Sorting

### Example-1:

```
SELECT * FROM student_details  
ORDER BY USN ASC;
```

#### Syntax:

```
SELECT  
    column1, column2,,...  
FROM table_name  
ORDER BY col-1, col-2...ASC|DESC;
```

### Example-2:

```
SELECT * FROM student_details  
ORDER BY USN ASC, CGPA DESC ;
```

# Recap

## MySQL

---

### DDL

- **CREATE**
  - DATA TYPE
    1. String
    2. Numeric
    3. DateTime
  - CONSTRAINTS
    1. NOT NULL
    2. UNIQUE
    3. PRIMARY KEY
    4. DEFAULT
- **DROP**
- **TRUNCATE**
- **ALTER**

# Recap

## MySQL

---

### DML

- **INSERT**
- **SELECT**
  1. SELECT
  2. SELECT DISTINCT
  3. SELECT WITH WHERE
    - OPERATORS
  4. SELECT WITH GROUP BY
    - AGGREGATE FUNCTION
  5. SELECT WITH ORDER BY



# Data manipulation language (DML)

- ✓ Data Manipulation Language commands allow you to manage the data stored in the database.
- ✓ DML Command is used by the database user/ application programs to retrieve, add, remove or update the information in the database.

## DML Commands

INSERT

SELECT

UPDATE

DELETE

# Data manipulation language (DML)

## UPDATE

The UPDATE statement is used to modify the existing records in a table.

\*\*\*Update statement without condition will update values for all the records available in Table

Syntax:

```
UPDATE table_name
```

```
SET column1 = value1,  
    column2 = value2
```

```
WHERE condition;
```

\*\*Columns with new values

# Data manipulation language (DML)

## UPDATE

The UPDATE statement is used to modify the existing records in a table.

\*\*\*Update statement without condition will update values for all the records available in Table

Syntax:

```
UPDATE table_name  
SET column1 = value1,  
    column2 = value2  
WHERE condition;
```

\*\*WHERE condition tells which records to be updated.

# Data manipulation language (DML)

## UPDATE

---

Example: Create a STUDENTS table with below given data

ID	NAME	CITY	STATE	COUNTRY
1	AAA	PUNE	MAH	INDIA
2	BBB	MUMBAI	MAH	
3	CCC	TUMKUR	KAR	
4	DDD	BANGALORE		
5	EEE	MYSORE		

# Data manipulation language (DML)

## UPDATE

Example: Updates States for Bangalore & Mysore with KAR value.

Syntax:

```
UPDATE table_name  
SET column1 = value1,  
    column2 = value2  
WHERE condition;
```

Update Query:

```
UPDATE STUDENTS  
SET STATE = "KAR"  
WHERE CITY = "BANGALORE" OR CITY = "MYSORE";
```

# Data manipulation language (DML)

## UPDATE

---

Example: Updates Country as INDIA for all the records.

Syntax:

```
UPDATE table_name  
SET column1 = value1,  
    column2 = value2;
```

Update Query:

```
UPDATE STUDENTS  
SET COUNTRY = "INDIA";
```

# Data manipulation language (DML)

## DELETE

---

The DELETE statement is used to delete records from the table.

\*\*\*Delete statement without condition will delete all the records from the table.

### Syntax:

```
DELETE FROM table_name  
WHERE condition;
```

# Data manipulation language (DML)

## DELETE

The DELETE statement is used to delete records from the table.

\*\*\*Delete statement without condition will delete all the records from the table.

### Syntax:

```
DELETE FROM table_name  
WHERE condition;
```

\*\*WHERE condition tells which records to be deleted.



# Data manipulation language (DML)

## DELETE

---

Example: After UPDATE statement STUDENTS table will look like this.

