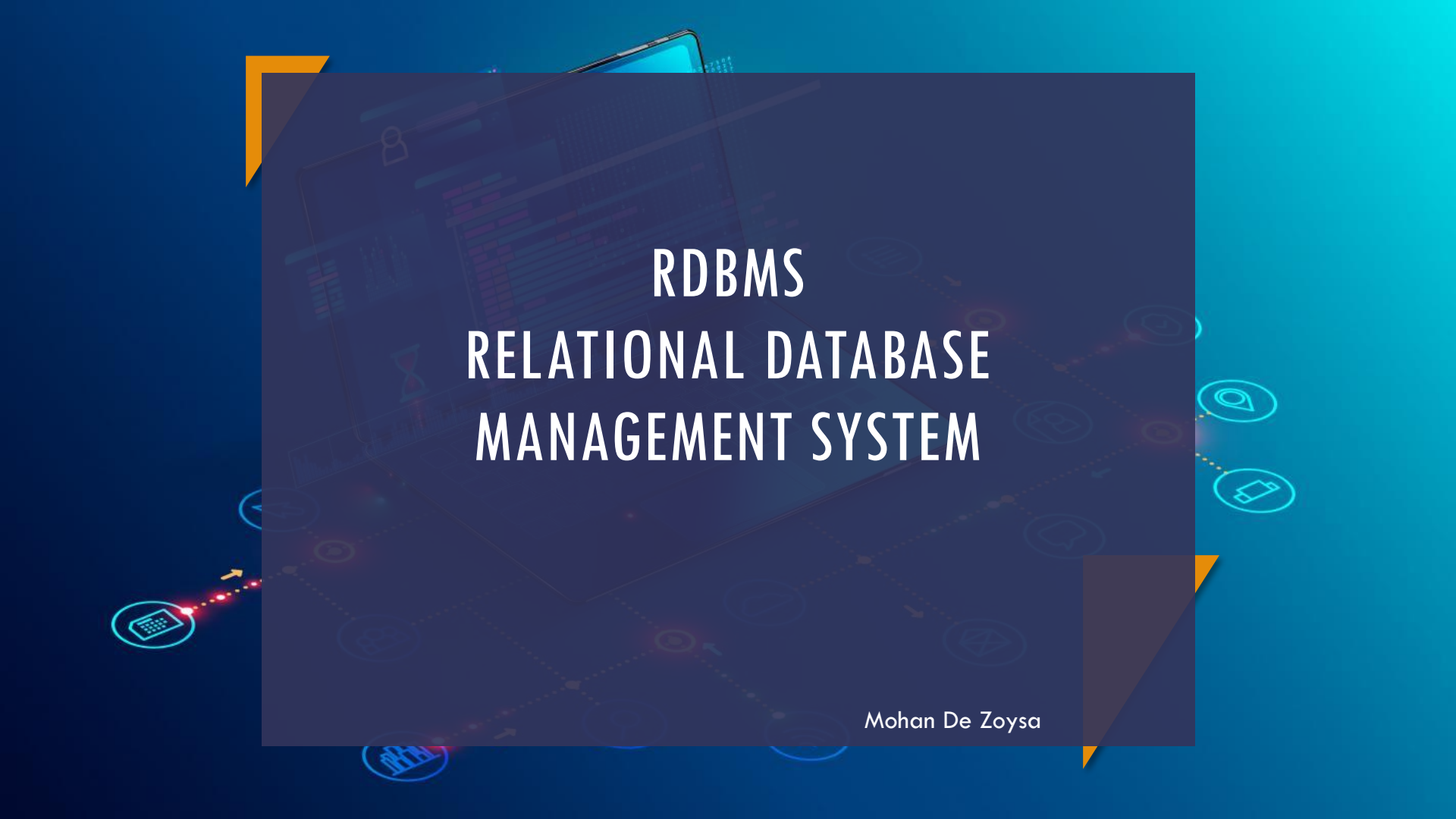


RDBMS
RELATIONAL DATABASE MANAGEMENT SYSTEM

Mohan De Zoysa

INTRODUCTION

RDBMS

A Relational Database Management System (RDBMS) is a software system designed to manage and organize structured data using a relational model. It provides an efficient and organized way to store, retrieve, update, and manage data in a tabular format. The key feature of an RDBMS is its ability to define relationships between tables, which allows for complex data modeling and querying. the complicated things in our world.

