

Full Stack Web Development

# Basic Database Design and Development

- Intro to database
- Relational Database Management System
- Database variable and data types
- CRUD Database & table
- CRUD Data
- Clauses (where, distinct, order, grouping, having)
- Integrating MySQL with Node JS

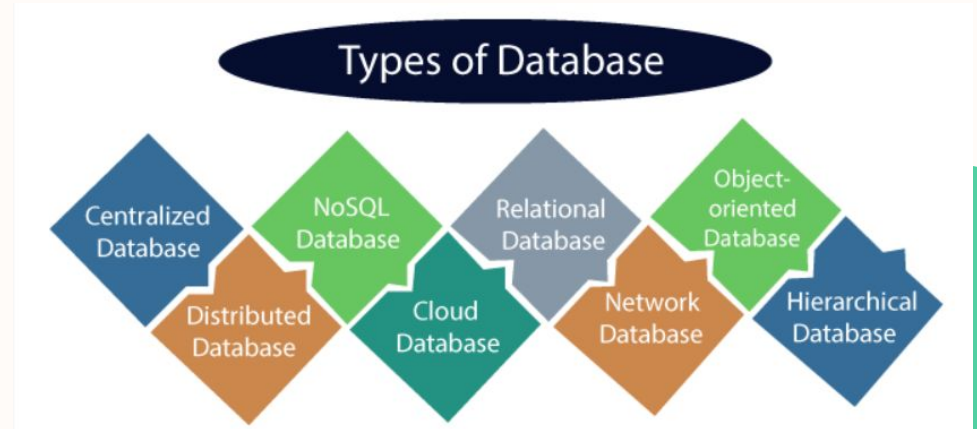
# Introduction to Database

- A database is an organized collection of data.
- The main purpose of database is to operate large amount of information by storing, retrieving and managing.
- There are many dynamic websites on the world wide web nowadays which are handled through databases. For example, an app to checks the availability of rooms in a hotel. It is an example of dynamic website that uses database.



# Common Types of Database

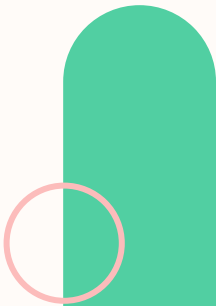
- Centralised database
- Distribution database
- Relational database
- NoSQL database
- Object-oriented database
- Etc ...



# Relational Database Management System (RDBMS)

- **How it works** - A relational database is the most commonly used database. It contains several tables, and each table has its primary key. Due to a collection of an organized set of tables, data can be accessed easily in RDBMS.
- **What is table with relation** - Everything in a relational database is stored in the form of relations. The RDBMS database uses tables to store data. A table is a collection of related data entries and contains rows and columns to store data. Each table represents some real-world objects such as person, place, or event about which information is collected. The organized collection of data into a relational table is known as the logical view of the database.

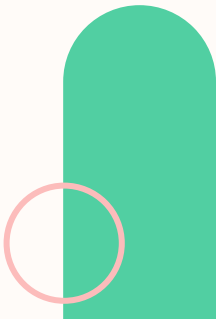
All modern database management systems like SQL, MS SQL Server, IBM DB2, ORACLE, My-SQL, and Microsoft Access are based on **RDBMS**.



# Relational Database Management System (RDBMS)

## Properties of a Relation:

- Each relation has a unique name by which it is identified in the database.
- Relation does not contain duplicate tuples.
- The tuples of a relation have no specific order.



# Relational Database Management System (RDBMS)

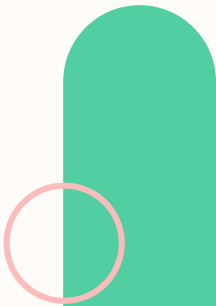
ID	Name	AGE	COURSE
1	Ajeet	24	B.Tech
2	aryan	20	C.A
3	Mahesh	21	BCA
4	Ratan	22	MCA
5	Vimal	26	BSC

A table is a collection of related data entries and contains rows and columns to store data,

- **Row or record** - A row of a table is also called a record or tuple. It contains the specific information of each entry in the table. It is a horizontal entity in the table.
- **Column or attribute** - A column is a vertical entity in the table which contains all information associated with a specific field in a table.