ID	NAME	CITY	STATE	COUNTRY
1	AAA	PUNE	MAH	INDIA
2	BBB	MUMBAI	MAH	INDIA
3	CCC	TUMKUR	KAR	INDIA
4	DDD	BANGALORE	KAR	INDIA
5	EEE	MYSORE	KAR	INDIA

# Data manipulation language (DML)

## DELETE

---

Example: Delete the students from PUNE

Delete Query:

```
DELETE FROM STUDENTS  
WHERE CITY = "PUNE";
```

Syntax:

```
DELETE FROM table_name  
WHERE condition;
```

# Data manipulation language (DML)

## DELETE

---

Example: Delete all the records from STUDENTS table

Here are the option to delete all the records:

1. Using DELETE FROM table -----> DML Statement without WHERE condition

# Data manipulation language (DML)

## DELETE

---

Example: Delete all the records from STUDENTS table

Here are the option to delete all the records:

1. Using DELETE FROM table -----> DML Statement without WHERE condition
2. Using TRUNCATE table ---- DDL Statement

# Data manipulation language (DML)

## DELETE

---

Example: Delete the students from PUNE

Delete Query:

**DELETE FROM** STUDENTS;

OR

**TRUNCATE TABLE** STUDENTS;

Syntax:

**DELETE FROM** *table\_name*;

# MYSQL Joins

## Joins

---

A Join is an operation performed on database tables to fetch data from related tables, based on common fields/columns.

OR

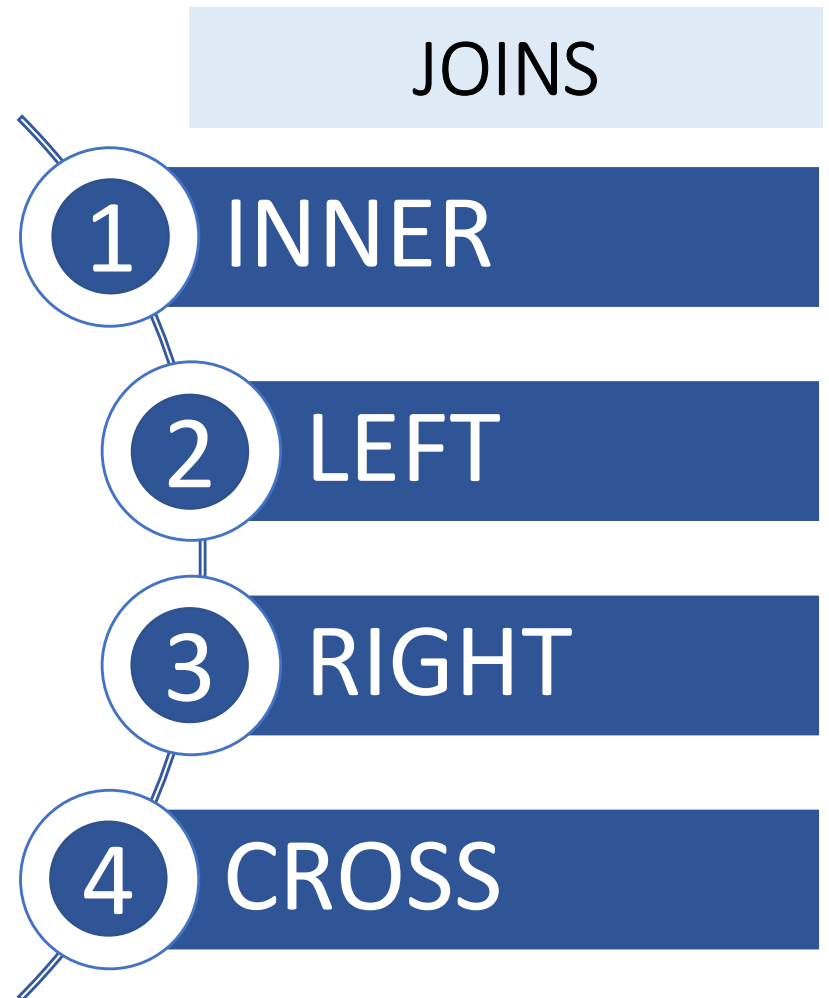
A Join is used to combine rows from two or more tables, based on a related column between them.

# MYSQL Joins

A Join is an operation performed on database tables to fetch data from related tables, based on common fields/columns.

OR

A Join is used to combine rows from two or more tables, based on a related column between them.