Examples

- **MySQL**
- **Oracle Database**
- **Microsoft SQL Server**
- **PostgreSQL**
- **SQLite**
- **IBM Db2**

INTRODUCTION

Concepts and components associated with RDBMS

Tables - Data in an RDBMS is stored in tables, which are structured as rows and columns. Each row represents a record, and each column represents a field or attribute of the record. Each table has a name and a predefined schema that specifies the data types and constraints for each column.

Table: Authors

AuthorID	FirstName	LastName
1	John	Doe
2	Jane	Smith
3	David	Johnson

Table: Books

BookID	Title	AuthorID (FK)
101	Introduction to Programming	1
102	Data Structures and Algorithms	1
103	History of Science	2
104	Database Design	3

INTRODUCTION

Concepts and components associated with RDBMS

Keys - RDBMS uses keys to establish relationships between tables and ensure data integrity.

- **Primary Key (PK):** A unique identifier for each row in a table. It ensures that each row can be uniquely identified.
- **Foreign Key (FK):** A field in one table that refers to the primary key in another table. It establishes relationships between tables.

Table: Authors

AuthorID (PK)	FirstName	LastName
1	John	Doe
2	Jane	Smith
3	David	Johnson

Table: Books

BookID	Title	AuthorID (FK)
101	Introduction to Programming	1
102	Data Structures and Algorithms	1
103	History of Science	2
104	Database Design	3

SQL STRUCTURED QUERY LANGUAGE

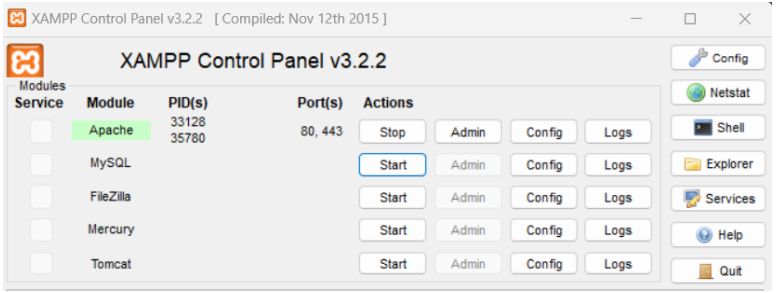
If someone asks to you, what is time now? So, seeing time in your watch and telling him is also a kind of problem solving

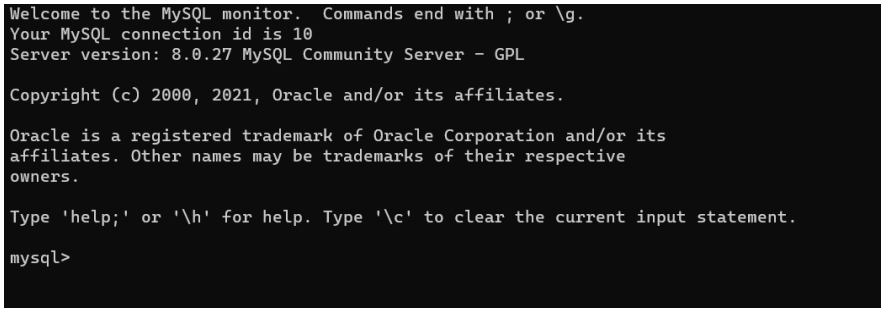
SQL is the standard language used to interact with RDBMS. It provides a set of commands for defining, manipulating, and querying the database