# Structured Query Language

- SQL (Structured Query Language) is used to communicate with a database. It's the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database.
- Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc.
- The standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

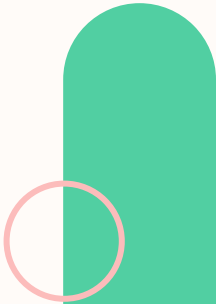




# Structured Query Language

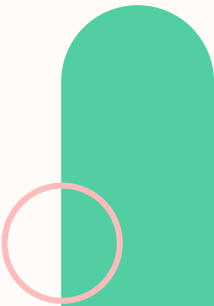
## SQL uses:

- **Data definition:** It is used to define the structure and organization of the stored data and relationships among the stored data items.
- **Data retrieval:** SQL can also be used for data retrieval.
- **Data manipulation:** If the user wants to add new data, remove data, or modifying in existing data then SQL provides this facility also.
- **Access control:** SQL can be used to restrict a user's ability to retrieve, add, and modify data, protecting stored data against unauthorized access.
- **Data sharing:** SQL is used to coordinate data sharing by concurrent users, ensuring that changes made by one user do not inadvertently wipe out changes made at nearly the same time by another user.



# MySQL

MySQL is the world's most popular open source database. With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice for web-based applications, used by high profile web properties including Facebook, Twitter, YouTube, Yahoo and many more.



# MySQL

There are two ways to access MySQL, through command line or using GUI.

Install MySQL workbench to easier maintain and manage your db through GUI

Go to: <https://www.mysql.com/products/workbench/>

```
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 8  
Server version: 8.0.21 MySQL Community Server - GPL  
  
Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
mysql>
```



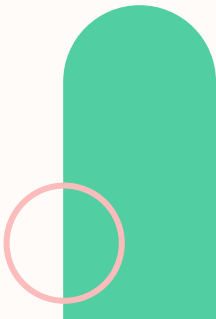
# MySQL Data Types

**The data type** of a column defines what value the column can hold: integer, character, money, date and time, binary, and so on.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. 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.

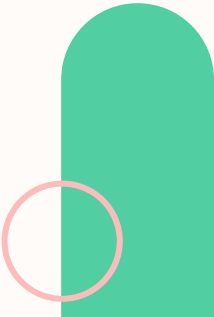
**In MySQL there are three main data types: string, numeric, and date time.**

.



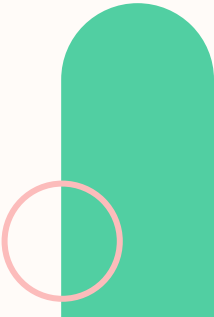
# MySQL - String Data Types

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The <i>size</i> parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The <i>size</i> parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The <i>size</i> parameter specifies the column length in bytes. Default is 1



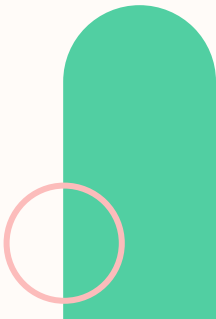
# MySQL - String Data Types

Data type	Description
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The <i>size</i> parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data



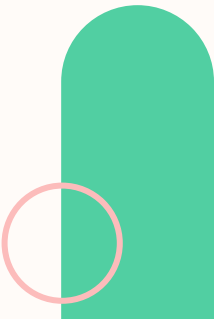
# MySQL - String Data Types

Data type	Description
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LOBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data



# MySQL - String Data Types

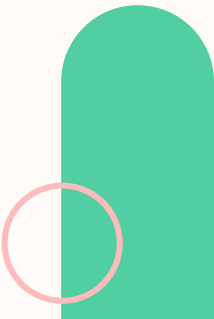
Data type	Description
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list





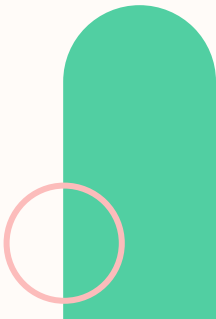
# MySQL - Numeric Data Types

Data type	Description
BIT( <i>size</i> )	A bit-value type. The number of bits per value is specified in <i>size</i> . The <i>size</i> parameter can hold a value from 1 to 64. The default value for <i>size</i> is 1.
TINYINT( <i>size</i> )	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The <i>size</i> parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL



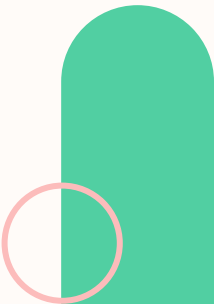
# MySQL - Numeric Data Types

Data type	Description
SMALLINT( <i>size</i> )	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The <i>size</i> parameter specifies the maximum display width (which is 255)
MEDIUMINT( <i>size</i> )	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The <i>size</i> parameter specifies the maximum display width (which is 255)
INT( <i>size</i> )	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The <i>size</i> parameter specifies the maximum display width (which is 255)



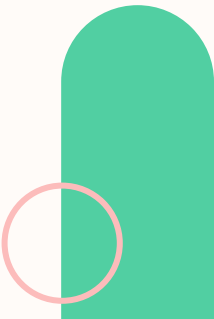
# MySQL - Numeric Data Types

Data type	Description
INTEGER( <i>size</i> )	Equal to INT( <i>size</i> )
BIGINT( <i>size</i> )	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The <i>size</i> parameter specifies the maximum display width (which is 255)
FLOAT( <i>size</i> , <i>d</i> )	A floating point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions



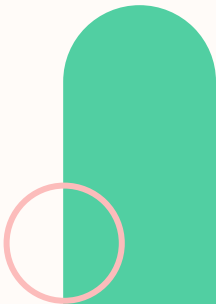
# MySQL - Numeric Data Types

Data type	Description
DOUBLE( <i>size</i> , <i>d</i> )	A normal-size floating point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter
DECIMAL( <i>size</i> , <i>d</i> )	An exact fixed-point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter. The maximum number for <i>size</i> is 65. The maximum number for <i>d</i> is 30. The default value for <i>size</i> is 10. The default value for <i>d</i> is 0.



# MySQL - Date and Time Data Types

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME( <i>fsp</i> )	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
TIMESTAMP( <i>fsp</i> )	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss.
TIME( <i>fsp</i> )	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format.



# CRUD Database

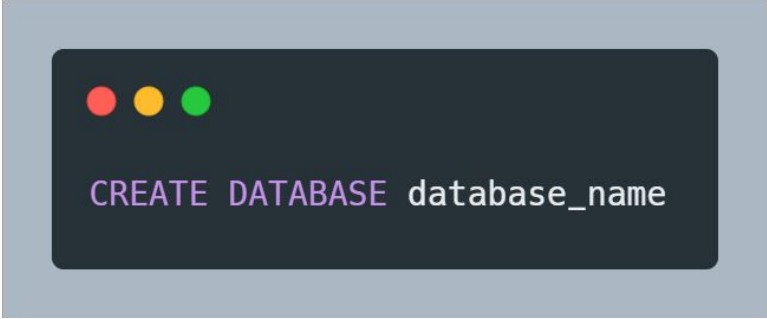
As we know that we can use MySQL to use Structured Query Language to store the data in the form of RDBMS. SQL is the most popular language for adding, accessing, and managing content in a database. It is most noted for its quick processing, proven reliability, ease, and flexibility of use. The application is used for a wide range of purposes, including data warehousing, e-commerce, and logging applications. The most common use for MySQL, however, is for the purpose of a web database.

MySQL provides a set of some basic but most essential operations that will help you to easily interact with the MySQL database and these operations are known as CRUD operations.



<b>C</b>	→	<b>Create</b>
<b>R</b>	→	<b>Read</b>
<b>U</b>	→	<b>Update</b>
<b>D</b>	→	<b>Delete</b>


# Create Database

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text 'CREATE DATABASE database\_name' is displayed in a light purple font.

```
CREATE DATABASE database_name
```


- **database\_name** - specify name of the database after the the CREATE DATABASE keywords. The database name must be unique within a MySQL server instance. If you attempt to create a database with a name that already exists, MySQL will issue an error.