# MySQL Joins

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Download Data:

[https://raw.githubusercontent.com/sitmbadept/sitmbadept.github.io/main/BDTM/SQL/join\\_demo.sql](https://raw.githubusercontent.com/sitmbadept/sitmbadept.github.io/main/BDTM/SQL/join_demo.sql)

## **Orders Table**

OrderID	CustomerID	OrderDate
10308	2	2022-08-15
10309	1	2022-08-26
10310	2	2022-09-01

## **Customers Table**

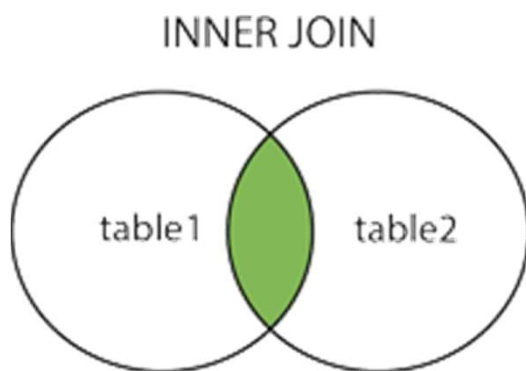
CustomerID	CustomerName	Country
1	John Todd	Germany
2	Dominic Dom	Mexico
3	Paul S	Mexico



# MYSQL Joins

## INNER

The INNER JOIN keyword selects records that have matching values in both tables.



### Syntax

**SELECT** *col-1, col-2....columns(s)*

**FROM** *table1*

**INNER JOIN** *table2*

**ON** *table1.column\_name = table2.column\_name;*

# INNER JOIN

Example: Selecting all the columns from both customer & orders the table

```
SELECT customers.* ,  
        orders.*
```

```
FROM customers
```

```
INNER JOIN orders
```

```
ON orders.CustomerID = customers.CustomerID;
```

# INNER JOIN

Example: Selecting specified columns from tables

```
SELECT orders.CustomerID,  
       orders.OrderID,  
       orders.OrderDate,  
       customers.CustomerName,  
       customers.Country  
  
FROM customers  
INNER JOIN orders  
ON orders.CustomerID = customers.CustomerID;
```

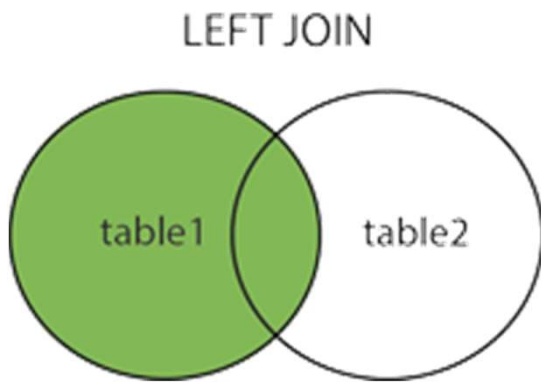
**Output:**

CustomerID	OrderID	OrderDate	customername	country
2	10308	2022-08-15	Dominic Dom	Mexico
1	10309	2022-08-26	John Todd	Germany
2	10310	2022-09-01	Dominic Dom	Mexico

# MYSQL Joins

## LEFT

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table



### Syntax

**SELECT** *col-1, col-2....columns(s)*

**FROM** *table1*

**LEFT JOIN** *table2*

**ON** *table1.column\_name = table2.column\_name;*

# LEFT JOIN

Example:

```
SELECT customers.* ,  
       orders.*  
FROM customers  
LEFT JOIN orders  
ON orders.CustomerID = customers.CustomerID;
```

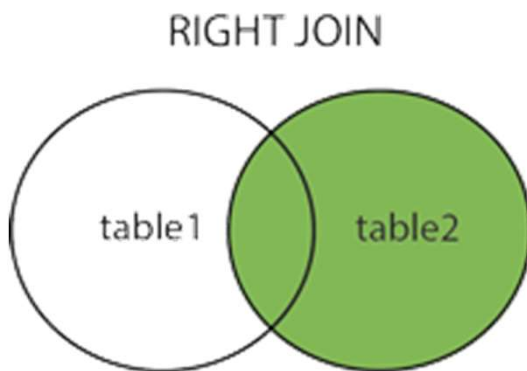
Output:

CustomerID	CustomerName	Country	OrderID	CustomerID	OrderDate
2	Dominic Dom	Mexico	10308	2	2022-08-15
1	John Todd	Germany	10309	1	2022-08-26
2	Dominic Dom	Mexico	10310	2	2022-09-01
3	Paul S	Mexico	NULL	NULL	NULL

# MYSQL Joins

## RIGHT

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1)