PRACTICAL'S WE ARE USING MYSQL RDBMS

Installations Option one

Install XAMP software on the PC & Run the MySQL service





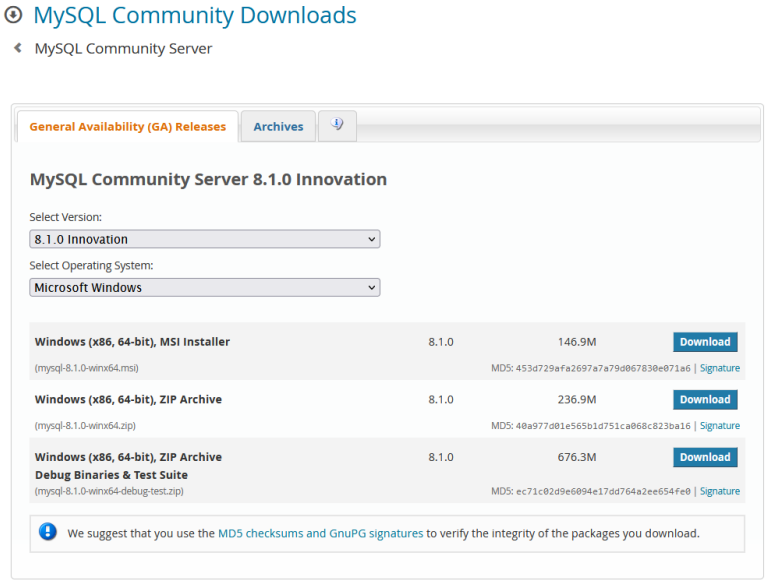
Run the command form MS Dos - CMD

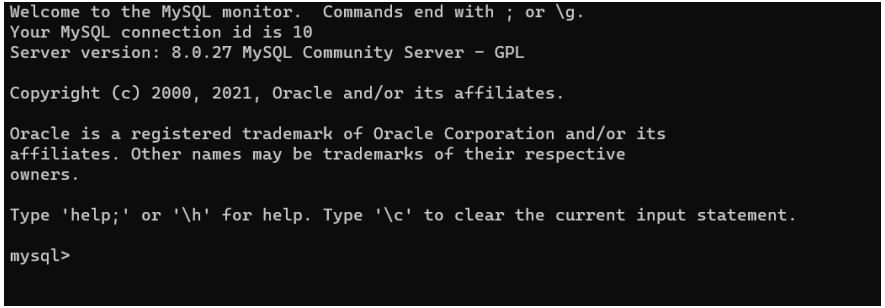
```
C:\Users\Mohan>mysql -u UserName -p Password
```