# Show and Use Database



```
SHOW DATABASES;
```

It will display the current databases available on the server using the SHOW DATABASES statement



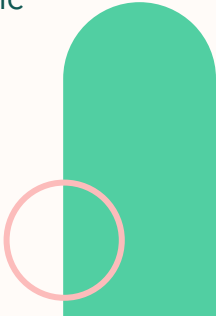
```
SHOW CREATE DATABASE database_name;
```

This command used to review the created database. MySQL returns the database name and the character set and collation of the database




```
USE database_name;
```

In order to use specific database, this command will pointing to use the specific database





# Drop or Delete Database



```
DROP DATABASE database_name;
```

- In this statement, you specify the name of the database which you want to delete after the DROP DATABASE keywords. Replace the database\_name with the selected database that would be remove.


# Create Table



```
CREATE TABLE table_name (column_name column_type constraints);
```

- **column\_name** – Name of the particular column with any space.
- **column\_type** – Datatype of the column. Datatype depends upon the data of the reference column. Datatype can be – char(), varchar(), int(), float(), etc.
- **constraints** – In order to give restrictions to particular column constraints are used. Constraints can be – not null, primary key, foreign key, etc. These are the keywords which give set of restriction to the particular column.

# Create Table - Example




```
CREATE TABLE student(name Varchar(30) NOT NULL, marks Integer);
```

Field	Type	Null	Default
name	varchar(30)	No	Null
marks	int(11)	YES	Null

# Read Table



```
SHOW TABLES;
```




```
SHOW FULL TABLES;
```




```
SHOW TABLES LIKE pattern;
```

- The SHOW TABLES command allows you to show if a table is a base table or a view. To include the table type in the result, you use the following form of the SHOW TABLES statement.
- the SHOW TABLES command provides you with an option that allows you to filter the returned tables using the LIKE operator or an expression in the WHERE clause

# Update Table



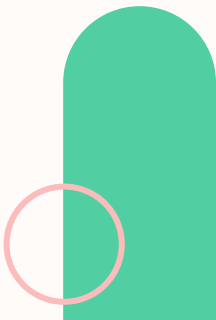
```
ALTER TABLE table_name  
ADD  
    new_column_name column_definition  
    [FIRST | AFTER column_name]
```



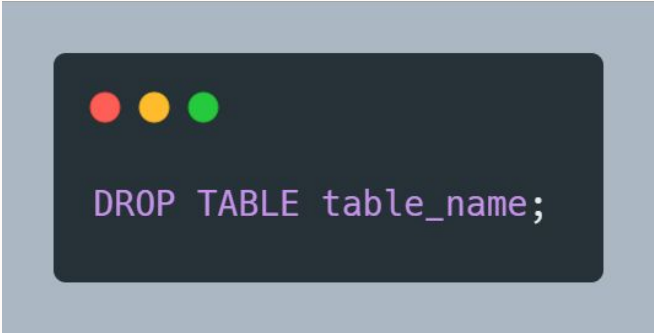
```
ALTER TABLE vehicles  
ADD model VARCHAR(100) NOT NULL;
```

The **ALTER TABLE** statement is used to add, delete, or modify columns in an existing table.

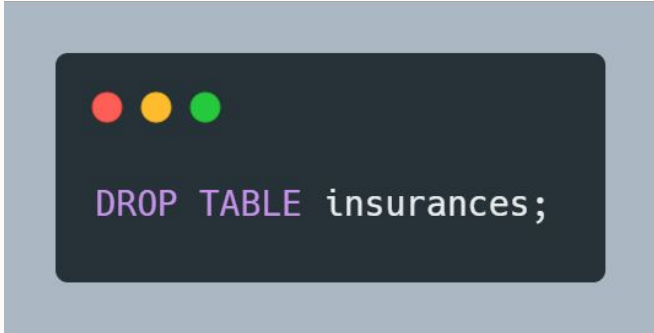
The **ALTER TABLE** statement is also used to add and drop various constraints on an existing table.



# Delete Table

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text `DROP TABLE table_name;` is displayed in a light purple font.

```
DROP TABLE table_name;
```

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text `DROP TABLE insurances;` is displayed in a light purple font.

```
DROP TABLE insurances;
```

To remove existing tables, you use the MySQL DROP TABLE statement. The DROP TABLE statement removes a table and its data permanently from the database. In MySQL, you can also remove multiple tables using a single DROP TABLE statement, each table is separated by a comma (,).