### Syntax

**SELECT** *col-1, col-2....columns(s)*

**FROM** *table1*

**RIGHT JOIN** *table2*

**ON** *table1.column\_name = table2.column\_name;*

# RIGHT JOIN

Example:

```
SELECT customers.* ,  
        orders.*
```

```
FROM customers
```

```
RIGHT JOIN orders
```

```
ON orders.CustomerID = customers.CustomerID;
```

Output:

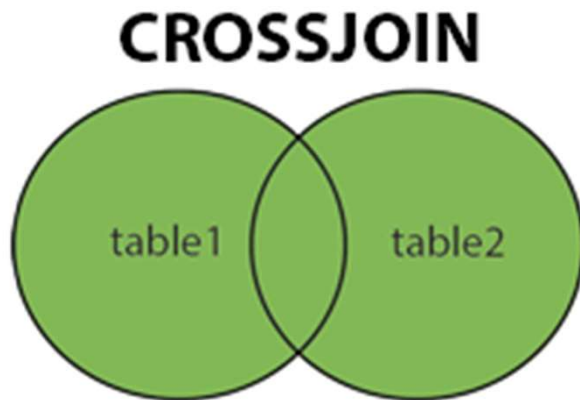
CustomerID	CustomerName	Country	OrderID	CustomerID	OrderDate
1	John Todd	Germany	10309	1	2022-08-26
2	Dominic Dom	Mexico	10308	2	2022-08-15
2	Dominic Dom	Mexico	10310	2	2022-09-01

# MYSQL Joins

## CROSS

---

The CROSS JOIN keyword returns all records from both tables (table1 and table2).



### Syntax

```
SELECT col-1, col-2....columns(s)  
FROM table1  
CROSS JOIN table2;
```



# CROSS JOIN

Example:

```
SELECT customers.* ,  
       orders.*  
FROM customers  
CROSS JOIN orders;
```

Output:

CustomerID	CustomerName	Country	OrderID	CustomerID	OrderDate
1	John Todd	Germany	10308	2	2022-08-15
2	Dominic Dom	Mexico	10308	2	2022-08-15
3	Paul S	Mexico	10308	2	2022-08-15
1	John Todd	Germany	10309	1	2022-08-26
2	Dominic Dom	Mexico	10309	1	2022-08-26
3	Paul S	Mexico	10309	1	2022-08-26
1	John Todd	Germany	10310	2	2022-09-01
2	Dominic Dom	Mexico	10310	2	2022-09-01
3	Paul S	Mexico	10310	2	2022-09-01

# MYSQL SET Operators

## UNION

---

The UNION operator is used to combine the result-set of two or more SELECT statements.

Important points :

- Every SELECT statement with UNION must have the same number of columns.
- Columns order must be same in all the SELECT statement.

# MYSQL SET Operators

## UNION

---

### UNION Syntax

```
SELECT col1, col2,..etc FROM table1  
UNION  
SELECT col1,col2,..etc FROM table2;
```

### UNION ALL Syntax

```
SELECT col1, col2,..etc FROM table1  
UNION ALL  
SELECT col1,col2,..etc FROM table2;
```

# MYSQL SET Operators

## UNION

---

### UNION Syntax

```
SELECT col1, col2,..etc FROM table1  
UNION  
SELECT col1,col2,..etc FROM table2;
```

### UNION ALL Syntax

```
SELECT col1, col2,..etc FROM table1  
UNION ALL  
SELECT col1,col2,..etc FROM table2;
```

*\*\*\*\*The UNION operator selects only distinct values by default. To allow duplicate values we need to use UNION ALL operators*

# UNION

Example:

```
SELECT customerNumber, customerName,  
city FROM customers_1
```

**UNION**

```
SELECT customerNumber, customerName,  
city FROM customers_2;
```

Output:

customerNumber	customerName	city
103	Atelier graphique	Nantes
112	Signal Gift Stores	Las Vegas
114	Australian Collectors, Co.	Melbourne
119	La Rochelle Gifts	Nantes
121	Baane Mini Imports	Stavern
124	Mini Gifts Distributors Ltd.	San Rafael
125	Havel & Zbyszek Co	Warszawa
128	Blauer See Auto, Co.	Frankfurt
129	Mini Wheels Co.	San Francisco
131	Land of Toys Inc.	NYC
141	Euro+ Shopping Channel	Madrid
144	Volvo Model Replicas, Co	Luleå