PRACTICAL'S WE ARE USING MYSQL RDBMS

Installations Option Two

Install Mysql Community Server software on the pc

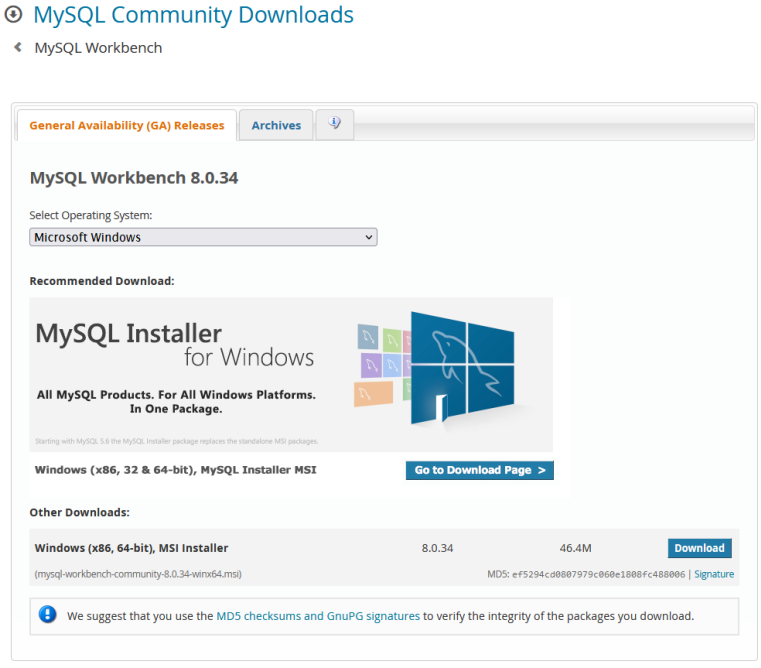




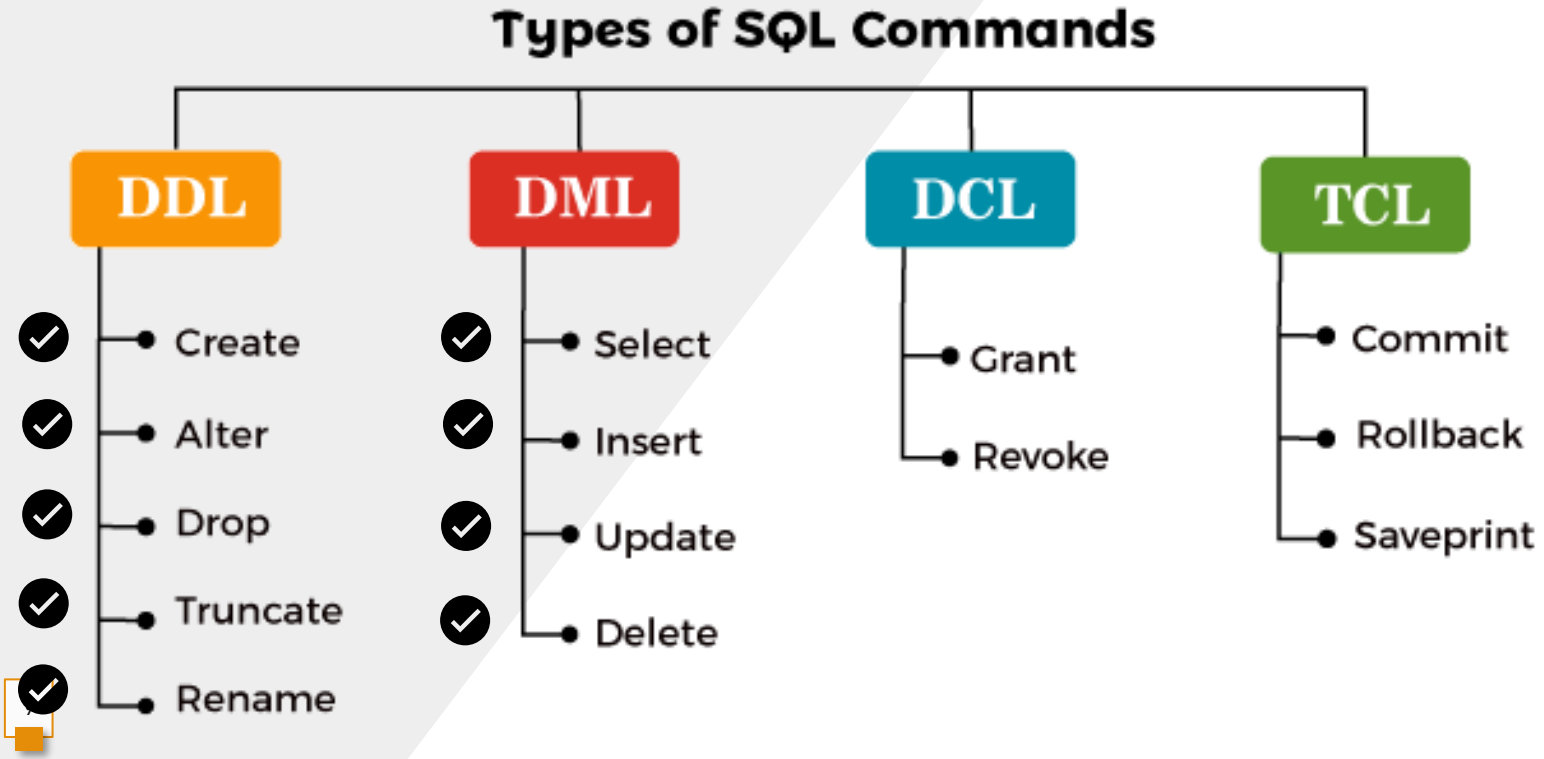
PRACTICAL'S WE ARE USING MYSQL RDBMS

MYSQL work bench or command line tool can be used to run the sql commands

https://dev.mysql.com/downloads/workbench/



TYPE OF SQL STATEMENTS



DDL – DATA DEFINITION LANGUAGE

DDL stands for Data Definition Language. It is a subset of SQL commands used to define, modify, and manage the structure of database objects. DDL commands are responsible for creating, altering, and dropping database schemas, tables, indexes, constraints, and other structural components. Here are some common DDL commands:

DDL CONTINUE

CREATE DATABASE

Syntax - CREATE DATABASE *databasename*;

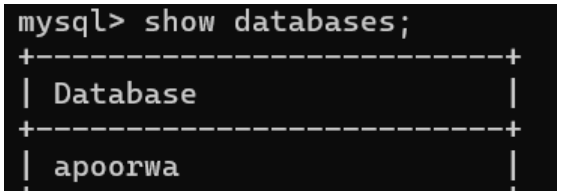
Example - CREATE DATABASE IIT;

Syntax - DROP DATABASE *databasename*;

Example - DROP DATABASE IIT;

To view data bases created use show database command

mysql> show databse;



To use a data base mysql> use {data base name}; Ex : - use IIT;

DDL CONTINUE

CREATE TABLE

The CREATE TABLE statement is used to create a new table in a database.

```
Syntax - CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

The column parameters specify the names of the columns of the table.
The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

describe persons;

To view the table structure, use the describe command

DDL CONTINUE

CREATE TABLE

The CREATE TABLE statement is used to create a new table in a database.