# Create Data



```
INSERT INTO table(c1,c2,...) VALUES (v1,v2,...);
```

In this syntax:

**First**, specify the table name and a list of comma-separated columns inside parentheses after the INSERT INTO clause.

**Then**, put a comma-separated list of values of the corresponding columns inside the parentheses following the VALUES keyword.

The number of columns and values must be the same. In addition, the positions of columns must be corresponding with the positions of their values.

# Create Data - Example

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It contains two lines of MySQL code: 'INSERT INTO tasks(title,priority)' and 'VALUES('Learn MySQL INSERT Statement',1);'.

```
INSERT INTO tasks(title,priority)
VALUES('Learn MySQL INSERT Statement',1);
```

The following statement inserts a new row into the tasks table. One row has been inserted into the tasks table successfully.



# Read Data



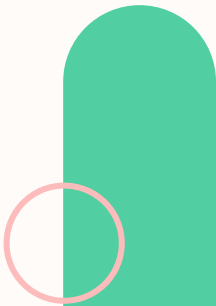
```
SELECT * FROM table_name;
```



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

There are two ways to read data from table:

- The left one is to select all field from table
- The second one is to select specific field from table by write down the field name



# Read Data - Select Example



```
SELECT * FROM Customers;
```



```
SELECT Country, Region FROM Customers;
```

There are two ways to read data from table:

- The left one is to select all field from table
- The second one is to select specific field from table by write down the field name

# Read Data

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The text inside the terminal is a SQL query: 

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

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

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

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

# Read Data - Select Distinct Example



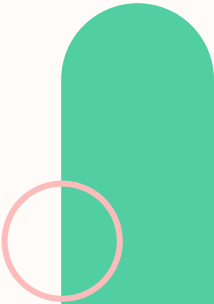
```
SELECT DISTINCT Country FROM Customers;
```



```
SELECT COUNT(DISTINCT Country), Region FROM Customers;
```


The following SQL statement on the left one, selects only the DISTINCT values from the "Country" column in the "Customers" table.

The following SQL statement on the right, lists the number of different (distinct) customer countries



# Update Data

- **table\_to\_update:** table\_to\_update is the name of the table whose column records we wish to update. Further, we need to specify all the columns and the expressions we want to assign to them in the comma-separated format after the SET keyword.
- **Column names:** column\_name1, column\_name2 are the names of the column and expression is the value we want to assign to them.
- **Expressions:** Expression can be any literal values, constraints, any manipulated values of expressions that involve operations like addition, subtraction, product, division, square, etc or variables another column values or expressions formed from another table's columns in case of update join statement.
- **Restrictions:** Whenever, we have to mention that only some of the records that satisfy certain conditions should be updated from the update statement, we will have to mention all the conditions that should be fulfilled using the where clause in SQL. These conditions are referred to as restrictions. Using the where clause is optional.



```
UPDATE table_to_update
SET column_name1 = expression,
    column_name2 = expression, ....
[WHERE any_restrictions];
```

# Update Data - Example

CustomerID	CustomerName	ContactName	Address	City
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.

```
UPDATE Customers SET ContactName = 'Alfred Schmidt', City= 'Frankfurt' WHERE CustomerID = 1;
```

CustomerID	CustomerName	ContactName	Address	City
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.

# Update Data - Multiple Example

It is the **WHERE** clause that determines how many records will be updated.

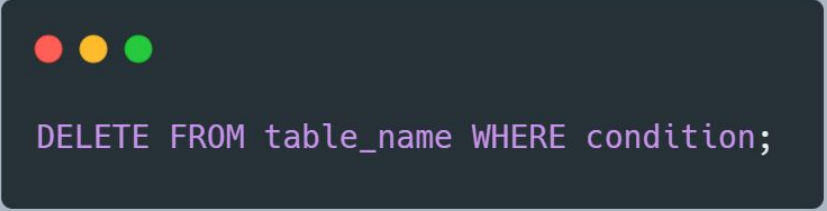
The following SQL statement will update the **ContactName** to "Juan" for all records where country is "Mexico":

```
UPDATE Customers SET ContactName='Juan' WHERE Country='Mexico';
```

CustomerID	CustomerName	ContactName	Address	City
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt
2	Ana Trujillo Emparedados y helados	Juan	Avda. de la Constitución 2222	México D.F.
3	Antonio Moreno Taquería	Juan	Mataderos 2312	México D.F.