# UNION ALL

Example:

```
SELECT customerNumber, customerName,  
city FROM customers_1
```

**UNION ALL**

```
SELECT customerNumber, customerName,  
city FROM customers_2;
```

Output:

customerNumber	customerName	city
103	Atelier graphique	Nantes
112	Signal Gift Stores	Las Vegas
114	Australian Collectors, Co.	Melbourne
119	La Rochelle Gifts	Nantes
121	Baane Mini Imports	Stavern
124	Mini Gifts Distributors Ltd.	San Rafael
103	Atelier graphique	Nantes
112	Signal Gift Stores	Las Vegas
114	Australian Collectors, Co.	Melbourne
125	Havel & Zbyszek Co	Warszawa
128	Blauer See Auto, Co.	Frankfurt
129	Mini Wheels Co.	San Francisco
131	Land of Toys Inc.	NYC
141	Euro+ Shopping Channel	Madrid
144	Volvo Model Replicas, Co	Luleå

# MYSQL SET Operators

## INTERSECT

---

The INTERSECT operator returns the distinct and common elements in two sets or common records from two or more tables

Important points for INTERSECT operator:

- Every SELECT statement with INTERSECT must have the same number of columns.
- Columns order must be same in all the SELECT statement.

# MYSQL SET Operators

## INTERSECT

---

### Syntax

```
SELECT col1, col2,..etc FROM table1
INTERSECT
SELECT col1,col2,..etc FROM table2
INTERSECT
.....;
```



# INTERSECT

Example:

```
SELECT customerNumber, customerName,  
city FROM customers_1
```

**INTERSECT**

```
SELECT customerNumber, customerName,  
city FROM customers_2;
```

Output:

+ Options

customerNumber	customerName	city
103	Atelier graphique	Nantes
112	Signal Gift Stores	Las Vegas
114	Australian Collectors, Co.	Melbourne

# MYSQL Views

## VIEW

---

In SQL, a view is a virtual table, based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

Create Syntax:

```
CREATE VIEW view_name AS  
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

# MYSQL Views

## VIEW

---

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

### Create Syntax:

```
CREATE VIEW view_name AS  
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

### Drop Syntax:

```
DROP VIEW view_name;
```

# MYSQL Views

## VIEW

---

In SQL, a view is a virtual table based on the result-set of an SQL statement.

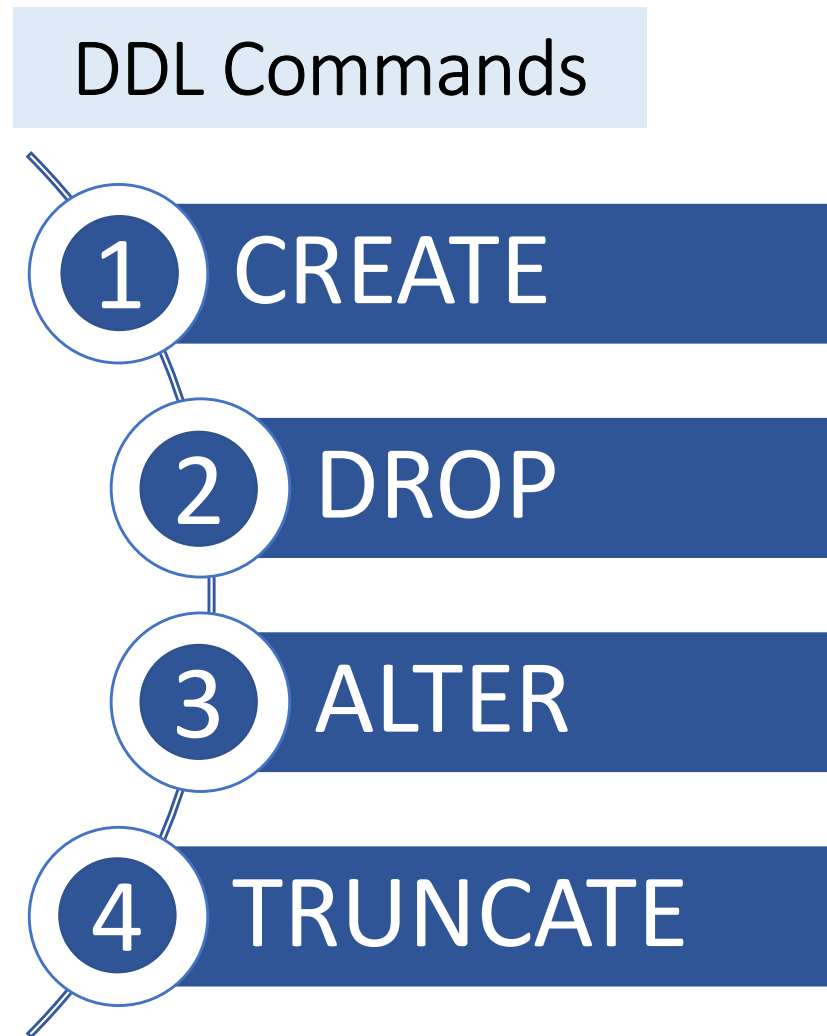
A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

