UNIVERSITY OF MISKOLC
FACULTY OF MECHANICAL ENGINEERING
AND INFORMATICS

Java programming

GEIAL31A-B2a

Java Database Connectivity (JDBC)

with MySQL basics

Tamás Tompa, PhD
assistant professor
Department of Information Technology

Based on materials prepared by Miklós Szűcs

Miskolc, 2026

What is MySQL?

- MySQL is the most popular and a free **Open Source**
- **Relational Database Management System (RDBMS)**
  - allows users to create, manage, and interact with relational databases
  - in an RDBMS, data is stored in **tables** (rows and columns) where **relationships** between tables are **defined by keys** (like primary and foreign keys)
- An RDBMS **system stores the data in the form of tables** that might be related to each other

What is MySQL?

- MySQL **uses Structured Query Language (SQL)** to store, manage and retrieve data, and control the accessibility to the data
- It is one of the best RDBMS being used for developing web-based software applications
- MySQL is written in C and C++
- Developed by Michael Widenius & David Axmark beginning in 1994

Example

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	Hyderabad	4500.00
7	Muffy	24	Indore	10000.00

```
SELECT * FROM CUSTOMERS WHERE AGE = 25;
```

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
4	Chaitali	25	Mumbai	6500.00

What is database?

- **Database is used to store a collection of data** (which can either be structured or unstructured)
  - each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds

- Nowadays, we use **relational database management systems** (**RDBMS**) to store and manage huge volume of data

What is RDBMS?

- A **Relational DataBase Management System (RDBMS)**
  - enables you to implement a database with tables, columns and indexes
  - guarantees the Referential Integrity between rows of various tables.
  - updates the indexes automatically
  - interprets an SQL query and combines information from various tables
  - Elements:
    - tables
    - column
    - row
    - redundancy
    - primary Key
    - foreign Key
    - etc.

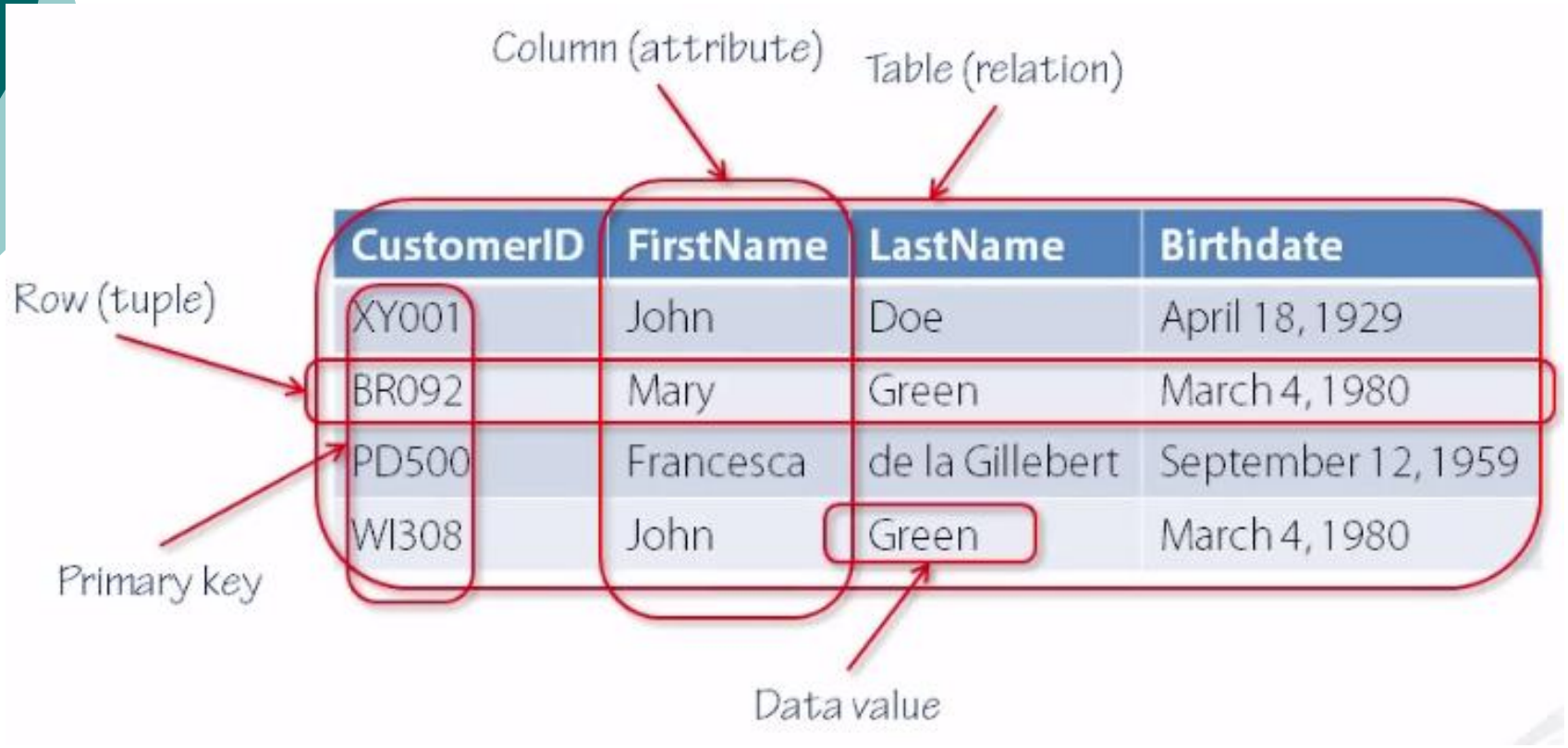
RDBMS Terminology

- **Database** – A database is a collection of tables, with related data
- **Table** – A table is a matrix with data. A table in a database looks like a simple spreadsheet
- **Column** – One column (data element) contains data of one and the same kind, for example the column postcode
- **Row** – A row (= tuple, entry or record) is a group of related data, for example the data of one subscription
- **Redundancy** – Storing data twice, redundantly to make the system faster

RDBMS Terminology

- **Primary Key** – A primary key is unique. A key value can not occur twice in one table. With a key, you can only find one row
- **Foreign Key** – A foreign key is the linking pin between two tables
  - "parent/child" relationship
  - can be make a specific column in the "child" table a foreign key that references a specific column in the "parent" table
- **Compound Key** – A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique
- **Index** – An index in a database resembles an index at the back of a book
- **Referential Integrity** – Referential Integrity makes sure that a foreign key value always points to an existing row

RDBMS Terminology



CustomerID	FirstName	LastName	Birthdate
XY001	John	Doe	April 18, 1929
BR092	Mary	Green	March 4, 1980
PD500	Francesca	de la Gillebert	September 12, 1959
WI308	John	Green	March 4, 1980

Primary key and Foreign key

A row can only be added or updated in the **orders** table if the value in **orders.user_id** matches an existing user ID in the **users** table.

This type of **database rule** is called a **foreign key constraint**.

RDBMS visualization

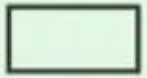
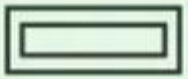
- **Entity Relationship (ER) Diagram** can be used the graphical representation of relationships between tables

- Flowchart that illustrates how "entities" such as people, objects or concepts relate to each other within a system
  - tables are represented as entities, and relationships between them are shown with connecting lines that indicate foreign keys and constraints, which help in understanding how tables relate to each other in the database

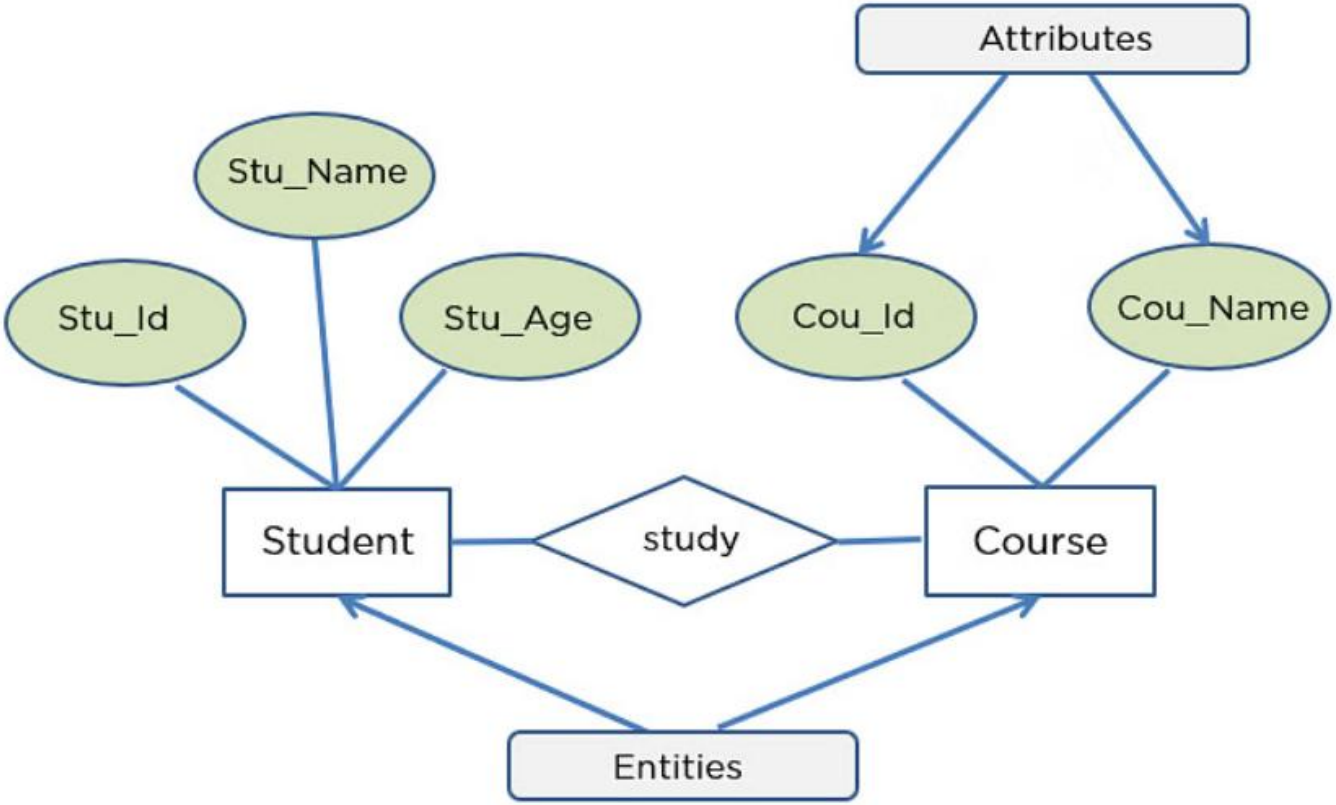
- Elements:
  - entity
  - attribute
  - relationship
  - primary Key (PK)
  - foreign Key (FK)

RDBMS visualization

- ER diagram elements

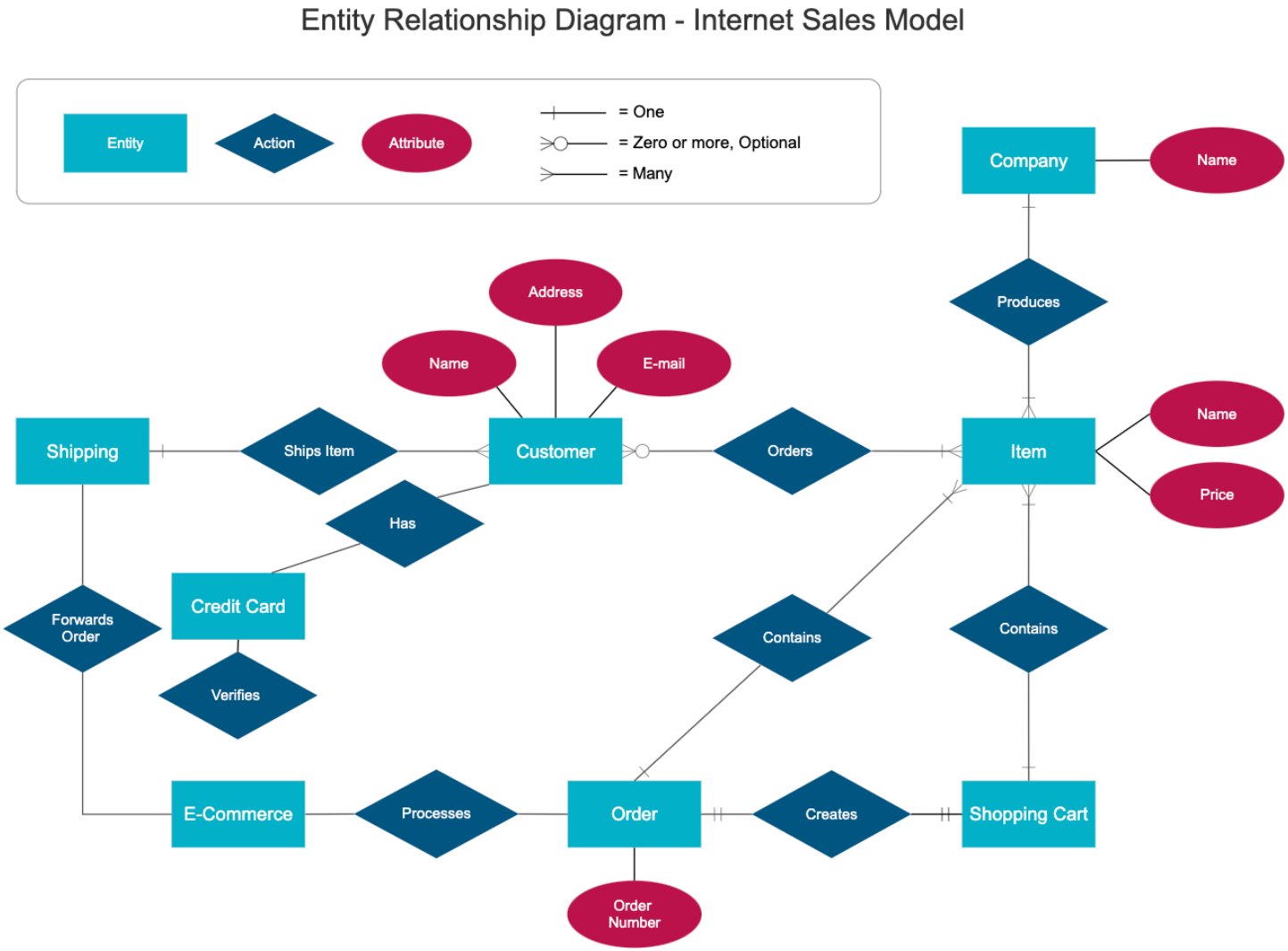
Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

ER diagram example



- more information: https://www.geeksforgeeks.org/introduction-of-er-model/

ER diagram example



Environment settings

- **Download and install**
  - https://dev.mysql.com/downloads/installer/

- Start MySQL server
  - `mysqld` command in the command prompt (run as an administrator)

  - Services.msc, mysql service …
  - Setting Up a MySQL User Account

Administrative MySQL Commands

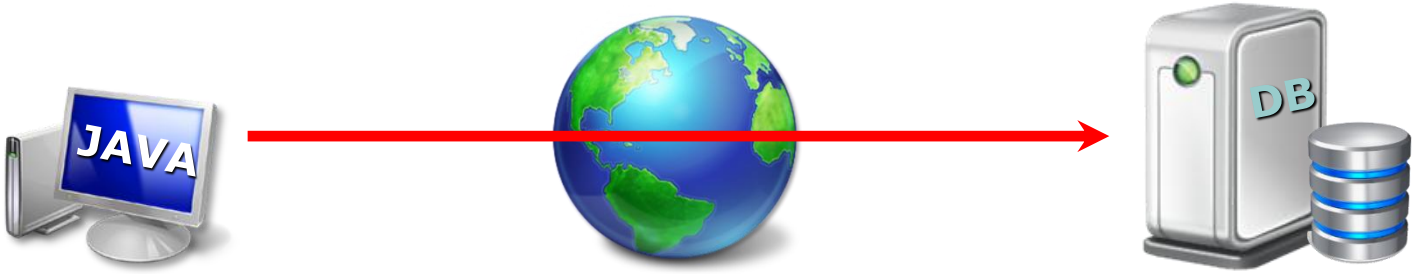
- **USE database_name** – This will be used to select a database in the MySQL
- **SHOW DATABASES** – Lists out the databases that are accessible by the MySQL DBMS
- **SHOW TABLES** – Displays the list of the tables in the current database
- **SHOW COLUMNS FROM *table_name:*** Shows the attributes, types of attributes, key information, whether NULL is permitted, defaults, and other information for a table
- **SHOW INDEX FROM table_name** – Presents the details of all indexes on the table, including the PRIMARY KEY
- **SHOW TABLE STATUS LIKE table_name\G** – Reports details of the MySQL DBMS performance and statistics