# Delete Data


**SQL delete** is used to delete the records from the table. We can use the where clause with the delete statement to delete the selected records from the table. Suppose we have not used any condition in our query all the records from the table are deleted. The SQL delete is a DML query that allows the modification of database tables; it will delete the existing rows from the database tables.



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




# Delete Data - Example



```
DELETE FROM Customers WHERE id=1;
```

## Single Row Delete

Delete would be perform **only** row of data with id is 1 from table named as Customers

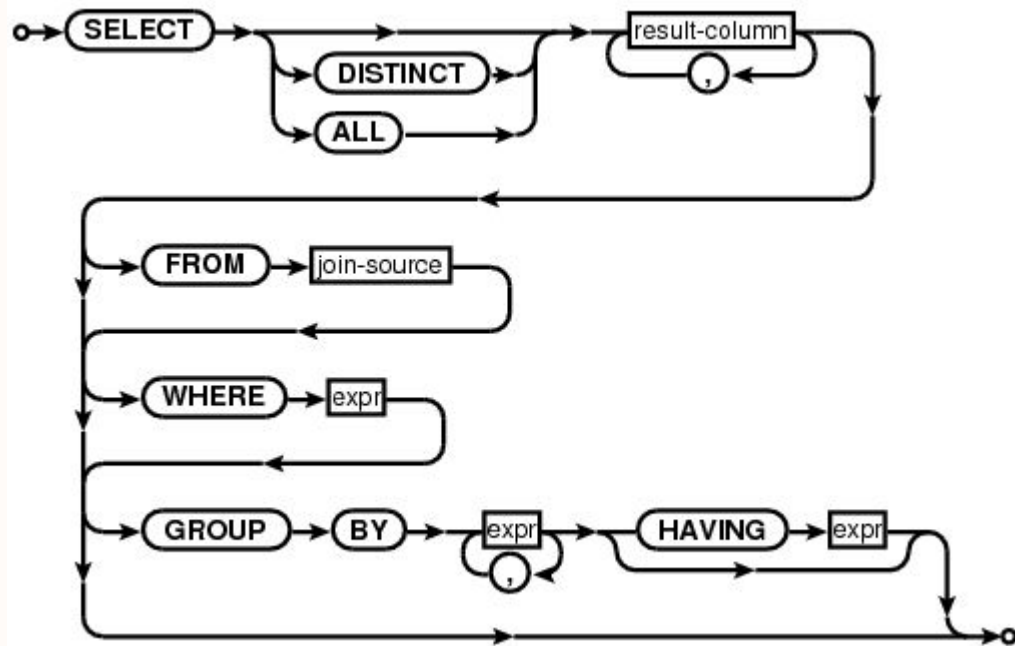


```
DELETE FROM Customers  
WHERE id='Alfreds Futterkiste';
```

## Multiple Row Delete

Delete would be perform on every row of data that CustomerName is 'Alfreds Futterkiste' from table named as Customers

# Clauses

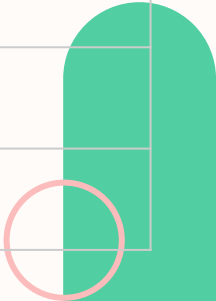


**MySQL system clauses** are keywords or statements to handle information. It helps to operate a group of the data and apply it to require conditions. The clauses apply conditions or select patterns to get information.


**MySQL clauses** are not used to insert new data. You retrieve data using several clauses. The table uses either single or multiple clauses.

# Type of Clauses

Clauses	Description
MySQL "WHERE" clause	This clause uses a condition to retrieve and delete data.
MySQL "FROM" clause	This clause works with a pattern to display the table and its value.
MySQL "ORDER BY" clause	It helps to show data in ascending or descending order.
MySQL "GROUP BY" clause	This clause displays information in a particular group.
MySQL "HAVING" clause	This clause applies after the "group by" clause.
MySQL "DISTINCT" clause	It removes the duplicate information of the table.
MySQL "LIMIT" clause	This clause helps to assign a limit to the table data.