Display Syntax:

```
SELECT * FROM view_name WHERE condition;
```

# Data definition language (DDL)

- ✓ DDL statements are used to build and modify the structure of tables in the database.
- ✓ When you execute a DDL statement, it takes effect immediately.
- ✓ It is also known as data descriptive language.



# Data definition language (DDL)

## ALTER

---

- The ALTER statement is used when to change the name of table or any table field. It is also used to add or delete an existing column in a table.
- The ALTER statement is always used with "ADD", "DROP" and "MODIFY" commands according to the situation.
- The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

# Data definition language (DDL)

## ALTER

---

Example: Create a STUDENTS table with below given data

ID	NAME	CITY	STATE	COUNTRY
1	AAA	PUNE	MAH	INDIA
2	BBB	MUMBAI	MAH	INDIA
3	CCC	TUMKUR	KAR	INDIA
4	DDD	BANGALORE	KAR	INDIA
5	EEE	MYSORE	KAR	INDIA

# Data definition language (DDL)

## ALTER

Adding Column:

ID	NAME	CITY	STATE	COUNTRY	MOBILE
1	AAA	PUNE	MAH	INDIA	
2	BBB	MUMBAI	MAH	INDIA	
3	CCC	TUMKUR	KAR	INDIA	
4	DDD	BANGALORE	KAR	INDIA	
5	EEE	MYSORE	KAR	INDIA	



# Data definition language (DDL)

## ALTER

---

Adding Column : ALTER with ADD statement allows to add new columns in existing table.

### Syntax:

**ALTER TABLE** table\_name

**ADD COLUMN** column\_name data-types constraints;

# Data definition language (DDL)

## ALTER

---

### Adding Column:

#### Example:

```
ALTER TABLE STUDENTS  
ADD COLUMN MOBILE VARCHAR(10);
```

#### Syntax:

```
ALTER TABLE table_name  
ADD COLUMN colum_name data-types constraints;
```

# Data definition language (DDL)

## ALTER

---

### Adding Multiple Columns:

#### Syntax:

**ALTER TABLE** table\_name

**ADD COLUMN** colum\_name data-types constraints,

**ADD COLUMN** colum\_name data-types constraints,

....

...;

# Data definition language (DDL)

## ALTER

### Adding Multiple Columns:

ID	NAME	CITY	STATE	COUNTRY	MOBILE	ALT_MOBILE	EMAIL
1	AAA	PUNE	MAH	INDIA			
2	BBB	MUMBAI	MAH	INDIA			
3	CCC	TUMKUR	KAR	INDIA			
4	DDD	BANGALORE	KAR	INDIA			
5	EEE	MYSORE	KAR	INDIA			

# Data definition language (DDL)

## ALTER

---

### Adding Multiple Columns:

#### Example:

```
ALTER TABLE STUDENTS
```

```
ADD COLUMN ALT_MOBILE VARCHAR(10),
```

```
ADD COLUMN EMAIL VARCHAR(20);
```

# Data definition language (DDL)

## ALTER

---

Modify Columns: ALTER with MODIFY statement allows modifying of the column definition.