Most Important SQL Commands

- **SELECT** - extract data from a database
- **UPDATE** - update data in a database
- **DELETE** - delete data from a database
- **INSERT INTO** – insert new data into a database
- **CREATE DATABASE** - create a new database
- **ALTER DATABASE** - modify a database
- **CREATE TABLE** - create a new table
- **ALTER TABLE** - modify a table
- **DROP TABLE** - delete a table
- **CREATE INDEX** - create an index (search key)
- **DROP INDEX** - delete an index

JDBC

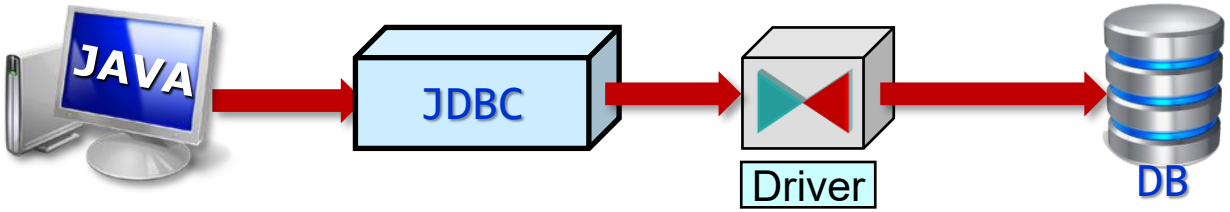
- **Goal: managing a database using Java programming language**



- **Database independence**: it would be nice to write universal code that can handle multiple (almost any) databases

JDBC

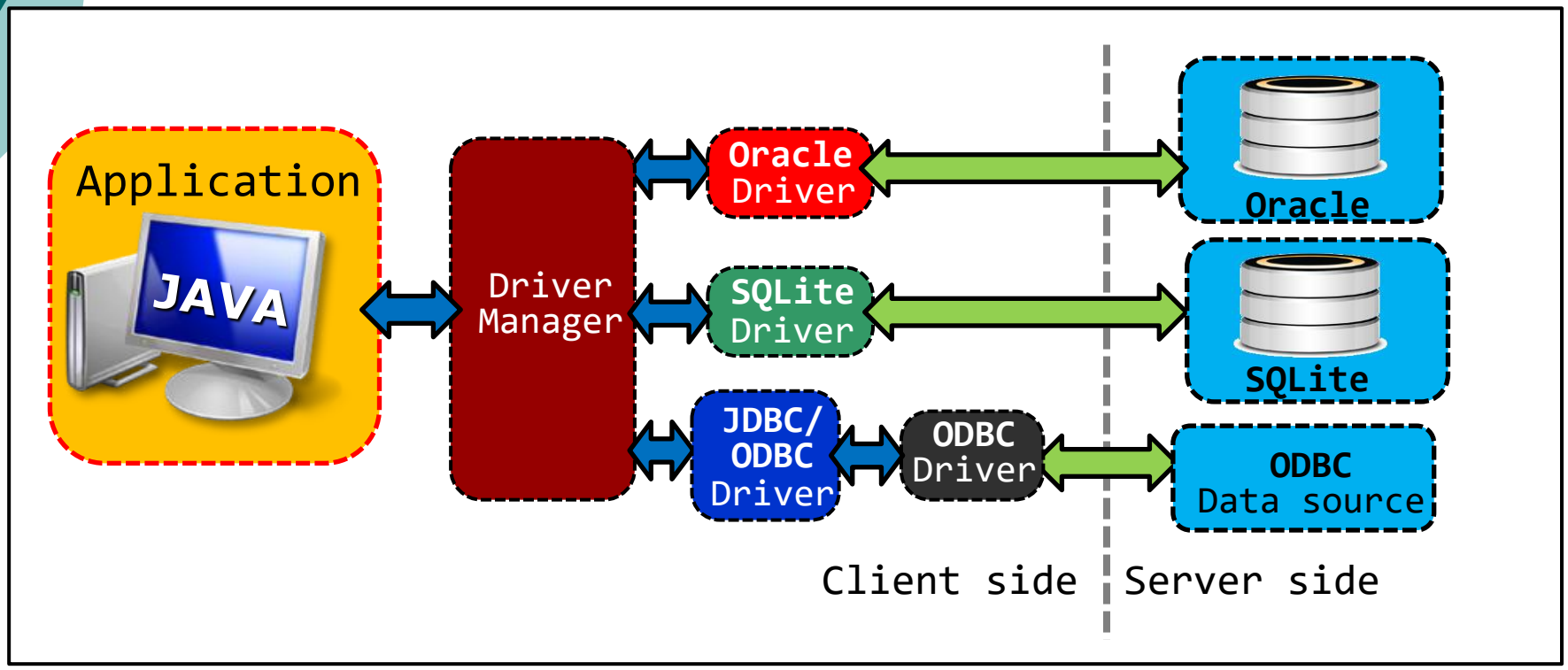
- **Solution**



  - Java code uses the JDBC API
  - The JDBC API loads the driver
  - The communication between the Java code and the database through the driver
  - The driver must adapt to the database, meaning it must be created differently for each database
    - different database → different driver

JDBC

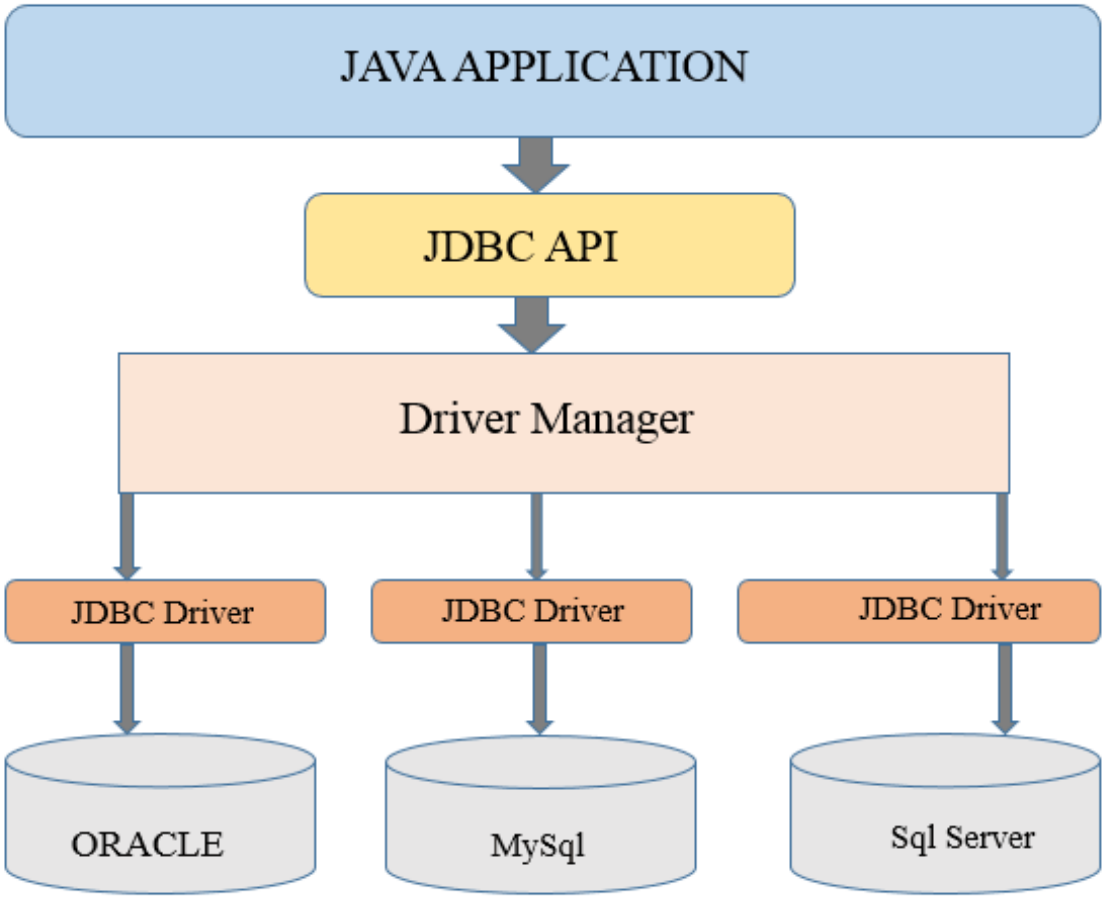
- different database → different driver



Java connector to MySQL

- To communicate with databases Java provides a library known as **JDBC (Java Database Connectivity)**
- **JDBC provides a set of classes and methods specifically designed for database connectivity**, enabling Java developers to perform tasks such as establishing connections, executing queries, and managing data in MySQL databases
- Need to use a JDBC (Java Database Connectivity) driver to connect your Java application to a MySQL database
- Have to download MySQL Connector
  - mysql-connector-j-9.1.0.jar
  - https://dev.mysql.com/downloads/connector/j/
  - https://dev.mysql.com/downloads/file/?id=534782

Java connector to MySQL



JDBC methods

`DriverManager.getConnection(String url, String user, String password)`	Establishes a connection to the database using the specified URL, username, and password
`createStatement()`	Creates a Statement object for executing SQL queries
`executeQuery(String sql)`	Executes a SQL SELECT query and returns a ResultSet object containing the result set
`executeUpdate(String sql)`	Executes a SQL INSERT, UPDATE, DELETE, or other non-query statement
`next()`	Moves the cursor to the next row in the result set. Returns true if there is a next row, false otherwise
`getInt(String columnLabel)`	Retrieves the value of the specified column in the current row of the result set

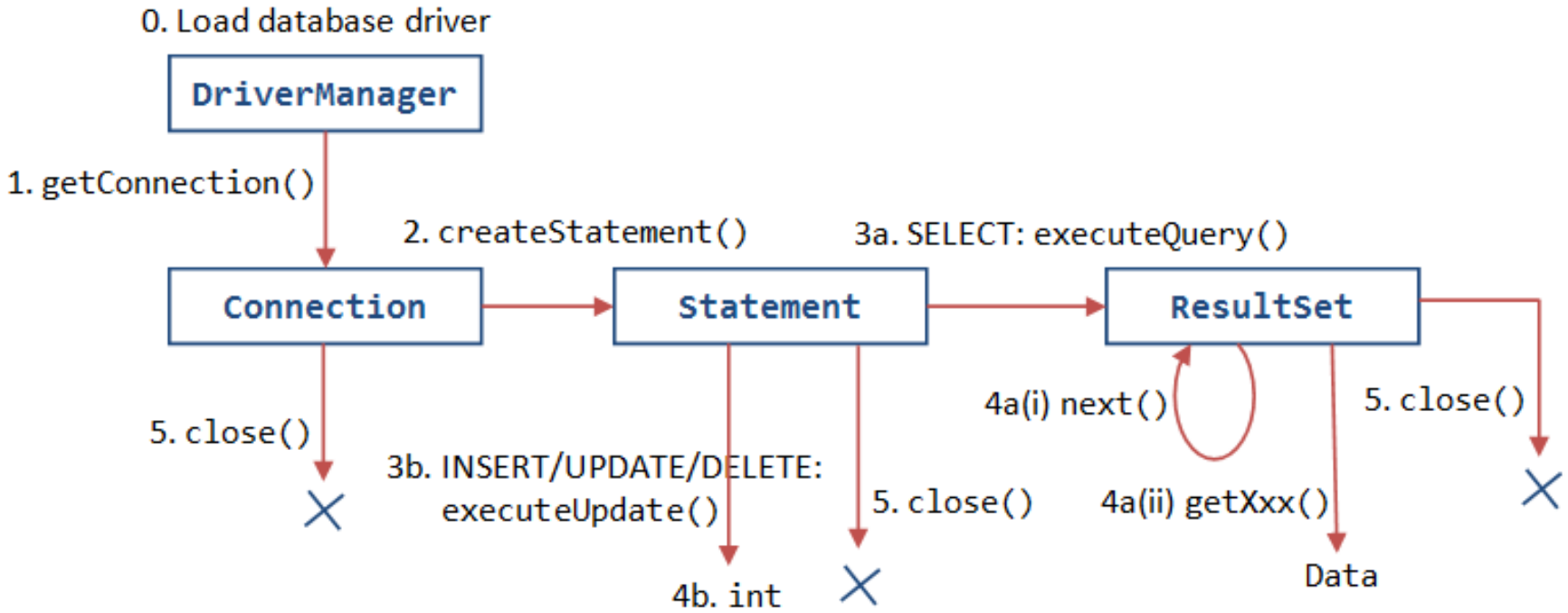
JDBC methods

`prepareStatement(String sql)`	Creates a PreparedStatement object for executing parameterized SQL queries
`setXXX(int parameterIndex, XXX value)`	Sets the value of a specified parameter in the prepared statement
`executeQuery(), executeUpdate()`	Execute the prepared statement as a query or update
`setAutoCommit(boolean autoCommit)`	Enables or disables auto-commit mode
`commit()`	Commits the current transaction
`rollback()`	Rolls back the current transaction

JDBC steps

1. **Load the JDBC driver** specific to your database
2. **Create a connection** to the database using `DriverManager.getConnection()`
3. **Create a "Statement"** or "PreparedStatement" for executing SQL queries
4. **Use** `executeQuery()` for SELECT queries, or `executeUpdate()` for other statements
5. **Iterate** through the "**ResultSet**" to process the retrieved data
6. **Close "ResultSet"**, "Statement", and "Connection" to release resources
7. Wrap database code in try-catch blocks to handle exceptions
8. Use transactions if performing multiple operations as a single unit

JDBC steps


0. Load database driver
DriverManager
1. getConnection()
2. createStatement()
3a. SELECT: executeQuery()
Connection
Statement
ResultSet
5. close()
3b. INSERT/UPDATE/DELETE:
executeUpdate()
5. close()
4a(i) next()
5. close()
4a(ii) getXxx()
4b. int
Data

JDBC example class


```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DatabaseInteractionExample {

    public static void main(String[] args) {
        try {
            // Load JDBC Driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Connect to Database
            Connection connection =
            DriverManager.getConnection("jdbc:mysql://localhost:3306/your_database",
            "your_username", "your_password");

            // Execute Query
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery("Your SQL Query");
```

JDBC example class


```
            // Process Results
            while (resultSet.next()) {
            // Process data
            }

            // Close Resources
            resultSet.close();
            statement.close();
            connection.close();

        // Handle Exceptions
        } catch (ClassNotFoundException | SQLException e) {
        e.printStackTrace();
        }
    }
}
```

Variables

- Main purpose of a variable is to label a memory location(s) and store data in it so that it can be used throughout the program

- **In MySQL, there are three types of variables:**
  - **User-Defined Variable**
  - **Local Variable**
  - **System Variables**

Variables

- **User-Defined Variable**
  - allows us to store a value in one statement and subsequently refer to it in another
  - these variable names will have the symbol "@" as a prefix

```
SELECT @variable_name = value
```

```
SELECT @max_salary := MAX(salary) FROM CUSTOMERS;
```

Variables

- **Local Variable**
  - local variable can be declared using the **DECLARE** keyword
  - strongly typed variable, which means that we definitely need to declare a data type

```
DECLARE variable_name1, variabale_name2, ...
data_type [DEFAULT default_value];
```

```
DELIMITER //
CREATE PROCEDURE salaries()
BEGIN
  DECLARE Ramesh INT;
  DECLARE Khilan INT DEFAULT 30000;
  DECLARE Kaushik INT;
  DECLARE Chaitali INT;
  DECLARE Total INT;
  SET Ramesh = 20000;
  SET Kaushik = 25000;
  SET Chaitali = 29000;
  SET Total = Ramesh+Khilan+Kaushik+Chaitali;
  SELECT Total,Ramesh,Khilan,Kaushik,Chaitali;
END //
```