# Clauses Syntax



```
SELECT column1, column2, ..., columnN
FROM table-name
WHERE condition
GROUP BY column1, column2, ..., columnN
HAVING column regex value (condition);
ORDER BY expression [ASC] | [DES];
```


This syntax uses some clauses such as "FROM," "WHERE," "GROUP BY," and "ORDER BY." You select entire table columns or specific columns from the table. The particular field selects the other clauses.

# Clauses - From



```
Select column1, column2 FROM table name;
```

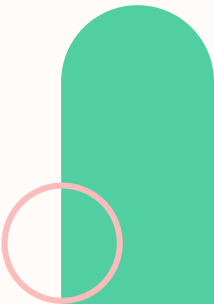
MySQL system stores multiple databases and tables. This table helps to display information in the application. Suppose an application requires a table from the system. The "FROM" operator gives a specific table. The "FROM" clause helps to get a specific table in the database.




```
SELECT chapter, marks, grade FROM mysql_tutorials;
```

# Clauses - Where

MySQL WHERE clause applies a condition on the table field. The table displays conditional information of the columns and rows. It supports the insert, updates, retrieve, and delete operations. This clause helps to display a particular position of the table. MySQL WHERE clause helps to apply other clauses on the table field.




# Clauses - Where to Retrieve



```
SELECT column_name1, column_name  
FROM table-name  
WHERE condition;
```


Execute the following query to retrieve data using the "WHERE" condition. Here, the condition applies to a "grade" column. The column is equal to an "A" grade

This clause comes with a condition of the table information. It requires a MySQL basic expression to display data. MySQL WHERE clause syntax shows beside.



```
SELECT * FROM mysql_tutorials  
WHERE grade = "A";
```

# Clauses - Where to Update



```
UPDATE table_name  
set column_name = 'new value'  
WHERE condition;
```

This clause comes with a condition of the table information. It requires a MySQL basic expression to display data. MySQL WHERE clause syntax shows beside.

Execute the besides query to get the updated chapter name. Here chapter column updates using the "WHERE" condition. The condition uses a marks chapter where value exists 40.



```
UPDATE mysql_tutorials  
set chapter = 'mysql introduction'  
WHERE marks = 40;
```



# Clauses - Group By to Retrieve Data


```
SELECT column1, column2, ..., column  
FROM table-name  
WHERE condition  
GROUP BY column1, column2, ..., columnN;
```

Execute the following query of the primary "Group BY" clause. The query uses a numerical column to create a group.

The "GROUP BY" clause collects multiple columns together and displays data. This clause supports the use of aggregation functions such as MIN, MAX, SUM, so on. It creates a group of a column and rows as per the requirement. This clause works with the "WHERE" condition. But, the "WHERE" condition shows optional.

```
SELECT chapter, marks  
FROM mysql_tutorials  
WHERE marks > 30 group by marks;
```


# Clauses - Having



```
SELECT column1, column2
FROM table name
GROUP BY column1, column2
HAVING condition;
```

Execute the following query to filter table data. Here, you search data using the "Having" condition with the "GROUP BY" clause. This example shows the use of a basic "having" query.

The "HAVING" clause comes with a "GROUP BY" clause. The "HAVING" clause is filtering data of specific rows. It provides a condition to the group of the row or aggregations. It works the same as a "where" clause after the "GROUP BY" clause.



```
SELECT * FROM mysql_tutorials
GROUP BY marks
HAVING marks = 40;
```

# Clauses - Order By

```
SELECT column1, column2
FROM table_name
ORDER BY
Column1 [ASC | DESC],
Column2 [ASC | DESC];
```

Execute the following query to filter table data. Here, you search data using the "Having" condition with the "GROUP BY" clause. This example shows the use of a basic "having" query.


The "ORDER BY" clause helps to display the table field per order.

ASC: This keyword displays column data in ascending order.