```
Syntax - CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

The column parameters specify the names of the columns of the table.
The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

DDL CONTINUE

ALTER TABLE

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.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

The following SQL adds an "Email" column to the " Persons " table:

```
ALTER TABLE Persons
ADD address varchar(255);
```

DDL CONTINUE

ALTER TABLE

The ALTER TABLE Rename table name

ALTER TABLE – Rename to

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
rename to new_name;
```

The following SQL adds an "Email" column to the " Persons " table:

```
ALTER TABLE Persons
rename to  new_person;
```

DDL CONTINUE

ALTER TABLE

ALTER TABLE - DROP Column

To add a column in a table, use the following syntax:

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

The following SQL Drop an "Email" column to the " Persons " table:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

DDL CONTINUE

ALTER TABLE

ALTER TABLE - RENAME COLUMN

To rename a column in a table, use the following syntax:

```
ALTER TABLE table_name
RENAME COLUMN old_name to new_name;
```

The following SQL rename an “Address" column to the *home_* address in “Persons " table:

```
ALTER TABLE Persons
RENAME COLUMN Address to home_ address;
```

DDL CONTINUE

ALTER TABLE

ALTER TABLE – ADD PRIMARY KEY

To rename a column in a table, use the following syntax:

ALTER TABLE *table_name*
add primary key(*column_name*);

The following SQL add a primary key of “PersonID” to the “Persons " table:

ALTER TABLE *Persons*
add primary key(PersonID);

DDL CONTINUE

ALTER TABLE

ALTER TABLE – ADD FOREIGN KEY

To rename a column in a table, use the following syntax:

ALTER TABLE *table_name*
add foreign key(*column_name*) *references*
reference_table_name(*column_name*);

```
alter table employees add foreign key(dep_id) references
department(dep_id);
```

DDL CONTINUE

DROP TABLE

The DROP TABLE command deletes a table in the database with data.

The following SQL deletes the table "Shippers":

```
DROP TABLE Shippers;
```

DDL CONTINUE

TRUNCATE TABLE

The DROP TABLE command deletes a data only form the table and structure will not be deleted .

The following SQL deletes the table "Shippers":

```
TRUNCATE TABLE Categories;
```

DDL CONTINUE

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted. Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

NOT NULL - Ensures that a column cannot have a NULL value
UNIQUE - Ensures that all values in a column are different
PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
FOREIGN KEY - Prevents actions that would destroy links between tables

DDL CONTINUE

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted. Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

NOT NULL - Ensures that a column cannot have a NULL value
UNIQUE - Ensures that all values in a column are different
PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
FOREIGN KEY - Prevents actions that would destroy links between tables

DDL CONTINUE

Contains

SQL NOT NULL Constraint

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.


```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

DDL CONTINUE

Contains

SQL UNIQUE Constraint

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

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.


```
CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

DDL CONTINUE

Contains

SQL FOREIGN KEY Constraint

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Look at the following two tables

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

DML – DATA DEFINITION LANGUAGE

DML stands for Data Manipulation Language. It is a subset of SQL (Structured Query Language) that deals with the manipulation of data stored in a relational database. DML commands are used to insert, update, delete, and retrieve data from database tables. Here are some common DML commands:

DML CONTINUE

INSERT

Insert into

The SQL INSERT INTO Statement

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

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:


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

DML CONTINUE

INSERT

Insert into

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

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


This method values will be added to the order of the table columns, so order is very important

DML CONTINUE

INSERT

Example

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES
('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway'),
('Greasy Burger', 'Per Olsen', 'Gateveien 15', 'Sandnes', '4306', 'Norway'),
('Tasty Tee', 'Finn Egan', 'Streetroad 19B', 'Liverpool', 'L1 0AA', 'UK');
```

DML CONTINUE

SELECT

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

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

SELECT
FROM
WHERE
Group by
Having
Order by

Selecting specific columns

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

Selecting all columns using *

```
SELECT * FROM table_name;
```

DML CONTINUE

The SQL AND, OR Operators

The WHERE clause can contain one or many AND, OR operator

```
SELECT * FROM Customers
WHERE Country = 'Germany'
AND City = 'Berlin'
AND PostalCode > 1200;
```

```
SELECT * FROM Customers
WHERE Country = 'Germany'
AND City = 'Berlin'
OR PostalCode > 1200;
```