Variables

- **System Variables**
  - contains the data we need, to work with the database
  - the SET command in MySQL can be used at the runtime to dynamically change the values of the system variables

  - there are two variable scope modifiers
    - The GLOBAL variables are active throughout the lifecycle
    - The SESSION variables can be available only in the current session

```
SHOW [GLOBAL | SESSION] VARIABLES;
```

```
SHOW VARIABLES LIKE '%table%';
```

Database connection

- Must first establish a connection between client and the databse
- Connection parameters: consisting of a username and a password

- Set Password to MySQL Root

```
mysql -u root password "new_password";
```

- Reset Password

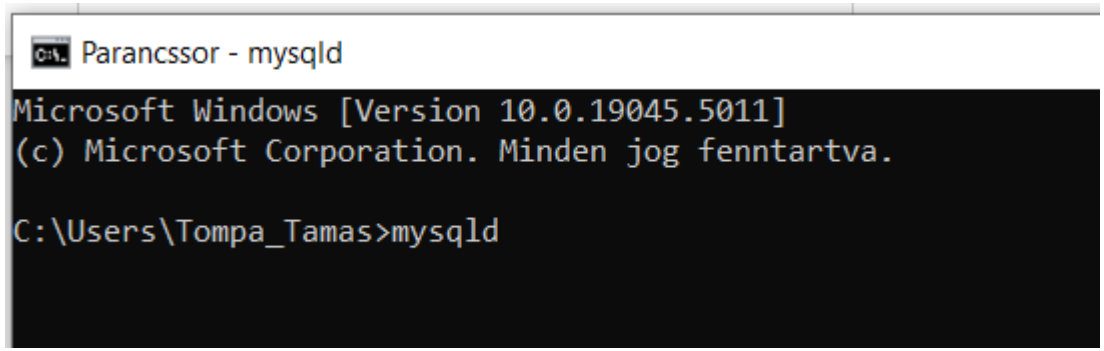
```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('password_name'); FLUSH PRIVILEGES;
```

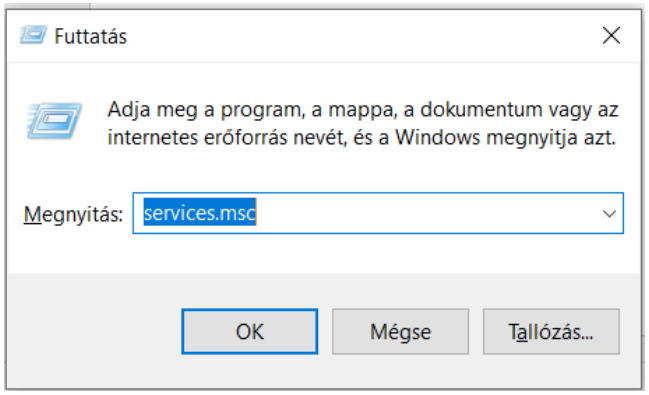
- Connect to the MySQL server from the command prompt

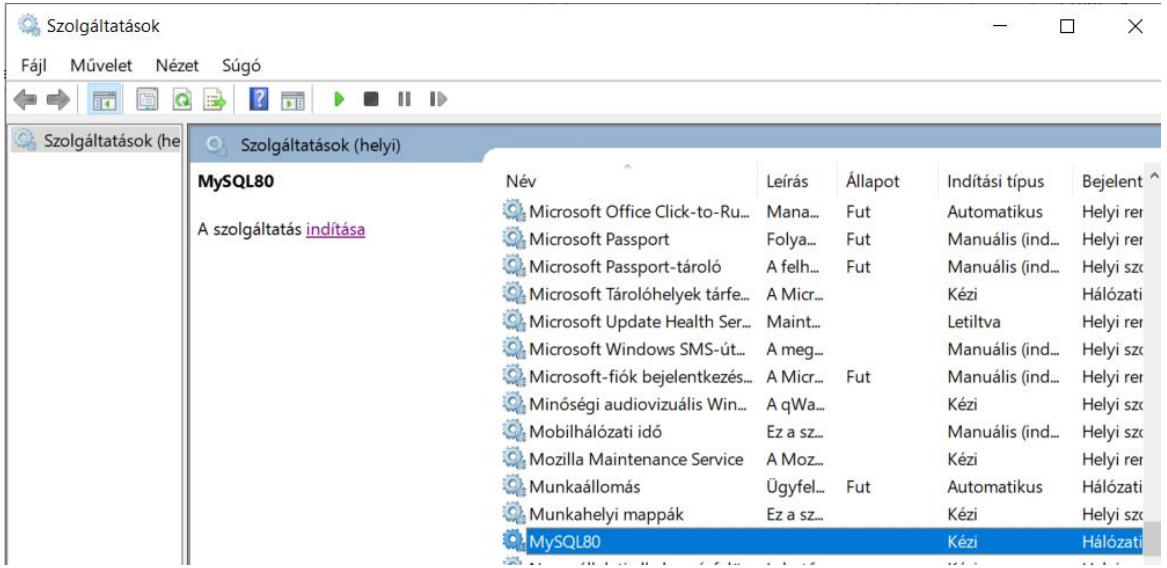
```
mysql -u root -p
```

MySQL server starting

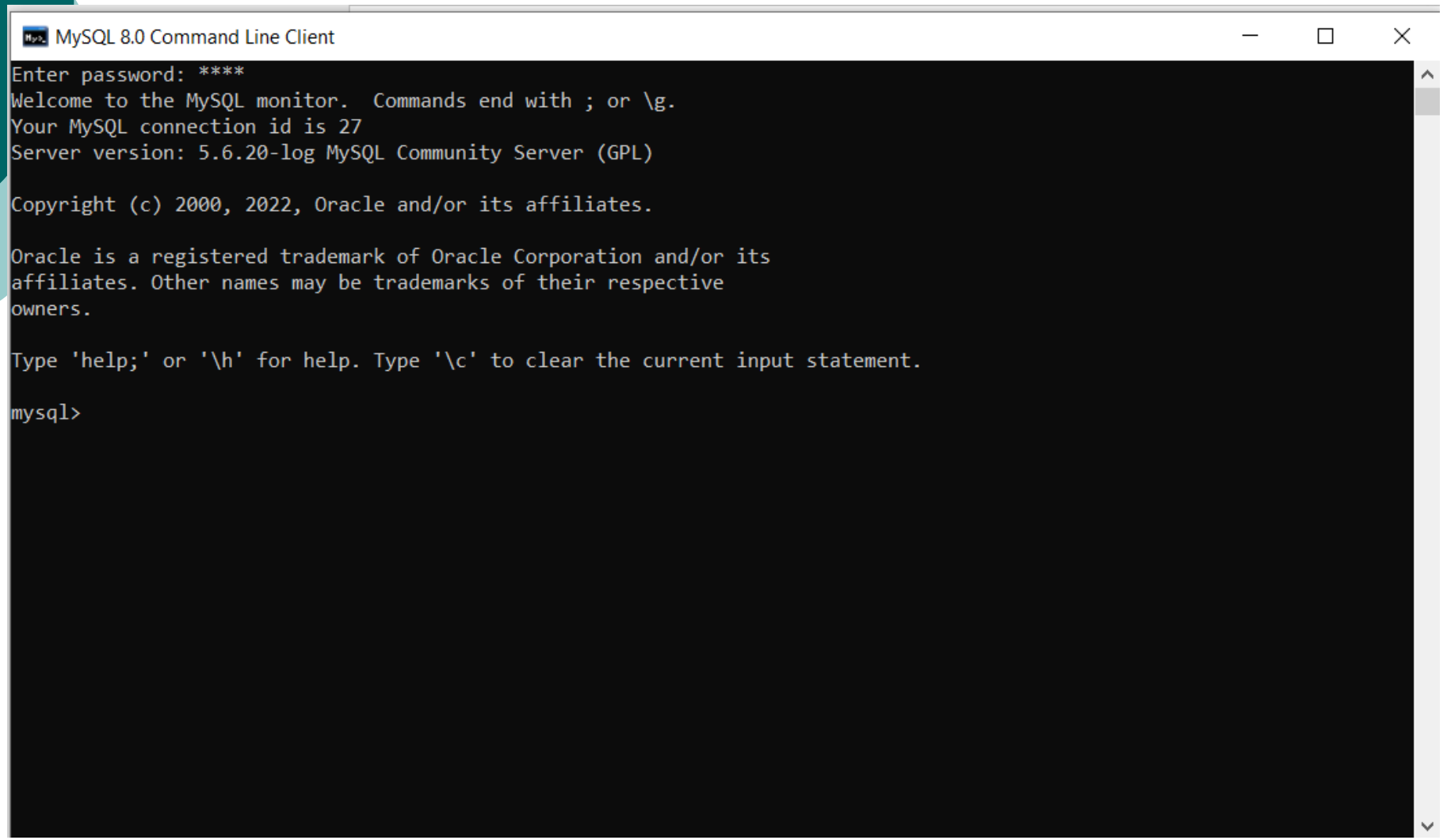
- **mysqld** command (in the cmd)







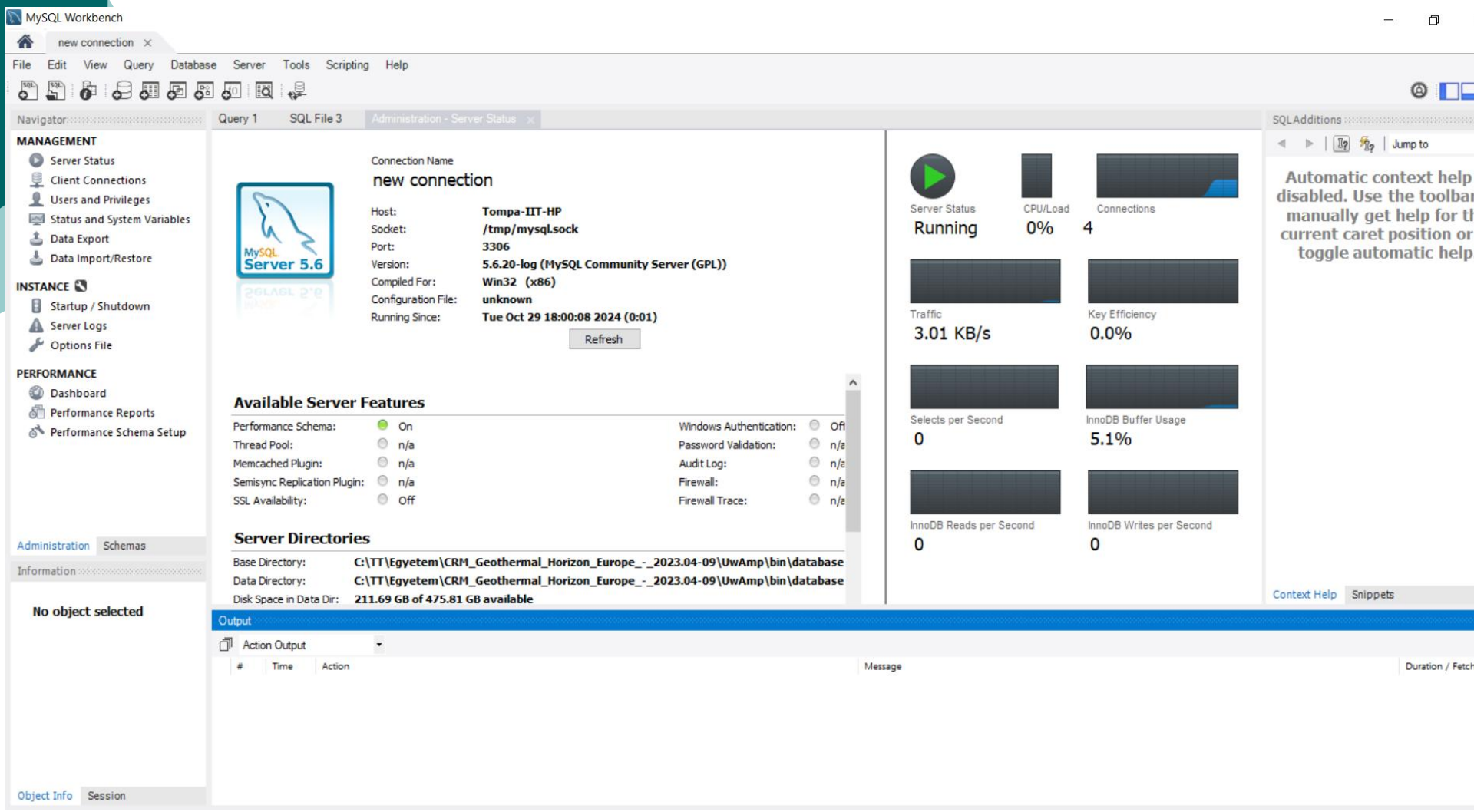
MySQL command line client



MySQL workbench

- The MySQL workbench is a **graphical tool for working with MySQL servers and databases**
- It is developed and maintained by Oracle
- This application includes various features such as **data modelling, data migration, SQL development, server administration, database backup, database recovery** and many more
- Features:
  - SQL Development
  - Data modelling
  - Server adminstration
  - Data migration

MySQL workbench



Create Database

- The **CREATE DATABASE** statement is a DDL (Data Definition Language) statement used to create a new database in MySQL RDBMS

```
CREATE DATABASE DatabaseName;
```

```
CREATE DATABASE hello_world;
```

- Verification
  - once the database TUTORIALS is created, you can check it in the list of databases using the SHOW statement

```
SHOW DATABASES;
```

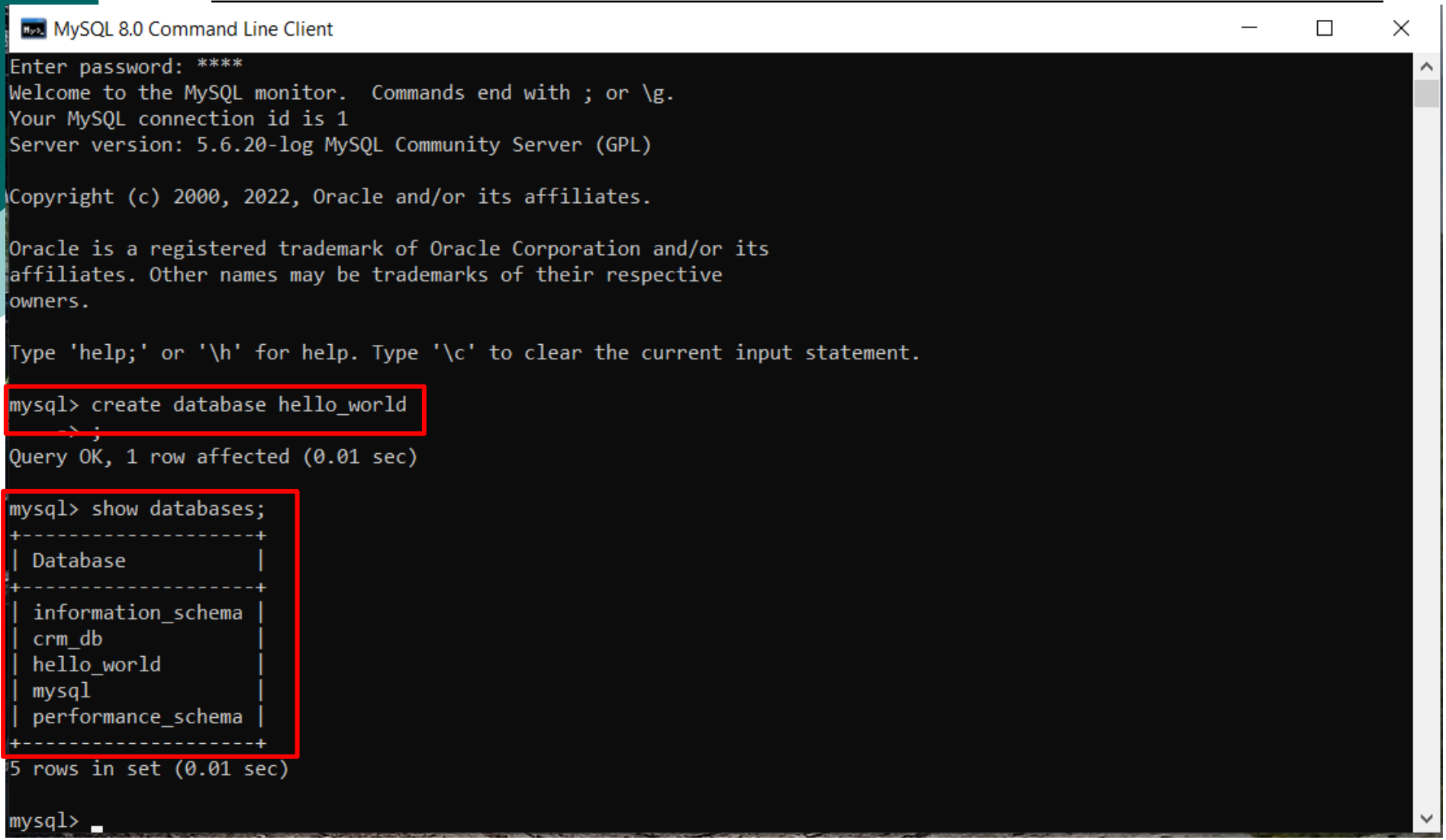
- If the database is existing then an error will be generated

```
CREATE DATABASE IF NOT EXISTS myDatabase
```