DESC: This keyword displays column data in descending order.

```
SELECT chapter, marks, grade, remark
FROM mysql_tutorials
GROUP BY marks
ORDER BY marks ASC, grade DESC;
```


# Clauses - Limit



```
SELECT column1, column2  
FROM table name  
LIMIT [OFFSET,] row count;
```

The "LIMIT" clause is displaying several outputs. This clause is showing information from the starting row in the table. The "LIMIT" clause is working with the "SELECT" statement. This clause can use the WHERE condition, but it is not necessary.

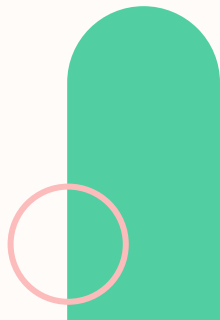
Execute the following query to limit the row result.



```
SELECT chapter, marks, remark  
FROM mysql_tutorials LIMIT 2
```


# Integrating MySQL with NodeJs

- MySQL package is a Node.JS driver for MySQL database. It's written in JavaScript, doesn't require compiling & 100% MIT licensed.
- More info: <https://www.npmjs.com/package/mysql2>
- On project dir, install MySQL package:
  - `$ npm install mysql2`



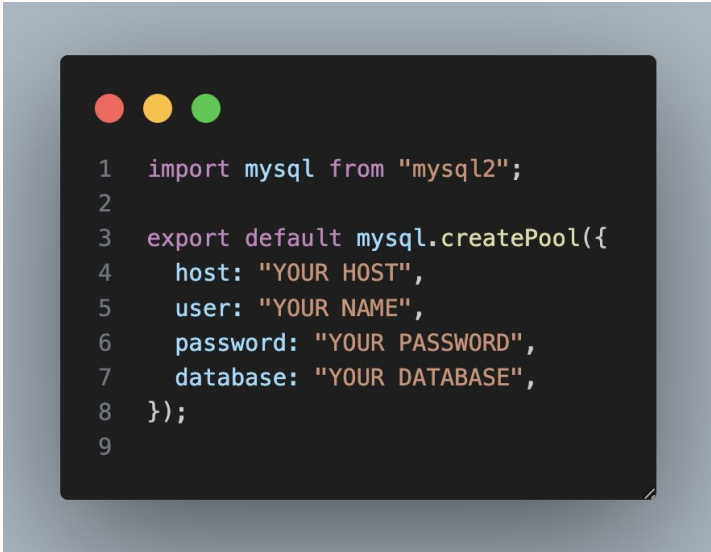
# Integrating MySQL with NodeJs

Initiate new project using:



```
npm init -y  
npm i mysql2
```

Create index.js and define your mysql and connection  
in the projects.



```
1  import mysql from "mysql2";  
2  
3  export default mysql.createPool({  
4    host: "YOUR HOST",  
5    user: "YOUR NAME",  
6    password: "YOUR PASSWORD",  
7    database: "YOUR DATABASE",  
8  });  
9
```

# Integrating MySQL with Node Js

After define your db connection,

Use db.connect method to check your connection.

Write down check connection at index.ts.

And run your script.

```
1 import express, { Request, Response, Application } from "express";
2 import db from "../config/db";
3 const PORT = 2323;
4 const app: Application = express();
5
6 app.use(express.json());
7
8 app.get("/", (req: Request, res: Response) => {
9   res.status(200).send("<h1>TS API</h1>");
10 });
11
12 // #check DB Connection
13 db.getConnection((err, connection) => {
14   if (err) {
15     return console.log(err);
16   }
17   console.log("Success Connection", connection.threadId);
18 });
19 app.listen(PORT, () => console.log("Running TS API", PORT));
20
```

# Try to get data from MySQL

After test connection success, write down route and middleware for get data from database.

```
1  app.get("/products", (req: Request, res: Response) => {
2    // #define product model. note : you can create model in other file, and import to middleware
3    interface Product {
4      id: number;
5      name: string;
6      description: string;
7      stock: number;
8      price: number;
9    }
10   db.query("Select * from product", (err: QueryError, result: Product[]) => {
11     if (err) {
12       return res.status(500).send(err);
13     }
14
15     return res.status(200).send(result);
16   });
17 });
```



# Thank You!