DML CONTINUE

The SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

The percent sign (%) represents zero, one, or multiple characters

The underscore sign (_) represents one, single character


```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

DML CONTINUE

The SQL IN Operator

DML CONTINUE

The SQL UPDATE Statement

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

UPDATE Syntax

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

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

DML CONTINUE

The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

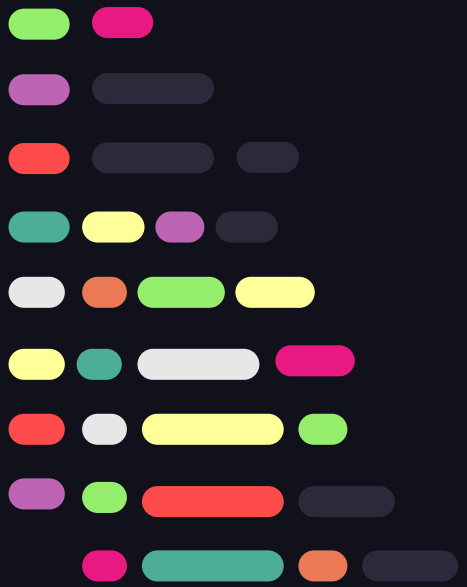
```
DELETE FROM table_name WHERE
condition;
```

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

Java Programming Week-08

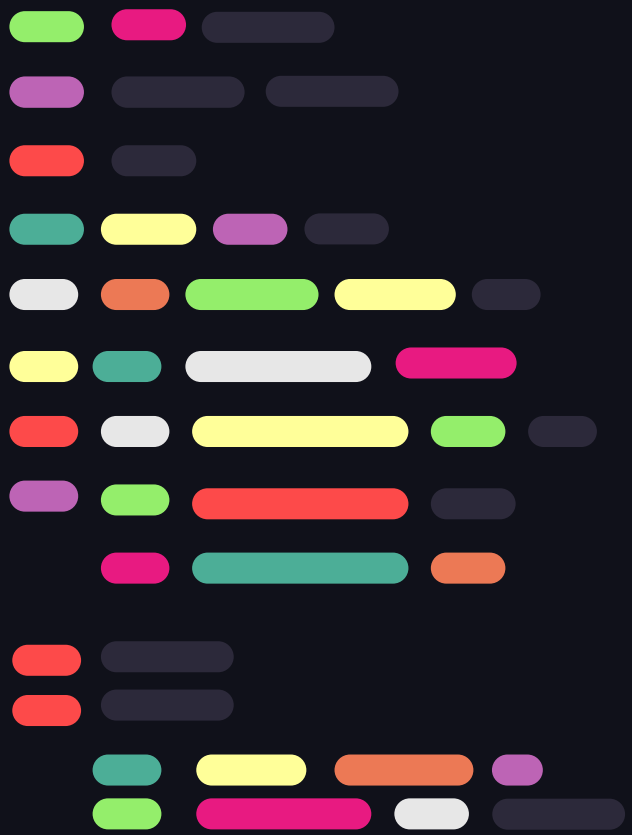
<Mohan De Zoysa –IIT JAVA CERTIFICATION>

Outline