- In Java:

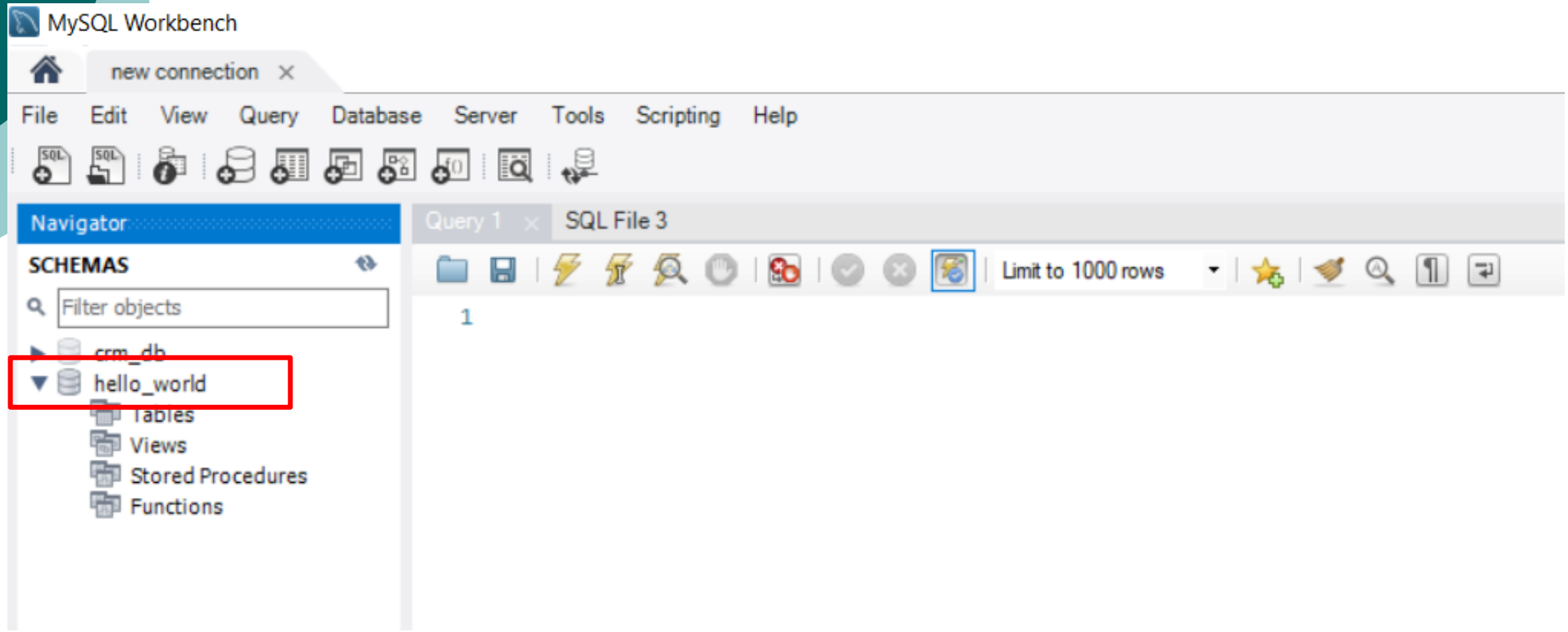
```
String sql = "CREATE DATABASE DatabaseName";
st.executeUpdate(sql);
```

Create Database



Create Database

- The created database can be check in the MySQL Workbench



Create Database (Java)


```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class createDatabase {
    public static void main(String[] args) {
          String url = "jdbc:mysql://localhost:3306/";
          String user = "root";
          String password = "password";
          ResultSet rs;

          try {
               Class.forName("com.mysql.cj.jdbc.Driver");
               Connection con = DriverManager.getConnection(url, user, password);
               Statement st = con.createStatement();
               //System.out.println("Connected successfully...!");
               String sql = "CREATE DATABASE TUTORIALS";
               st.execute(sql);
               System.out.println("Database created successfully...!");
          } catch(Exception e) {
              e.printStackTrace();
          }
    }
}
```

Create Database (Python)


```
import mysql.connector

# creating the connection object
connection = mysql.connector.connect( host ="localhost", user ="root", password ="password" )

# creating cursor object
cursorObj = connection.cursor()

# creating the database cursorObj.execute("CREATE DATABASE MySqlPythonDB")
print("Database Created Successfully")

# disconnecting from server
connection.close()
```

Drop Database

- The **DROP DATABASE** statement in MySQL is used to delete a database along with all the data such as tables, views, indexes, stored procedures, and constraints

```
DROP DATABASE DatabaseName;
```

```
DROP DATABASE TUTORIALS;
```

- Verification

```
SHOW DATABASES;
```

- Dropping a Database using mysqladmin

```
mysqladmin -u root -p drop DatabaseName
```

- Java

```
String sql = "DROP DATABASE DatabaseName;";
st.execute(sql);
```

Drop Database

Drop Database (Java)


```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class DropDatabase {
     public static void main(String[] args) {
         String url = "jdbc:mysql://localhost:3306/tutorials";
         String user = "root";
         String password = "password";
         ResultSet st;
         try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url, user, password);
            Statement st1 = con.createStatement();
            //System.out.println("Connected successfully...!");
            String sql = "DROP DATABASE TUTORIALS";
            st1.execute(sql);
            System.out.println("Database dropped successfully...!");
         }catch(Exception e) {
            e.printStackTrace();
         }
     }
}
```

Select database

- Once you get connected with the MySQL server, it is required to select a database to work with
  - this is because there might be more than one database available with the MySQL Server

- To select a database in MySQL, can be use the SQL **USE** statement

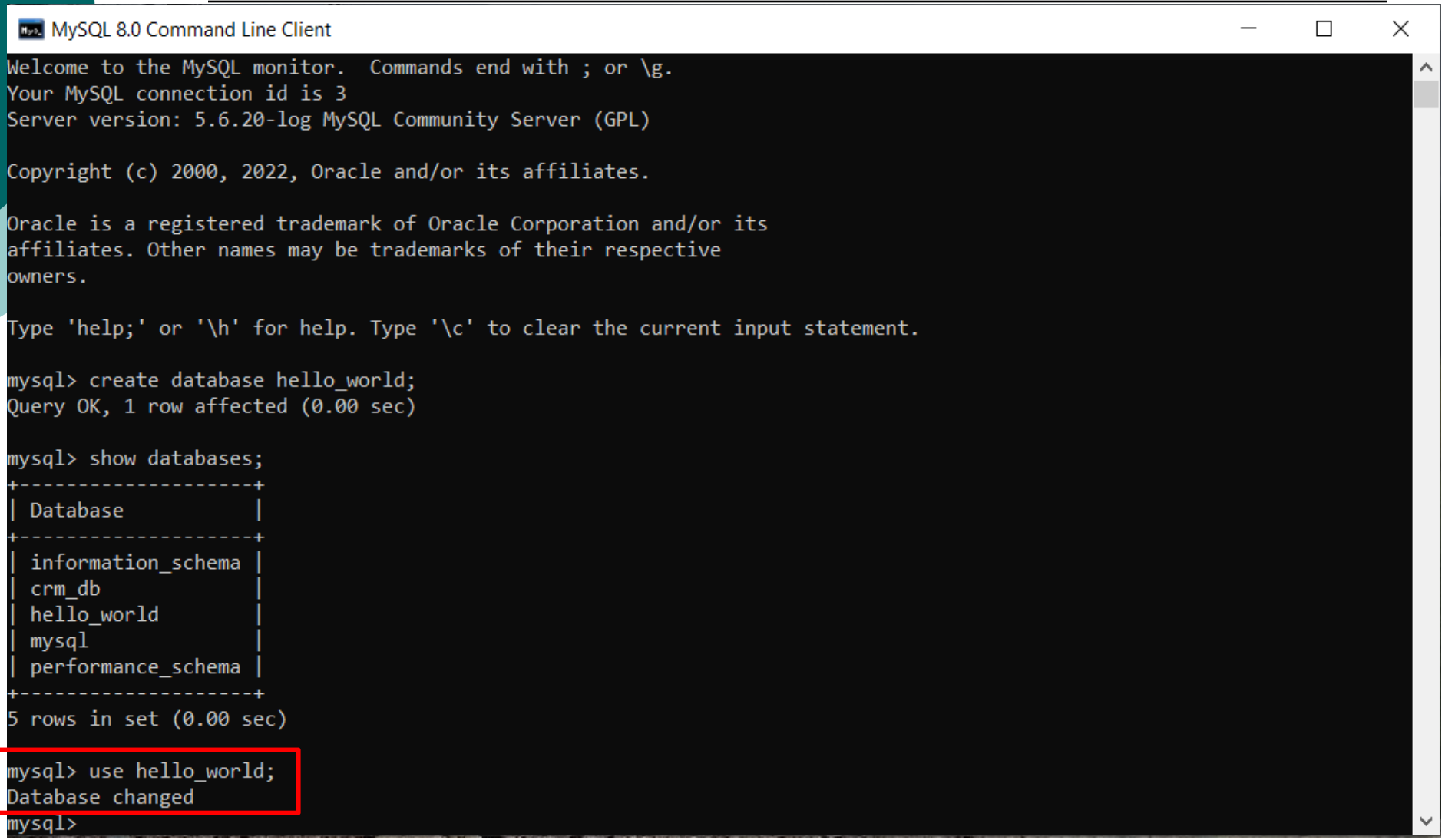
```
USE DatabaseName;
```

```
USE hello_world;
```

- Java

```
String sql = "USE Database_name";
st.execute(sql);
```

Select database



Select database (Java)


```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class SelectDatabase {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/";
        String user = "root";
        String password = "password";
        System.out.println("Connecting to select database.....!");

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url, user, password);
            Statement st1 = con.createStatement();
            String sql = "USE TUTORIALS";
            st1.execute(sql);
            System.out.println("Database selected successfully...!");
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Show database (Java)

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

    public class ShowDatabase {
        public static void main(String[] args) {
             String url = "jdbc:mysql://localhost:3306/TUTORIALS";
             String user = "root";
             String password = "password";
             ResultSet rs;
             try {
                Class.forName("com.mysql.cj.jdbc.Driver");
                Connection con = DriverManager.getConnection(url, user, password);
                Statement st1 = con.createStatement();
                //System.out.println("Database connected successfully...!");
                String sql = "SHOW DATABASES";
                rs = st1.executeQuery(sql);
                System.out.println("Show query executed successfully...!");
                System.out.println("Databases are: ");
                while(rs.next()) {
                   String db = rs.getNString(1);
                   System.out.println(db);
                }
             } catch(Exception e) {
                e.printStackTrace();
             }
            }
}
```

Export database

- Exporting a database in MySQL is commonly used for backup purposes or transferring data between servers
- Can be export entire database or just a portion of it
- The simplest way of exporting a database is by using the **mysqldump** command-line tool

```
mysqldump -u username -p database_name > output_file_path
```

  - **username**: It is the MySQL username to use when connecting to the database
  - **database_name**: It is the name of the database to be exported
  - **output_file_path**: It is the path of the backup file. This is where the backup data will be stored
  - **>**: This symbol **exports** the output of the mysqldump command into a file named *output_file_path*

```
mysqldump -u root -p TUTORIALS > data-dump.sql
```

Import database

- In MySQL, to import an existing dump or backup file into a database, can use the **mysql** command-line tool

```
mysql -u username -p new_database_name < dumpfile_path
```

  - **username**: This is the MySQL username to use when connecting to the MySQL server
  - **new_database_name**: The name of the database where you want to import the data
  - **dumpfile_path**: It is the path of the backup file. The data will be imported from this file
  - **<**: This symbol **imports** the data from the file named *output_file_path*

```
mysql -u root -p testdb < data-dump.sql
```

Create users

- Can create a new user account using the CREATE USER Statement in
- To execute this statement, the current account must have the CREATE USER privilege or the INSERT privilege for the MySQL system schema

```
CREATE USER 'user_name'@'host_name' IDENTIFIED BY 'password';
```

  - **user_name** is the name of the user you need to create
  - **hostname** specifies the host from which the user can connect
  - **password** is the user's password

```
CREATE USER 'sample'@'localhost' IDENTIFIED BY '123456';
```

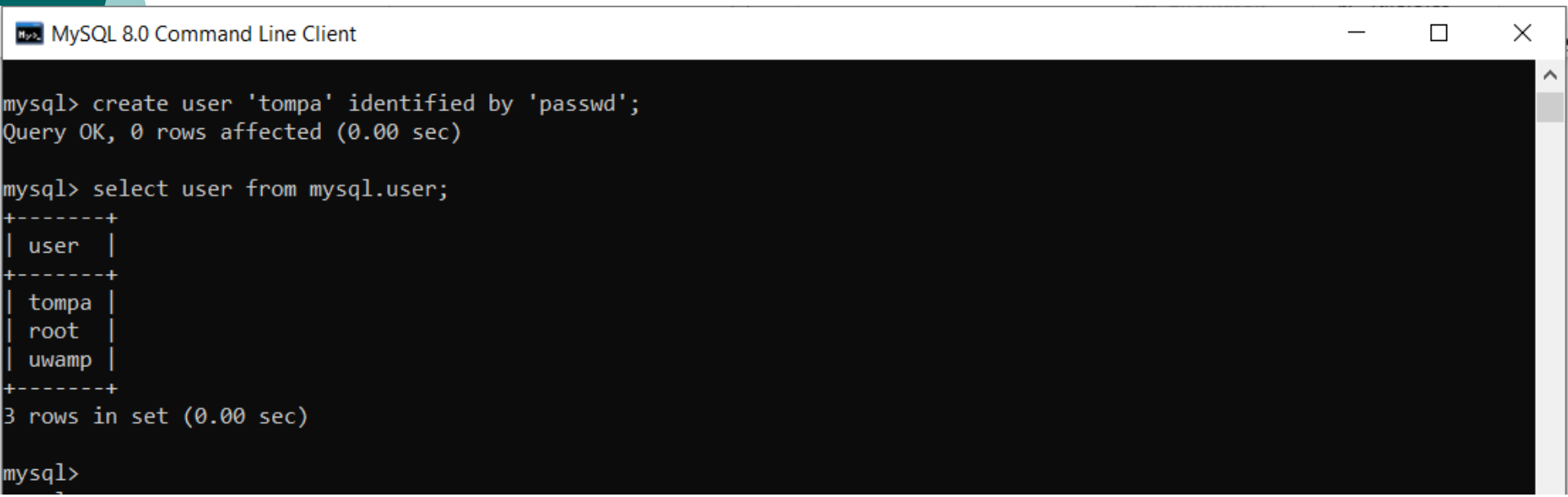
- Verification

```
SELECT USER FROM MySQL.USER;
```

```
SELECT * FROM mysql.user;
```

```
String sql = "SELECT USER FROM MYSQL.USER";
statement.executeQuery(sql);
```