### Syntax:

**ALTER TABLE** table\_name

**MODIFY COLUMN** column\_name data-types constraints;

# Data definition language (DDL)

## ALTER

### Modify Columns:

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	
NAME	varchar(10)	YES		NULL	
CITY	varchar(10)	YES		NULL	
STATE	varchar(10)	YES		NULL	
COUNTRY	varchar(10)	YES		NULL	
MOBILE	varchar(10)	YES		NULL	
ALT_MOBILE	varchar(10)	YES		NULL	
EMAIL	varchar(20)	YES		NULL	

Set Default  
Values For  
CITY & STATE  
(TUMKUR,  
KARNATAKA)

# Data definition language (DDL)

## ALTER

---

### Modify Columns:

#### Example:

```
ALTER TABLE STUDENTS  
MODIFY COLUMN CITY VARCHAR(20) DEFAULT "TUMKUR",  
MODIFY COLUMN STATE VARCHAR(20) DEFAULT "KARNATAKA";
```



# Data definition language (DDL)

## ALTER

### Modify Columns:

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	
NAME	varchar(10)	YES		NULL	
CITY	varchar(20)	YES		TUMKUR	
STATE	varchar(20)	YES		KARNATAKA	
COUNTRY	varchar(10)	YES		NULL	
MOBILE	varchar(10)	YES		NULL	
ALT_MOBILE	varchar(10)	YES		NULL	
EMAIL	varchar(20)	YES		NULL	

Set Default  
Values For  
CITY & STATE  
(TUMKUR,  
KARNATAKA)

# Data definition language (DDL)

## ALTER

---

Change Column: ALTER with CHANGE statement allows to rename the column name along with the data definition.

### Syntax:

**ALTER TABLE** table\_name

**CHANGE COLUMN** old\_column new\_column data-types;

# Data definition language (DDL)

## ALTER

---

Change Column :

Example:

```
ALTER TABLE STUDENTS
```

```
CHANGE COLUMN EMAIL PERSONAL_EMAIL VARCHAR(50);
```

# Data definition language (DDL)

## ALTER

Change Column :

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	
NAME	varchar(10)	YES		NULL	
CITY	varchar(20)	YES		TUMKUR	
STATE	varchar(20)	YES		KARNATAKA	
COUNTRY	varchar(10)	YES		NULL	
MOBILE	varchar(10)	YES		NULL	
ALT MOBILE	varchar(10)	YES		NULL	
PERSONAL_EMAIL	varchar(50)	YES		NULL	

# Data definition language (DDL)

## ALTER

Removing Column/s: ALTER with DROP statement allows to remove existing columns from the table.

ID	NAME	CITY	STATE	COUNTRY	MOBILE	ALT_MOBILE	EMAIL
1	AAA	PUNE	MAH	INDIA			
2	BBB	MUMBAI	MAH	INDIA			
3	CCC	TUMKUR	KAR	INDIA			
4	DDD	BANGALORE	KAR	INDIA			
5	EEE	MYSORE	KAR	INDIA			

# Data definition language (DDL)

## ALTER

---

### Removing Column/s:

#### Syntax:

**ALTER TABLE** table\_name

**DROP COLUMN** column\_name,

**DROP COLUMN** column\_name,

....

...;

# Data definition language (DDL)

## ALTER

---

Removing Column/s:

Example:

```
ALTER TABLE STUDENTS DROP COLUMN STATE;
```

Syntax:

```
ALTER TABLE table_name  
DROP COLUMN column_name,  
DROP COLUMN column_name,  
...  
...;
```

# Data definition language (DDL)

## ALTER

### Removing Column/s:

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	
NAME	varchar(10)	YES		NULL	
CITY	varchar(20)	YES		TUMKUR	
COUNTRY	varchar(10)	YES		NULL	
MOBILE	varchar(10)	YES		NULL	
ALT_MOBILE	varchar(10)	YES		NULL	
PERSONAL_EMAIL	varchar(50)	YES		NULL	



# Data definition language (DDL)

## ALTER

---

RENAME statement allows to rename the table name.

### Table

#### Syntax:

```
ALTER TABLE table_name RENAME TO new_table_name;
```

# Data definition language (DDL)

## ALTER

---

RENAME statement allows to rename the table name.

### Table

**Example:**

```
ALTER TABLE STUDENTS RENAME TO STUDENTS_DETAILS;
```

## UNIT-2 ASSIGNMENT

---

Course Link :

<https://cognitiveclass.ai/courses/learn-sql-relational-databases>

Assignment Submission: Complete the course from the given link and submit the course completion certificate here.

Submission Date : 16-Sep-2023

<https://docs.google.com/forms/d/e/1FAIpQLSewDToX1WaJ6DbS0B-HaKegQkHOBgkJNHdcmVXzZJlfRzxUhw/viewform?usp=sharing>