Java JDBC

- Introduction
- Connecting to a Database
- Writing data to DB
- Reading data form DB
- Transaction Handling

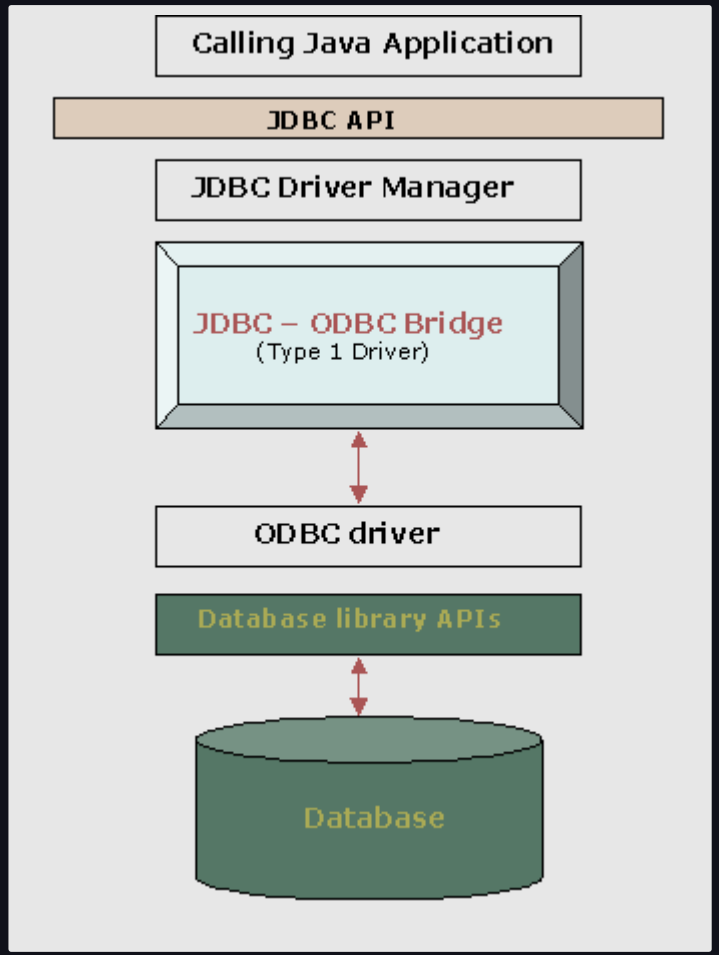


Java JDBC

JDBC stands for Java Database Connectivity, and it is a Java API that allows Java applications to connect to and interact with databases. JDBC is a fundamental part of Java Standard Edition (Java SE) and provides a standard interface for Java applications to communicate with relational databases

There are four types of JDBC drivers

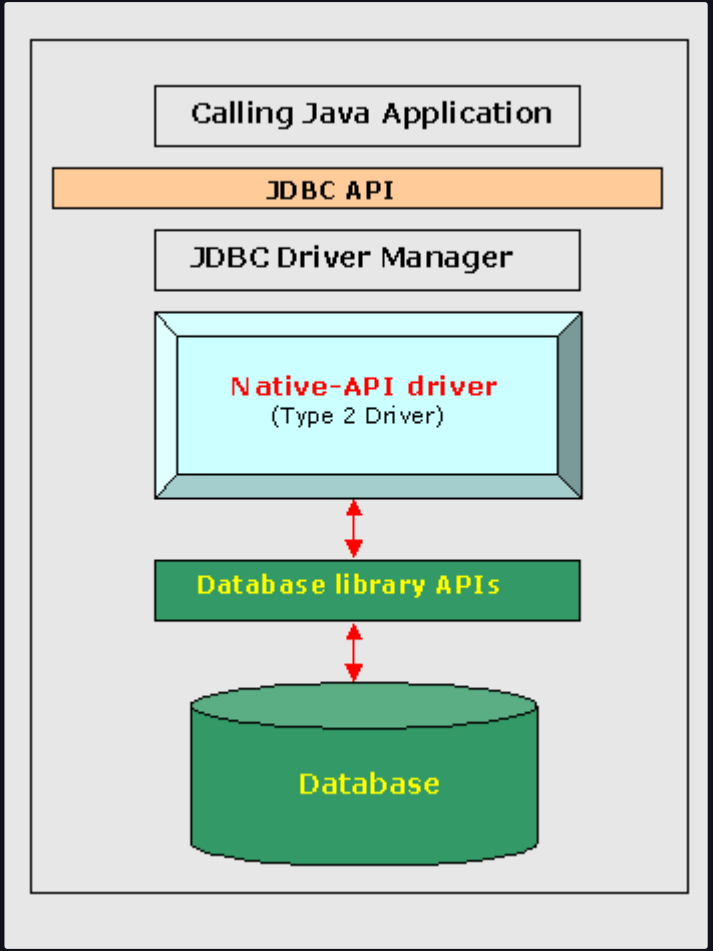
- **Type-1 Driver (JDBC-ODBC Bridge Driver):**
- **Type-2 Driver (Native-API Driver)**
- **Type-3 Driver (Network Protocol Driver or Middleware Driver):**
- **Type-4 Driver (Thin Driver or Direct-to-Database Driver)**



JDBC-ODBC Bridge Driver