Create users



Create users (Java)

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;


public class CreateUsers {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/TUTORIALS";
        String user = "root";
        String password = "password";

        try {
           Class.forName("com.mysql.cj.jdbc.Driver");
           Connection con = DriverManager.getConnection(url, user, password);
           Statement st = con.createStatement();
           //System.out.println("Database connected successfully...!");
           String sql = "CREATE USER 'Vivek'@'localhost' IDENTIFIED WITH mysql_native_password
BY 'password'";
           st.execute(sql);
           System.out.println("User 'Vivek' created successfully...!");
        } catch(Exception e) {
           e.printStackTrace();
        }
    }
}
```

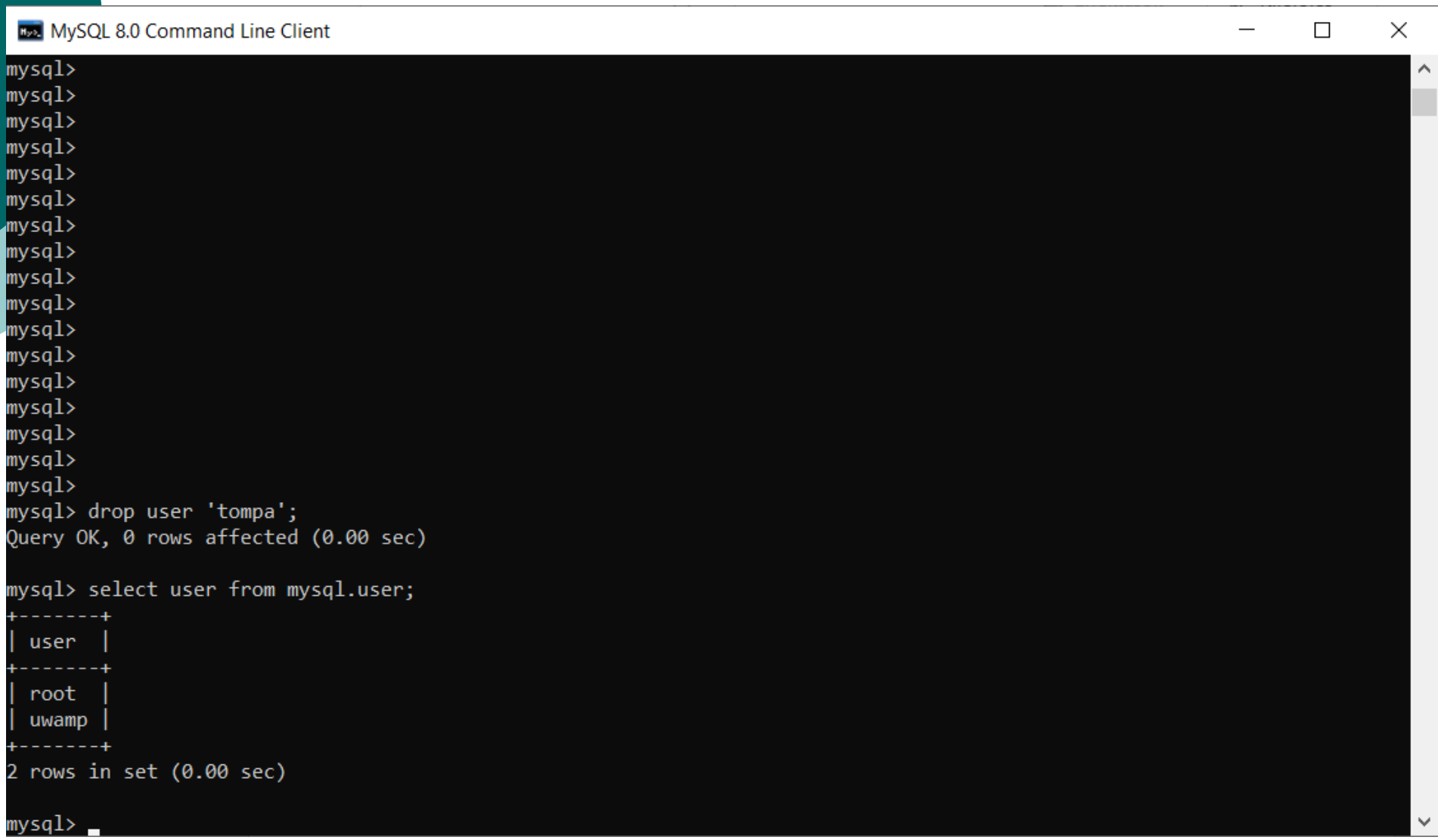
Drop users

- Can be drop/delete one or more existing users in MySQL using the DROP USER Statement
- Once you delete an account, all privileges of it are deleted
- To execute this statement, you need to have CREATE USER privilege

```
DROP USER [IF EXISTS] 'username'@'hostname';
```

```
DROP USER TestUser@localhost;
```

Drop users


MySQL 8.0 Command Line Client
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> drop user 'tompa';
Query OK, 0 rows affected (0.00 sec)
mysql> select user from mysql.user;
+-------+
| user |
+-------+
| root |
| uwamp |
+-------+
2 rows in set (0.00 sec)
mysql>

Drop users (Java)


```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
public class DropUsers {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/TUTORIALS";
        String user = "root";
        String password = "password";

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url, user, password);
            Statement st = con.createStatement();
            //System.out.println("Database connected successfully...!");
            String sql = "DROP USER 'Vivek'@'localhost'";
            st.execute(sql);
            System.out.println("User 'Vivek' dropped successfully...!");
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Create table

- SQL is used to store data in the form of structured tables
- These tables consist of fields and records
  - a field represents a column that defines the type of data to be stored in a table, and a record is a row containing the actual data
- The table creation command requires the following details:
  - name of the table
  - name of the columns
  - definitions for each column

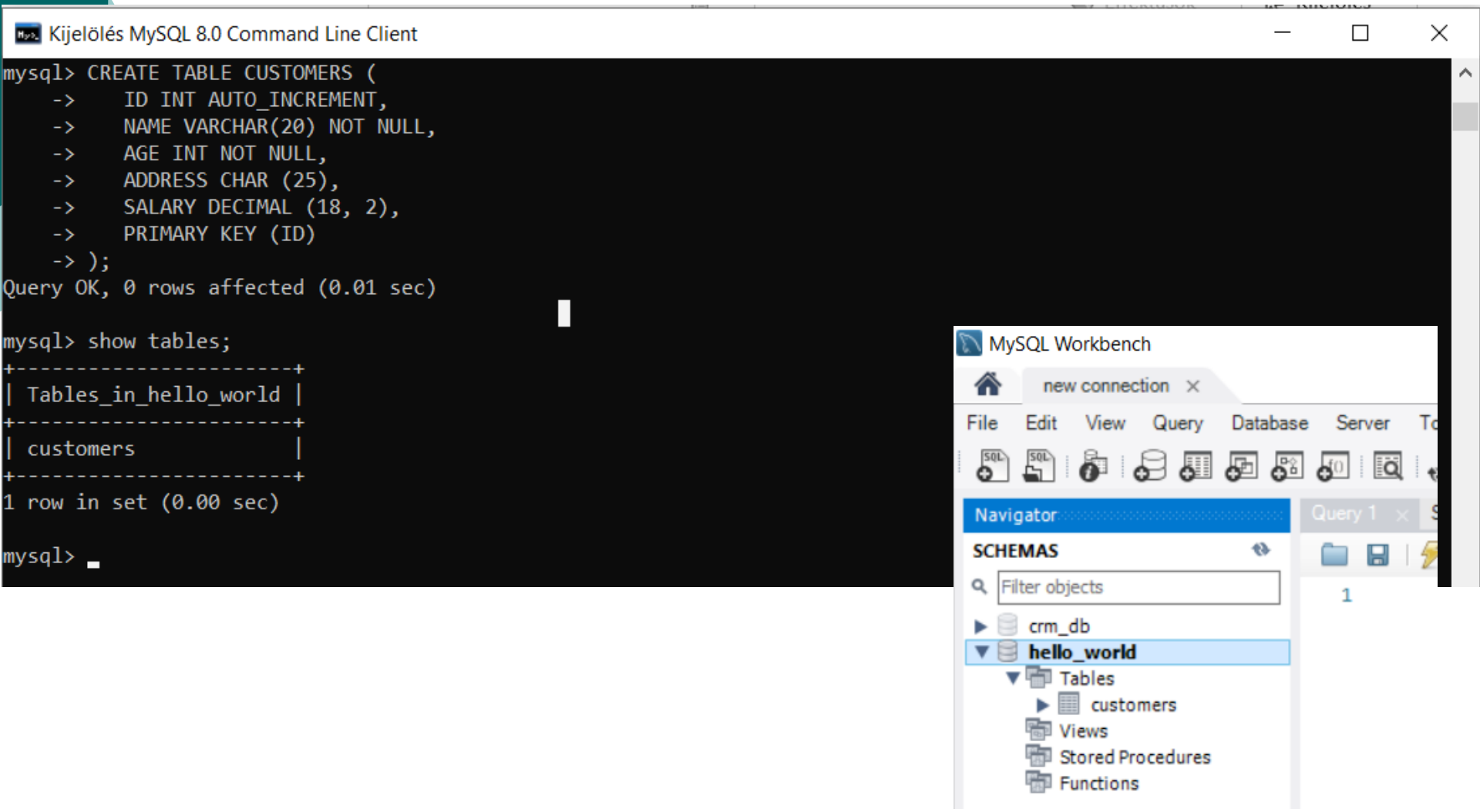
```
CREATE TABLE table_name(
   column1 datatype,
   column2 datatype,
.  ....
   columnN datatype,
   PRIMARY KEY( one or more columns )
);
```

```
CREATE TABLE CUSTOMERS (
   ID INT AUTO_INCREMENT,
   NAME VARCHAR(20) NOT NULL,
   AGE INT NOT NULL,
   ADDRESS CHAR (25),
   SALARY DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

- Verification: `DESC TABLENAME;`

`DESC CUSTOMERS;` or: `SHOW TABLES;`

Create table



Create table

- **AUTO_INCREMENT**
  - automatically increments the value in the ID column by one for each new record you add. It starts from the next available number
- **NOT NULL**
  - the field to be NULL.
  - if a user tries to create a record with a NULL value in that field, then MySQL will raise an error
- **PRIMARY KEY**
  - used to define a column as a primary key
  - ensures that every record in that column is unique
  - can be also use it for multiple columns by separating them with commas

Queries

- The queries in MySQL are commands that are used to retrieve or manipulate the data from a database table
  - SELECT, UPDATE, DELETE, INSERT INTO, CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE DATABASE, ALTER DATABASE, CREATE INDEX, DROP INDEX, etc

```
CREATE DATABASE tutorials;
```

```
USE tutorials;
```

```
CREATE TABLE CUSTOMERS (
   ID int,
   NAME varchar(20),
   AGE int,
   PRIMARY KEY (ID)
);
```

Insert Query

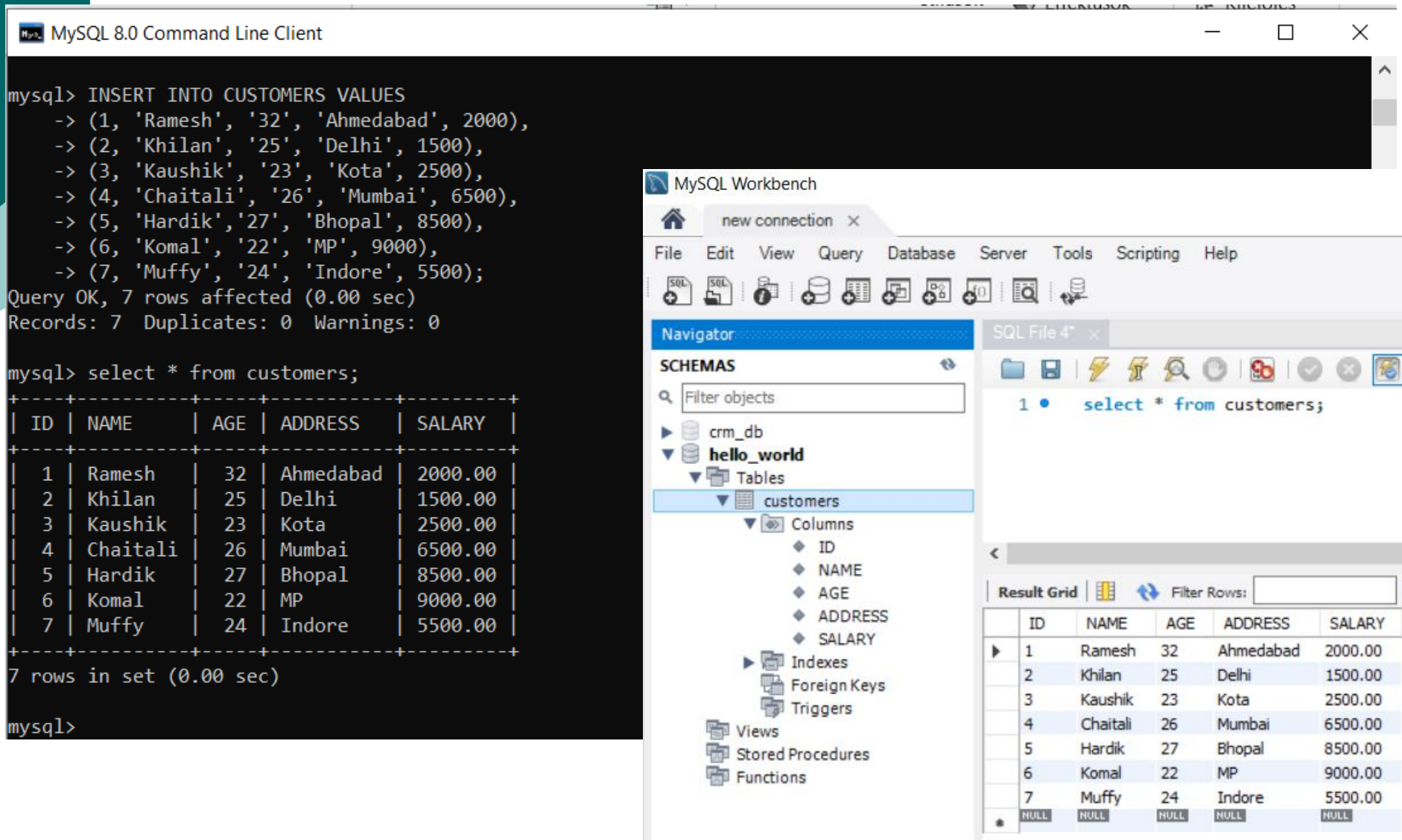
- The MySQL insert query can be used to insert records within a specified table

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO CUSTOMERS (ID, NAME, AGE) VALUES (1, "Nikhilesh", 28);
INSERT INTO CUSTOMERS (ID, NAME, AGE) VALUES (2, "Tomy", 36);
INSERT INTO CUSTOMERS (ID, NAME, AGE) VALUES (3, "Joe", 22);
INSERT INTO CUSTOMERS (ID, NAME, AGE) VALUES (4, "Kate", 21);
INSERT INTO CUSTOMERS (ID, NAME, AGE) VALUES (5, "Jonh", 42);
```

```
INSERT INTO CUSTOMERS VALUES
(1, 'Ramesh', '32', 'Ahmedabad', 2000),
(2, 'Khilan', '25', 'Delhi', 1500),
(3, 'Kaushik', '23', 'Kota', 2500),
(4, 'Chaitali', '26', 'Mumbai', 6500),
(5, 'Hardik','27', 'Bhopal', 8500),
(6, 'Komal', '22', 'MP', 9000),
(7, 'Muffy', '24', 'Indore', 5500);
```

Insert Query

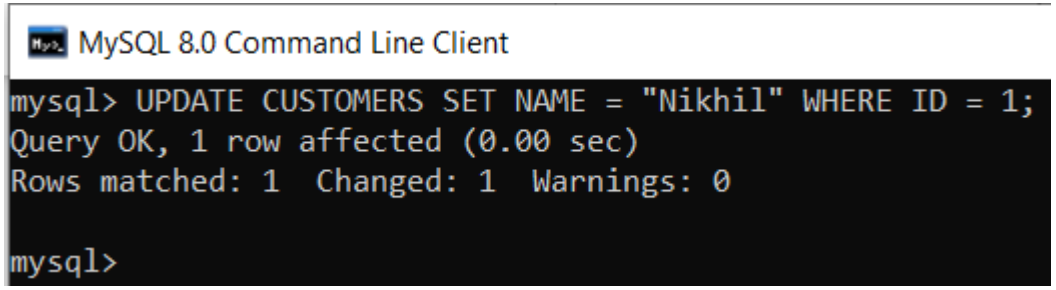


Update Query

- The MySQL update query can be used to **modify the existing records in a specified table**

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

```
UPDATE CUSTOMERS SET NAME = "Nikhil" WHERE ID = 1;
```



Update Query

```
UPDATE CUSTOMERS SET NAME = "Nikhil" WHERE ID = 1;
```

```
+----+----------+-----+-----------+---------+
| ID | NAME     | AGE | ADDRESS   | SALARY  |
+----+----------+-----+-----------+---------+
|  1 | Ramesh   |  32 | Ahmedabad | 2000.00 |
|  2 | Khilan   |  25 | Delhi     | 1500.00 |
|  3 | Kaushik  |  23 | Kota      | 2500.00 |
|  4 | Chaitali |  26 | Mumbai    | 6500.00 |
|  5 | Hardik   |  27 | Bhopal    | 8500.00 |
|  6 | Komal    |  22 | MP        | 9000.00 |
|  7 | Muffy    |  24 | Indore    | 5500.00 |
+----+----------+-----+-----------+---------+
7 rows in set (0.00 sec)

mysql> UPDATE CUSTOMERS SET NAME = "Nikhil" WHERE ID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from customers;
+----+----------+-----+-----------+---------+
| ID | NAME     | AGE | ADDRESS   | SALARY  |
+----+----------+-----+-----------+---------+
|  1 | Nikhil   |  32 | Ahmedabad | 2000.00 |
|  2 | Khilan   |  25 | Delhi     | 1500.00 |
|  3 | Kaushik  |  23 | Kota      | 2500.00 |
|  4 | Chaitali |  26 | Mumbai    | 6500.00 |
|  5 | Hardik   |  27 | Bhopal    | 8500.00 |
|  6 | Komal    |  22 | MP        | 9000.00 |
|  7 | Muffy    |  24 | Indore    | 5500.00 |
+----+----------+-----+-----------+---------+
```

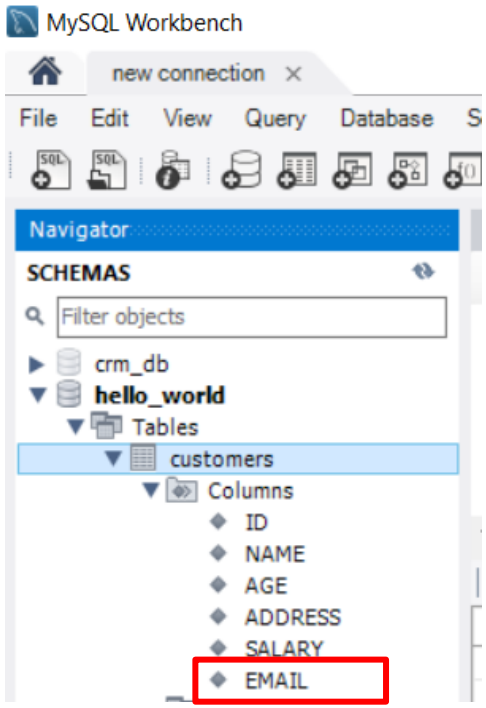
Alter Query

- The ALTER query in MySQL can be used to **add, delete, or modify columns in an existing table**

```
ALTER TABLE table_name
[ADD|DROP] column_name datatype;
```

```
ALTER TABLE CUSTOMERS ADD COLUMN EMAIL varchar(50);
```

```
mysql> ALTER TABLE CUSTOMERS ADD COLUMN EMAIL varchar(50);
Query OK, 7 rows affected (0.01 sec)
Records: 7  Duplicates: 0  Warnings: 0

mysql> select * from customers;
+----+----------+-----+-----------+---------+-------+
| ID | NAME     | AGE | ADDRESS   | SALARY  | EMAIL |
+----+----------+-----+-----------+---------+-------+
|  1 | Nikhil   |  32 | Ahmedabad | 2000.00 | NULL  |
|  2 | Khilan   |  25 | Delhi     | 1500.00 | NULL  |
|  3 | Kaushik  |  23 | Kota      | 2500.00 | NULL  |
|  4 | Chaitali |  26 | Mumbai    | 6500.00 | NULL  |
|  5 | Hardik   |  27 | Bhopal    | 8500.00 | NULL  |
|  6 | Komal    |  22 | MP        | 9000.00 | NULL  |
|  7 | Muffy    |  24 | Indore    | 5500.00 | NULL  |
+----+----------+-----+-----------+---------+-------+
7 rows in set (0.00 sec)
```

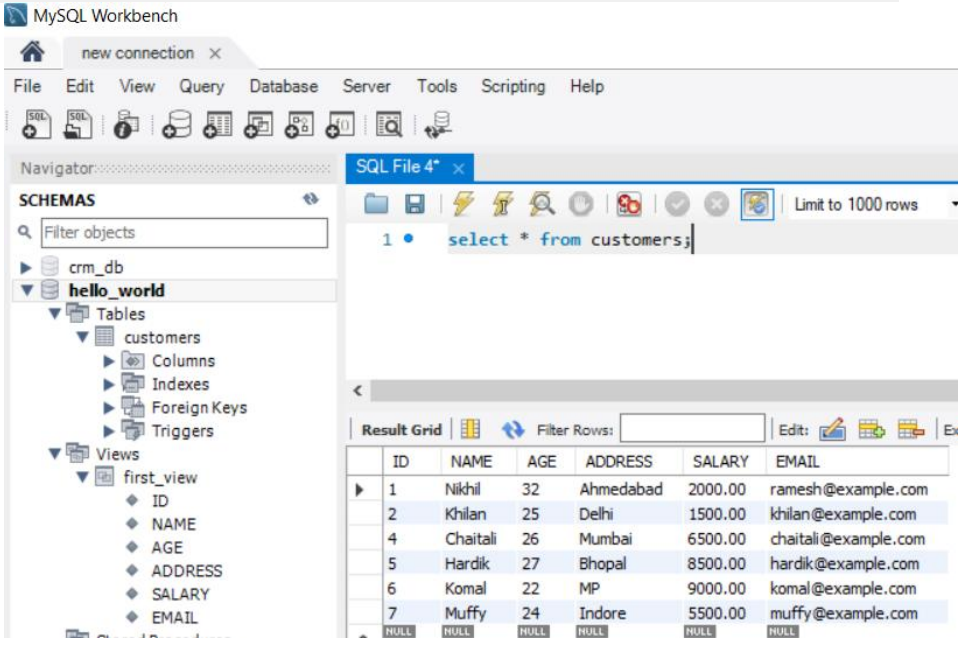


Alter Query

- Use the upadte SQL command to add email data to each record

```
UPDATE CUSTOMERS SET EMAIL = 'ramesh@example.com' WHERE ID = 1;
UPDATE CUSTOMERS SET EMAIL = 'khilan@example.com' WHERE ID = 2;
UPDATE CUSTOMERS SET EMAIL = 'kaushik@example.com' WHERE ID = 3;
UPDATE CUSTOMERS SET EMAIL = 'chaitali@example.com' WHERE ID = 4;
UPDATE CUSTOMERS SET EMAIL = 'hardik@example.com' WHERE ID = 5;
UPDATE CUSTOMERS SET EMAIL = 'komal@example.com' WHERE ID = 6;
UPDATE CUSTOMERS SET EMAIL = 'muffy@example.com' WHERE ID = 7;
```

```
mysql> UPDATE CUSTOMERS SET EMAIL = 'muffy@example.com' WHERE ID = 7;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from customers;
+----+----------+-----+-----------+---------+---------------------+
| ID | NAME     | AGE | ADDRESS   | SALARY  | EMAIL               |
+----+----------+-----+-----------+---------+---------------------+
|  1 | Nikhil   |  32 | Ahmedabad | 2000.00 | ramesh@example.com  |
|  2 | Khilan   |  25 | Delhi     | 1500.00 | khilan@example.com  |
|  4 | Chaitali |  26 | Mumbai    | 6500.00 | chaitali@example.com |
|  5 | Hardik   |  27 | Bhopal    | 8500.00 | hardik@example.com  |
|  6 | Komal    |  22 | MP        | 9000.00 | komal@example.com   |
|  7 | Muffy    |  24 | Indore    | 5500.00 | muffy@example.com   |
+----+----------+-----+-----------+---------+---------------------+
```



Delete Query

- The Delete query in MySQL can be used to **delete existing records in a specified table**

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

```
DELETE FROM CUSTOMERS WHERE ID = 3;
```

ID	NAME	AGE	ADDRESS	SALARY	EMAIL
1	Nikhil	32	Ahmedabad	2000.00	ramesh@example.com
2	Khilan	25	Delhi	1500.00	khilan@example.com
3	Kaushik	23	Kota	2500.00	kaushik@example.com
4	Chaitali	26	Mumbai	6500.00	chaitali@example.com
5	Hardik	27	Bhopal	8500.00	hardik@example.com
6	Komal	22	MP	9000.00	komal@example.com
7	Muffy	24	Indore	5500.00	muffy@example.com

deleted

ID	NAME	AGE	ADDRESS	SALARY	EMAIL
1	Nikhil	32	Ahmedabad	2000.00	ramesh@example.com
2	Khilan	25	Delhi	1500.00	khilan@example.com
4	Chaitali	26	Mumbai	6500.00	chaitali@example.com
5	Hardik	27	Bhopal	8500.00	hardik@example.com
6	Komal	22	MP	9000.00	komal@example.com
7	Muffy	24	Indore	5500.00	muffy@example.com

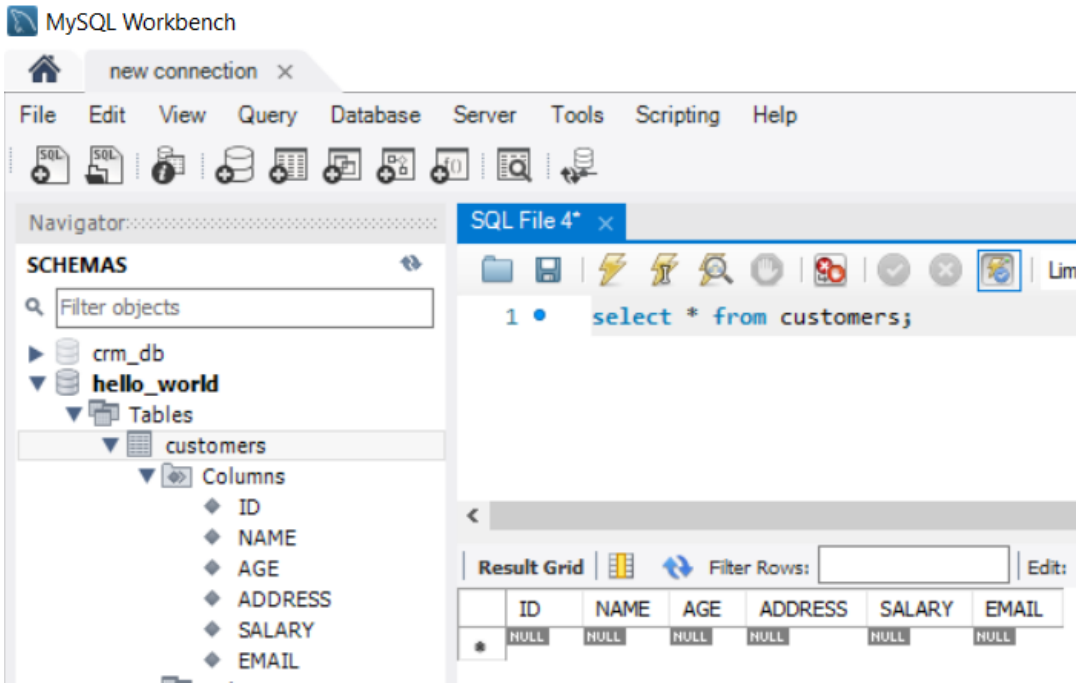
Truncate Query

- The MySQL truncate table query can be used to **remove all the records** but not the table itself

```
TRUNCATE [TABLE] table_name;
```

```
TRUNCATE TABLE CUSTOMERS;
```

```
mysql> truncate table customers;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from customers;
Empty set (0.00 sec)
```



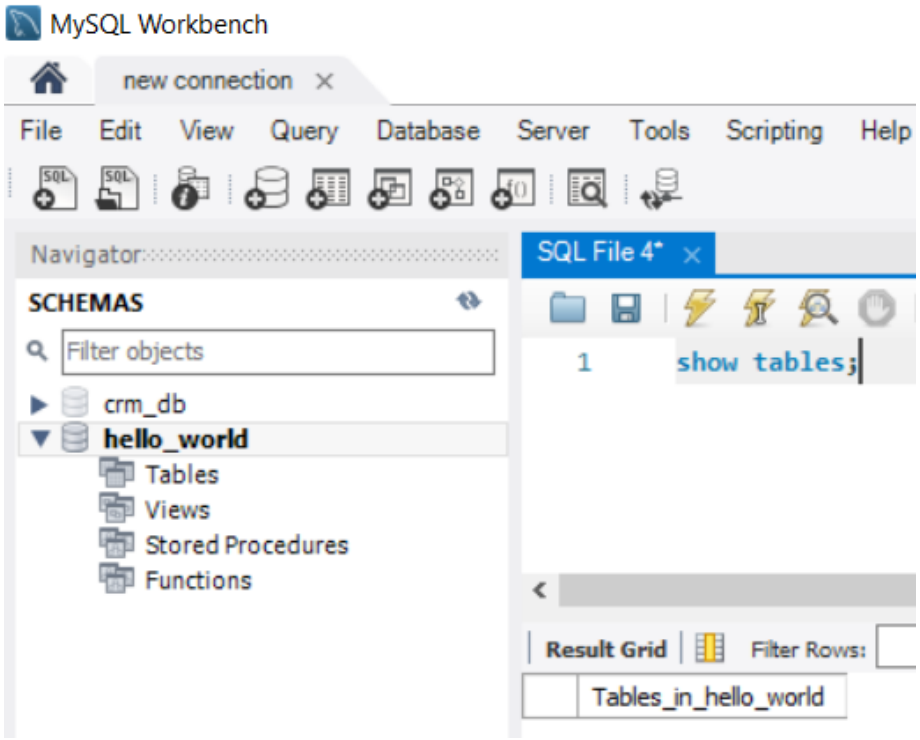
Drop Query

- The MySQL drop query is used to **delete an existing table** in a database

```
DROP TABLE table_name;
```

```
DROP TABLE CUSTOMERS;
```

```
mysql> drop table customers;
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
Empty set (0.00 sec)

mysql>
```



Constraints

- The MySQL constraints can be used to set certain **rules to the column(s) in a table**
- These constraints can **restrict the type of data that can be inserted** or updated in a particular column

- There are two types of MySQL constraints:
  - **column level constraints:** These type of constraints will only apply to a column in a table
  - **table level constraints:** These constraints will apply to the complete table

```
CREATE TABLE table_name (
   Column_name1 datatype constraint,
   Column_name2 datatype constraint,
   Column_name3 datatype constraint,
   .........
);
```

Constraints

- **NOT NULL**: cannot insert or update a record without adding a value
- **UNIQUE**: every value in a column must be distinct
- **PRIMARY KEY**: uniquely identify each record in a table. Can be define primary key on a particular column in a table, it must contain UNIQUE values, and cannot contain NULL values
- **FOREIGN KEY**: used to link a field or collection of fields in one table to the primary key of another table. A table with the foreign key is called a child table and the table with the primary key is called the parent table or referenced table

Constraints

- **CHECK**: restricts the range of values that can be inserted into a column
- **DEFAULT:** used to assign a default value to a specific column in a table
- **CREATE INDEX**: used to create indexes for one more columns in a table
- **AUTO_INCREMENT**: defined on a particular column of a table, it will automatically generate a unique number when a new record is inserted into that column

View

- MySQL views are a type of **virtual tables**
- They are stored in the database with an associated name
- Structure data in a way that users or classes of users find natural or intuitive
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more
- **Summarize data from various tables which can be used to generate reports**

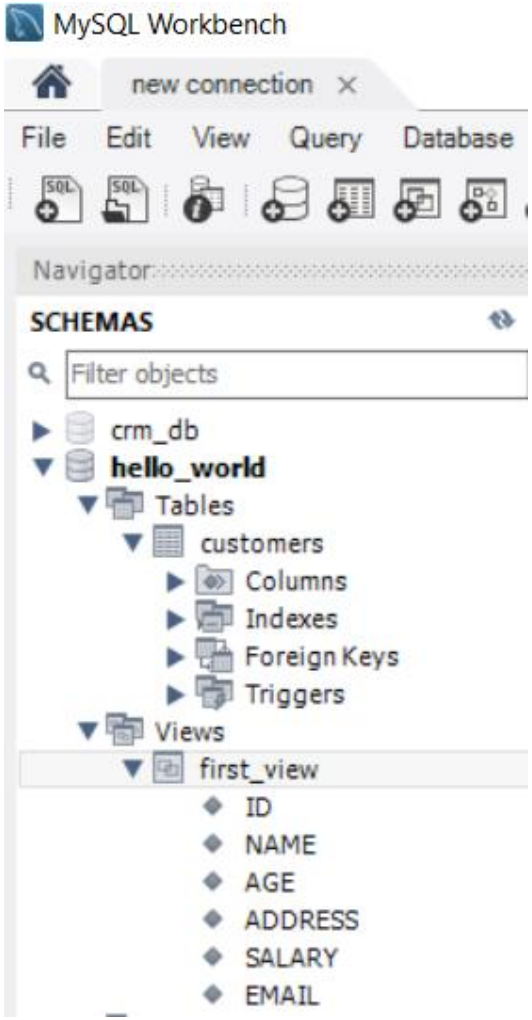
```
CREATE VIEW view_name AS select_statements FROM table_name;
```

```
CREATE VIEW first_view AS SELECT * FROM CUSTOMERS;
```

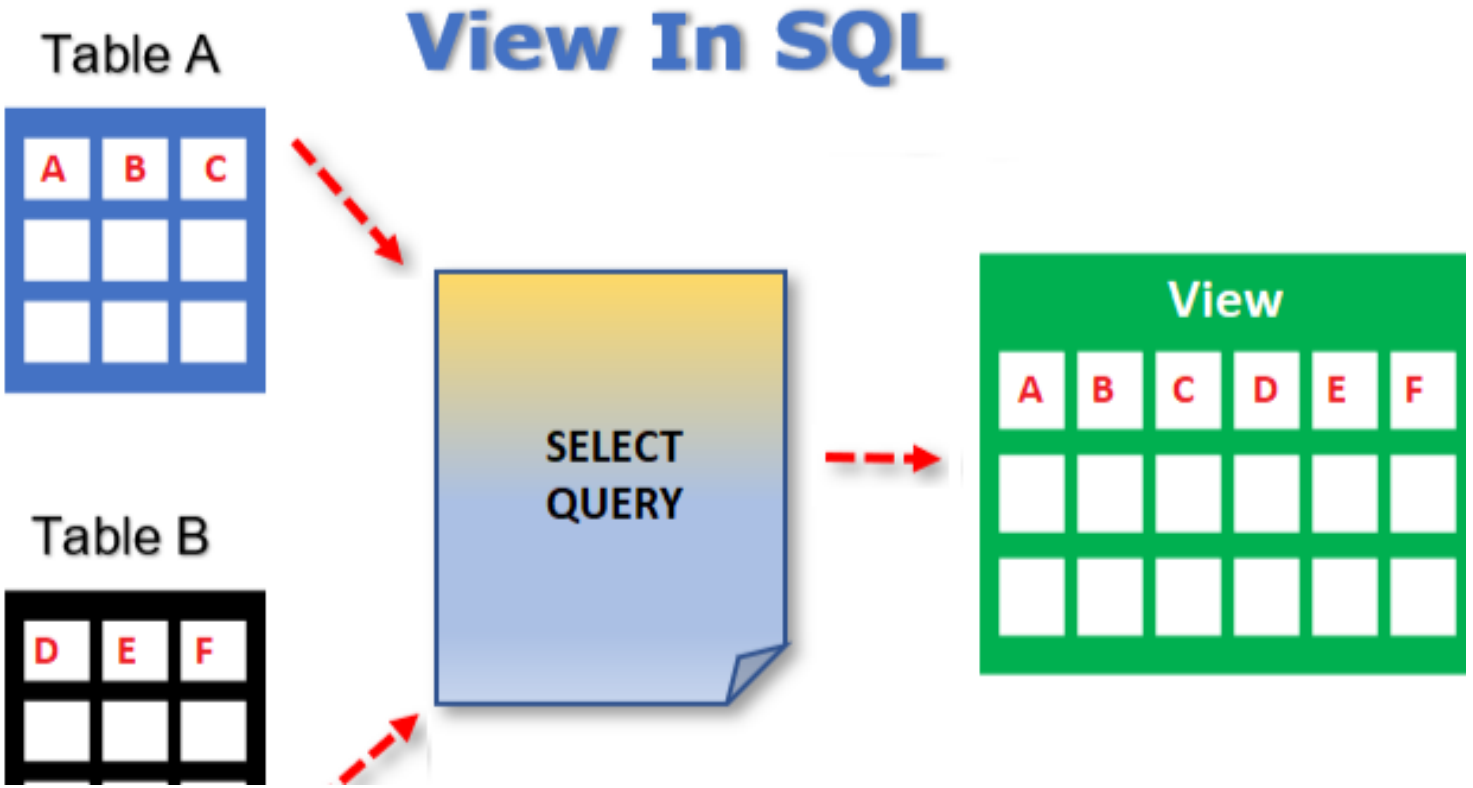
View

```
mysql> CREATE VIEW first_view AS SELECT * FROM CUSTOMERS;
Query OK, 0 rows affected (0.01 sec)

mysql> _
```

```
mysql> SELECT * FROM first_view;
+----+----------+-----+-----------+---------+----------------------+
| ID | NAME     | AGE | ADDRESS   | SALARY  | EMAIL                |
+----+----------+-----+-----------+---------+----------------------+
|  1 | Nikhil   |  32 | Ahmedabad | 2000.00 | ramesh@example.com   |
|  2 | Khilan   |  25 | Delhi     | 1500.00 | khilan@example.com   |
|  4 | Chaitali |  26 | Mumbai    | 6500.00 | chaitali@example.com |
|  5 | Hardik   |  27 | Bhopal    | 8500.00 | hardik@example.com   |
|  6 | Komal    |  22 | MP        | 9000.00 | komal@example.com    |
|  7 | Muffy    |  24 | Indore    | 5500.00 | muffy@example.com    |
+----+----------+-----+-----------+---------+----------------------+
6 rows in set (0.00 sec)
```



View



View

- Create a 2. table:

```
CREATE TABLE ORDERS (
    ORDER_ID INT AUTO_INCREMENT,
    CUSTOMER_ID INT,
    AMOUNT DECIMAL(18, 2),
    ORDER_DATE DATE,
    PRIMARY KEY (ORDER_ID),
    FOREIGN KEY (CUSTOMER_ID)
    REFERENCES CUSTOMERS(ID)
);
```

- Add records to the `ORDERS` table:

```
INSERT INTO ORDERS (CUSTOMER_ID, AMOUNT, ORDER_DATE) VALUES
(1, 500, '2024-01-15'),
(3, 1500, '2024-02-10'),
(5, 2000, '2024-03-05'),
(2, 800, '2024-04-20'),
(4, 2200, '2024-05-17');
```

View

- Customers table content:

ID	NAME	AGE	ADDRESS	SALARY	EMAIL
1	Nikhil	32	Ahmedabad	2000.00	ramesh@example.com
2	Khilan	25	Delhi	1500.00	khilan@example.com
3	Kaushik	23	Kota	2500.00	kaushik@example.com
4	Chaitali	26	Mumbai	6500.00	chaitali@example.com
5	Hardik	27	Bhopal	8500.00	hardik@example.com
6	Komal	22	MP	9000.00	komal@example.com
7	Muffy	24	Indore	5500.00	muffy@example.com

- Orders table content:

ORDER_ID	CUSTOMER_ID	AMOUNT	ORDER_DATE
1	1	500.00	2024-01-15
2	3	1500.00	2024-02-10
3	5	2000.00	2024-03-05
4	2	800.00	2024-04-20
5	4	2200.00	2024-05-17

View

- Create a new view from the `CUSTOMERS` and the `ORDERS` tables:

```
CREATE VIEW customer_orders_view AS SELECT
   c.ID AS CustomerID,
   c.NAME AS CustomerName,
   c.AGE AS CustomerAge,
   c.ADDRESS AS CustomerAddress,
   c.SALARY AS CustomerSalary,
   o.ORDER_ID AS OrderID,
   o.AMOUNT AS OrderAmount,
   o.ORDER_DATE AS OrderDate
FROM
   CUSTOMERS c
LEFT JOIN
   ORDERS o ON c.ID = o.CUSTOMER_ID;
```

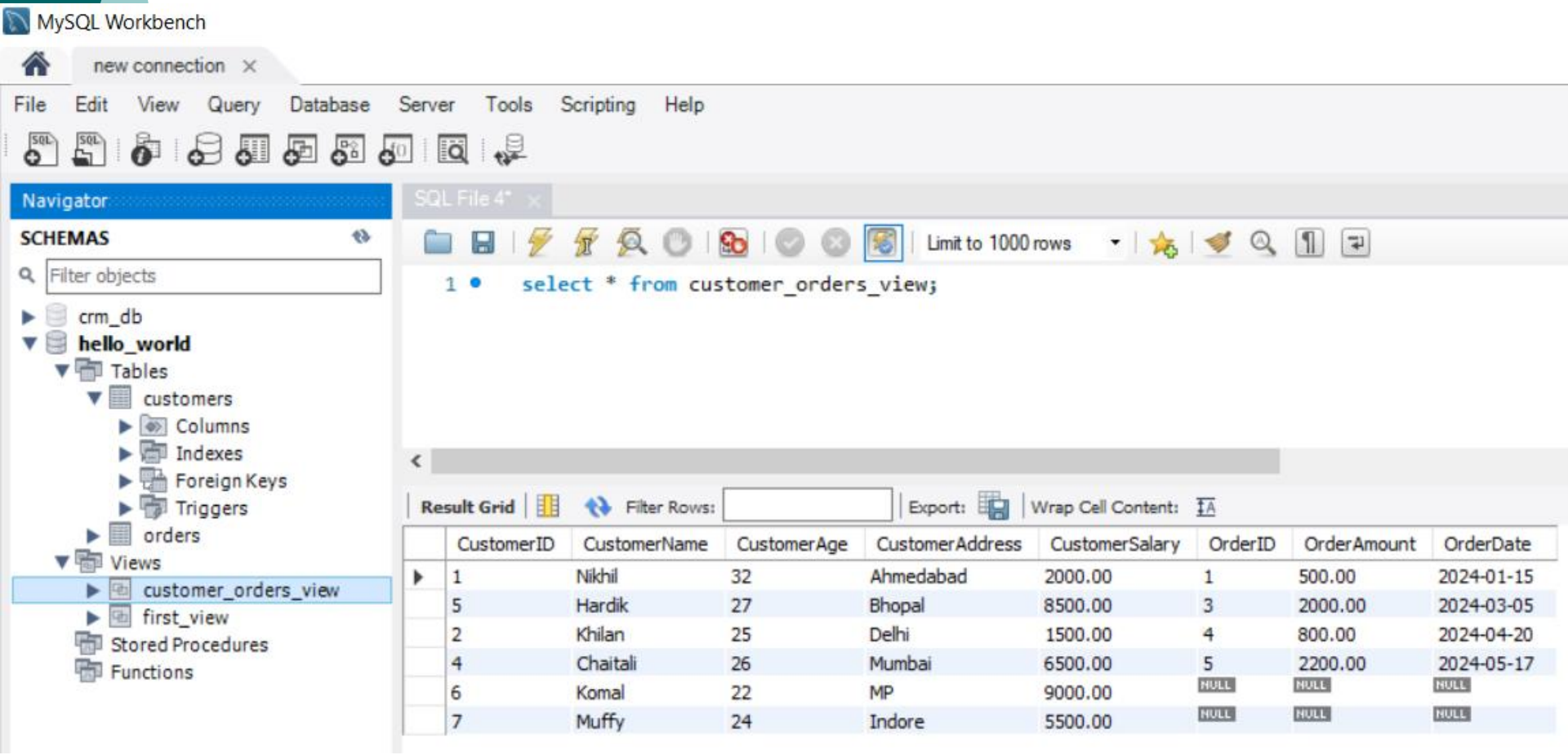
View

- The select query for the view (in mysql cmd):

```
mysql> select * from customer_orders_view;
+------------+--------------+-------------+-----------------+----------------+---------+-------------+------------+
| CustomerID | CustomerName | CustomerAge | CustomerAddress | CustomerSalary | OrderID | OrderAmount | OrderDate  |
+------------+--------------+-------------+-----------------+----------------+---------+-------------+------------+
|          1 | Nikhil       |          32 | Ahmedabad       |        2000.00 |       1 |      500.00 | 2024-01-15 |
|          5 | Hardik       |          27 | Bhopal          |        8500.00 |       3 |     2000.00 | 2024-03-05 |
|          2 | Khilan       |          25 | Delhi           |        1500.00 |       4 |      800.00 | 2024-04-20 |
|          4 | Chaitali     |          26 | Mumbai          |        6500.00 |       5 |     2200.00 | 2024-05-17 |
|          6 | Komal        |          22 | MP              |        9000.00 |    NULL |        NULL | NULL       |
|          7 | Muffy        |          24 | Indore          |        5500.00 |    NULL |        NULL | NULL       |
+------------+--------------+-------------+-----------------+----------------+---------+-------------+------------+
6 rows in set (0.00 sec)
```

View

- The select query for the view (in MySQL Workbench):

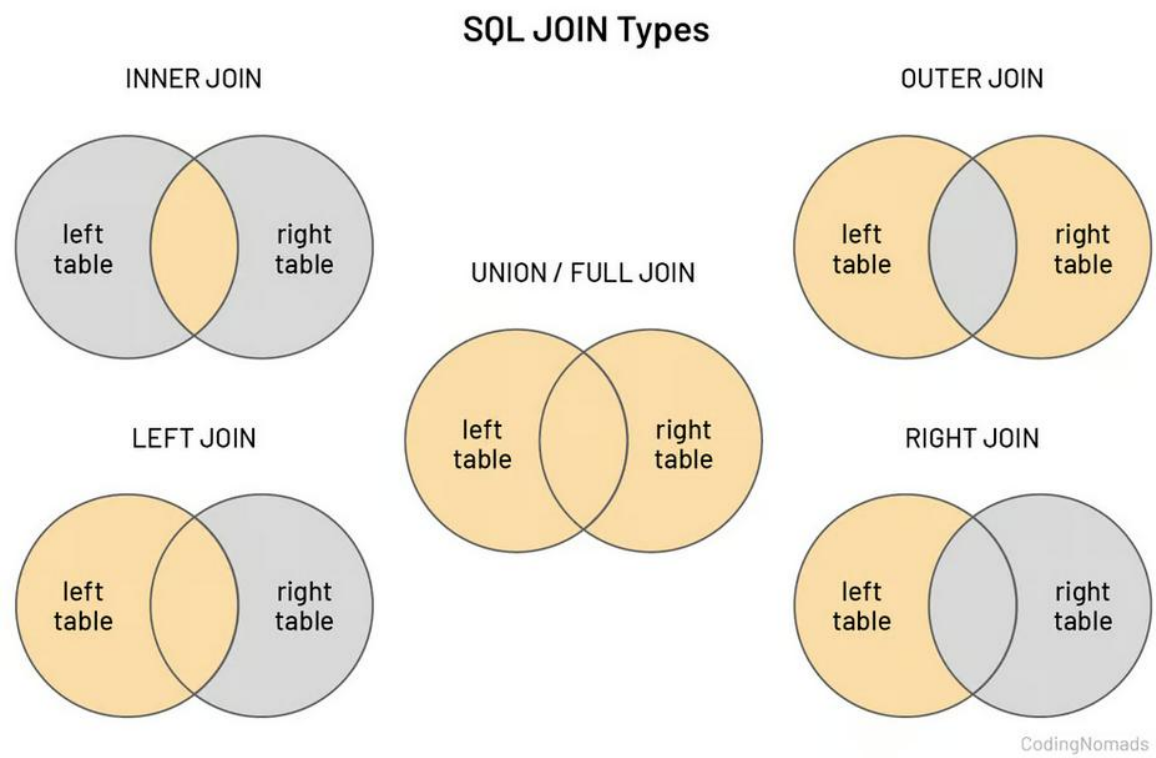


Join

- A Join clause in MySQL is used to **combine records from two or more tables in a database**
- These tables are joined together based on a condition, specified in a **WHERE clause**
  - **Inner Join:** An Inner Join retrieves the intersection of two tables. It compares each row of the first table with each row of the second table. If the pairs of these rows satisfy the join-predicate, they are joined together. This is a default join.
  - **Outer Join**: An Outer Join retrieves all the records in two tables even if there is no counterpart row of one table in another table, like Inner Join. Outer join is further divided into three subtypes: Left Join, Right Join and Full Join. We will learn about these Joins later in this tutorial.

```
SELECT a.ID, a.NAME, b.DATE, b.AMOUNT
FROM CUSTOMERS a, ORDERS b
WHERE a.ID = b.CUSTOMER_ID;
```

Join


SQL JOIN Types
INNER JOIN
left table
right table
OUTER JOIN
left table
right table
UNION / FULL JOIN
left table
right table
LEFT JOIN
left table
right table
RIGHT JOIN
left table
right table
CodingNomads


A
B
SELECT *
FROM A
INNER JOIN B ON A.key = B.key
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
WHERE A.key IS NULL OR
B.key IS NULL
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
WHERE B.key IS NULL
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
WHERE B.key IS NULL

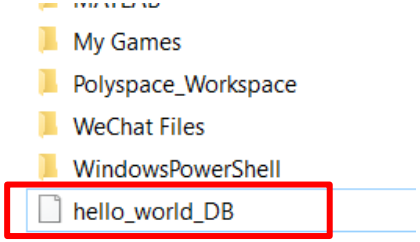
Export database

- **Exporting a database** in MySQL is commonly used for **backup** purposes or **transferring data between servers**
  - you can export entire database or just a portion of it

- The **mysqldump** command-line tool is used in MySQL to create backups of databases
  - it can be used to back up an entire database, specific tables, or even specific rows based of a table

```
mysqldump -u username -p database_name > output_file_path
```

```
mysqldump -u root -p hello_world > hello_world_DB
```



Import database

- We can **import the backup data into an MySQL database** using the mysql command-line tool. It takes the username, database name, and the backup file with the data

```
mysql -u username -p new_database_name < dumpfile_path
```

```
mysql -u root -p hello_world_imported < hello_world_DB
```

```
C:\Users\Tompa_Tamas\Documents>mysql -u root -p hello_world_imported < hello_world_DB
Enter password: ****
```

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| crm_db             |
| hello_world        |
| hello_world_imported |
| mysql              |
| performance_schema |
| user_management    |
+--------------------+
```

```
mysql> use hello_world_imported;
Database changed
mysql> show tables;
+-------------------------------+
| Tables_in_hello_world_imported |
+-------------------------------+
| customer_orders_view          |
| customers                     |
| first_view                    |
| orders                        |
+-------------------------------+
```

More MySQL example

- https://www3.ntu.edu.sg/home/ehchua/programming/sql/MySQL_Beginner.html
- https://www.tutorialspoint.com/mysql/index.htm
- https://www.w3schools.com/MySQL/default.asp

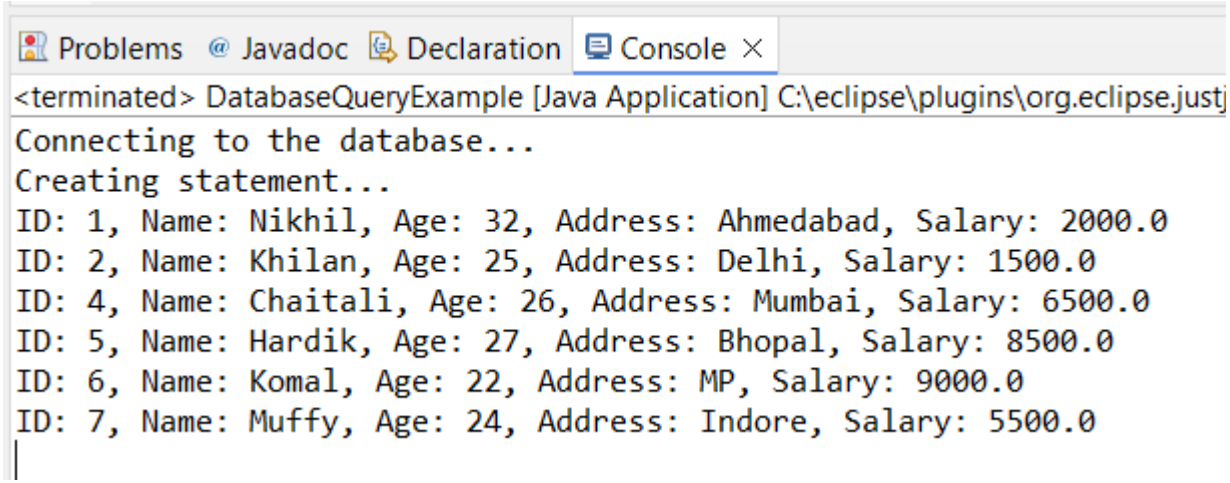
JDBC - query

- Query for the (**previously created Customers**) table

```java
package mysql_query;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class DatabaseQueryExample {

    // Database URL, username, and password
    static final String DB_URL = "jdbc:mysql://localhost:3306/hello_world";
    static final String USER = "root";
    static final String PASS = "root";

    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;

        try {
            // Step 1: Register JDBC driver (optional for newer JDBC versions)
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Step 2: Open a connection
            System.out.println("Connecting to the database...");
            conn = DriverManager.getConnection(DB_URL, USER, PASS);

            // Step 3: Execute a query
            System.out.println("Creating statement...");
            stmt = conn.createStatement();
            String sql = "SELECT ID, NAME, AGE, ADDRESS, SALARY FROM CUSTOMERS";
            ResultSet rs = stmt.executeQuery(sql);

            // Step 4: Process the ResultSet
            while (rs.next()) {
                // Retrieve data by column name
                int id = rs.getInt("ID");
                String name = rs.getString("NAME");
                int age = rs.getInt("AGE");
                String address = rs.getString("ADDRESS");
                double salary = rs.getDouble("SALARY");

                // Display values
                System.out.print("ID: " + id);
                System.out.print(", Name: " + name);
                System.out.print(", Age: " + age);
                System.out.print(", Address: " + address);
                System.out.println(", Salary: " + salary);
            }

            // Step 5: Clean up the environment
            rs.close();
            stmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // Finally block to close resources
            try {
                if (stmt != null) stmt.close();
                if (conn != null) conn.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

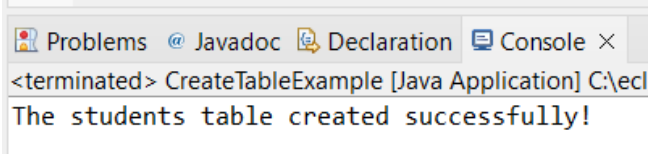
JDBC - query

- Query for the (**previously created Customers**) Customers table



JDBC – create table

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class CreateTableExample {
    public static void main(String[] args) {
        // JDBC URL, felhasználónév és jelszó
        String url = "jdbc:mysql://localhost:3306/hello_world";
        String user = "root";
        String password = "root";

        // Kapcsolat és Statement objektum
        Connection connection = null;
        Statement statement = null;

        try {
            connection = DriverManager.getConnection(url, user, password);
            statement = connection.createStatement();

            String sql = "CREATE TABLE students (" + "id INT AUTO_INCREMENT PRIMARY KEY,"
                    + "name VARCHAR(100) NOT NULL," + "age INT NOT NULL" + ")";

            statement.executeUpdate(sql);
            System.out.println("The students table created successfully!");

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (statement != null) {
                    statement.close();
                }
                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException ex) {
                ex.printStackTrace();
```



- Create of students table in the hello_world database

JDBC - insert record


```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class InsertStudentsExample {
    private static final String URL = "jdbc:mysql://localhost:3306/hello_world";
    private static final String USER = "root";
    private static final String PASSWORD = "root";

    public static void main(String[] args) {
        InsertStudentsExample example = new InsertStudentsExample();
        example.insertStudent("John Doe", 20);
        example.insertStudent("Jane Smith", 22);
        example.insertStudent("Emily Johnson", 19);
    }

    public void insertStudent(String name, int age) {
        String sql = "INSERT INTO students (name, age) VALUES (?, ?)";

        try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

            preparedStatement.setString(1, name);
            preparedStatement.setInt(2, age);

            int rowsAffected = preparedStatement.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Record inserted successfully: " + name + ", age: " + age);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Problems | Javadoc | Declaration | Console

```
<terminated> InsertStudentsExample [Java Application] C:\eclipse\plugi
Record inserted successfully: John Doe, age: 20
Record inserted successfully: Jane Smith, age: 22
Record inserted successfully: Emily Johnson, age: 19
```

```
mysql> select * from students;
+----+--------------+-----+
| id | name         | age |
+----+--------------+-----+
|  1 | John Doe     |  20 |
|  2 | Jane Smith   |  22 |
|  3 | Emily Johnson |  19 |
+----+--------------+-----+
3 rows in set (0.00 sec)
```

- Inserting records in the Students table

JDBC – update record

```java
public class UpdateStudentExample {
    private static final String URL = "jdbc:mysql://localhost:3306/hello_world";
    private static final String USER = "root";
    private static final String PASSWORD = "root";

    public static void main(String[] args) {
        UpdateStudentExample example = new UpdateStudentExample();
        example.updateStudentAge(1, 25);
        example.updateStudentName(2, "Michael Brown");
    }

    // age update
    public void updateStudentAge(int id, int newAge) {
        String sql = "UPDATE students SET age = ? WHERE id = ?";

        try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

            preparedStatement.setInt(1, newAge);
            preparedStatement.setInt(2, id);

            int rowsAffected = preparedStatement.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Successfully updated the record with ID: " + id + " New age: " + newAge);
            } else {
                System.out.println("The record doens't exist with this ID.");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

```
mysql> select * from students;
+----+--------------+-----+
| id | name         | age |
+----+--------------+-----+
|  1 | John Doe     |  25 |
|  2 | Michael Brown |  22 |
|  3 | Emily Johnson |  19 |
|  4 | John Doe     |  20 |
|  5 | Jane Smith   |  22 |
|  6 | Emily Johnson |  19 |
+----+--------------+-----+
6 rows in set (0.00 sec)
```

```
Problems  Javadoc  Declaration  Console ×
<terminated> UpdateStudentExample [Java Application] C:\eclipse\plugins\org.eclipse.just
Successfully updated the record with ID: 1 New age: 25
Successfully updated the record with ID: 2 New name: Michael Brown
```

- Updating record (with specified ID) in the Students table

JDBC – delete record

```java
public class DeleteStudentExample {
    private static final String URL = "jdbc:mysql://localhost:3306/hello_world";
    private static final String USER = "root";
    private static final String PASSWORD = "root";

    public static void main(String[] args) {
        DeleteStudentExample example = new DeleteStudentExample();
        example.deleteStudentById(1);
    }

    // delete record by ID
    public void deleteStudentById(int id) {
        String sql = "DELETE FROM students WHERE id = ?";

        try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD);
                PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

            preparedStatement.setInt(1, id);

            int rowsAffected = preparedStatement.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Successfully deleted the record with ID: " + id);
            } else {
                System.out.println("The record with this ID doens't exist.");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

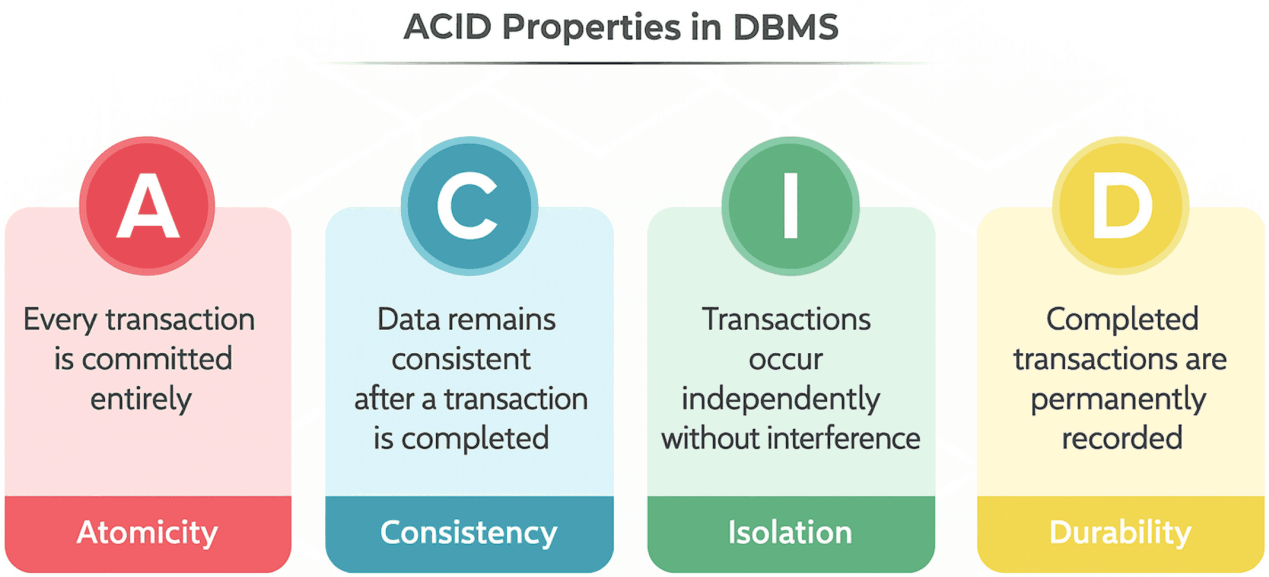
```
mysql> select * from students;
+----+---------------+-----+
| id | name          | age |
+----+---------------+-----+
|  2 | Michael Brown |  22 |
|  3 | Emily Johnson |  19 |
|  4 | John Doe      |  20 |
|  5 | Jane Smith    |  22 |
|  6 | Emily Johnson |  19 |
+----+---------------+-----+
5 rows in set (0.00 sec)
```

```
Problems  @ Javadoc  Declaration  Console ×
<terminated> DeleteStudentExample [Java Application] C:\
Successfully deleted the record with ID: 1
```

- Deleting record (with specified ID) from the Students table

Transactions in JDBC

- **Transactions play a vital role in maintaining data integrity by ensuring ACID properties**
  - atomicity, consistency, isolation, durability



Transactions in JDBC

- **Auto-Commit in JDBC**
  - every SQL statement is committed to the database upon its completion
  - but 3 reasons for turning off auto-commit and managing transactions manually
    - increase performance
    - maintain the integrity
    - use distributed transactions

- **Can be managed transactions manually by turning off auto-commit and using `commit()` and `rollback()`**
  - in Java, the `commit()` and `rollback()` methods play an essential role in database transactions, ensuring ACID properties

Transactions in JDBC

- `commit()`
  - **saves all changes** made during the transaction to the database
- `rollback()`
  - **discards all changes** made since the last commit, ensuring atomicity

```
Connection conn = DriverManager.getConnection(url, user, password);
conn.setAutoCommit(false); // Disable auto-commit
// Execute multiple SQL statements using PreparedStatement
conn.commit();   // Commit all changes if successful
conn.rollback(); // Rollback if any error occurs
```

Thank you for your attention!