This driver acts as a bridge between JDBC and ODBC (Open Database Connectivity). It relies on native ODBC libraries, so it is platform-dependent.

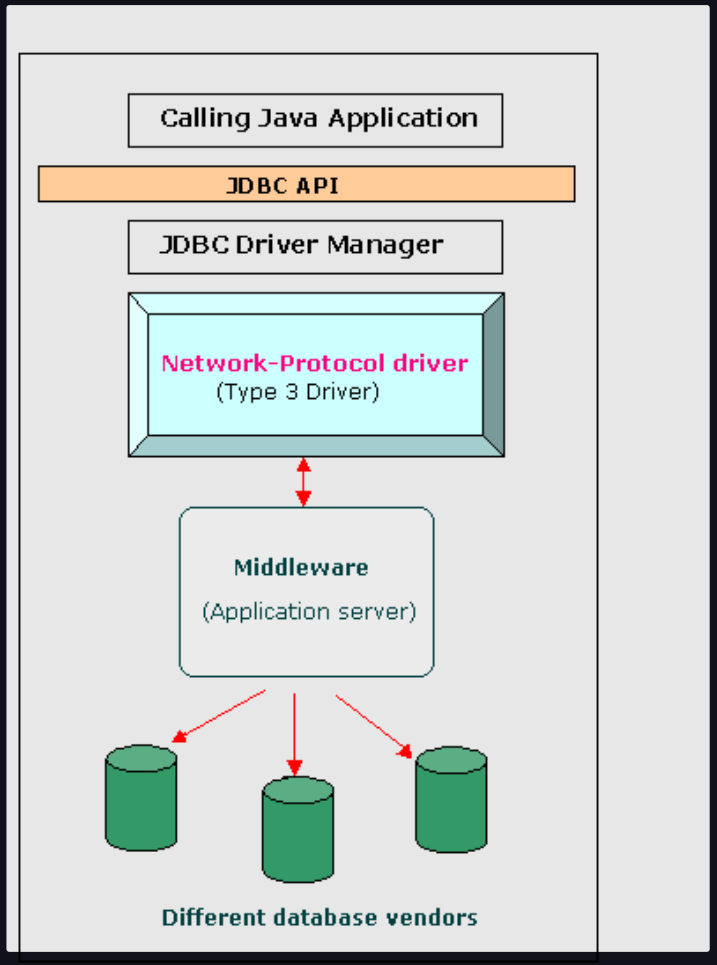
It is considered legacy and is rarely used in modern applications due to its limitations and platform dependencies.



Native-API Driver

This driver uses a database-specific native client library to communicate with the database server.

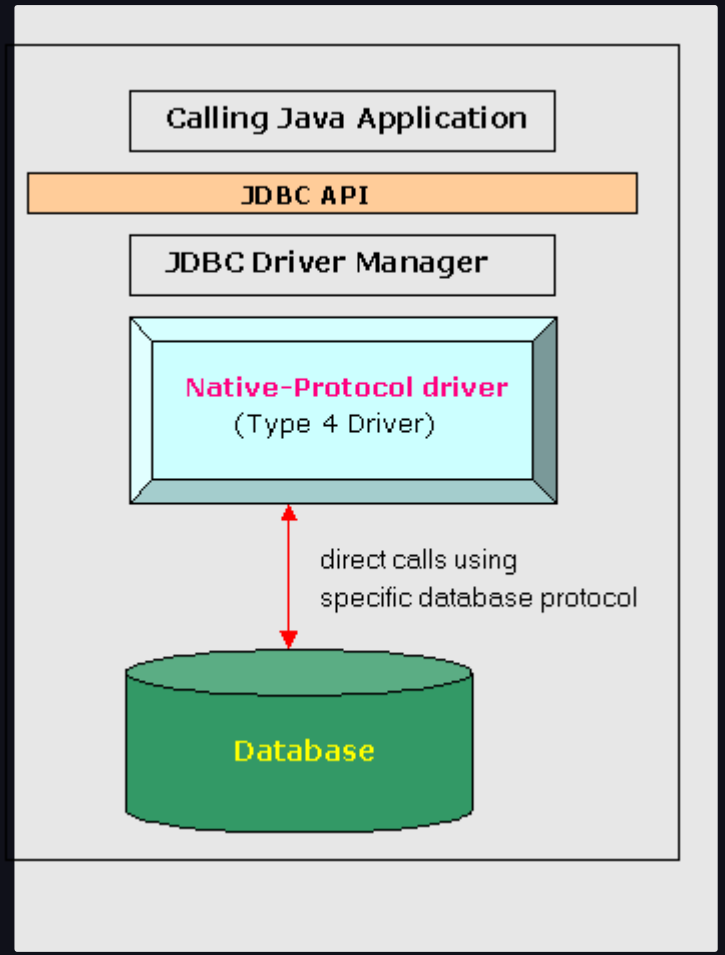
It is platform-dependent like the Type-1 driver but provides better performance compared to the Type-1 driver.



Network Protocol Driver or Middleware Driver

This driver uses a middle-tier server to communicate with the database. The middle-tier server converts JDBC calls into a database-specific protocol.

It is platform-independent, and the client-side code is written in Java. It offers the advantage of database independence.

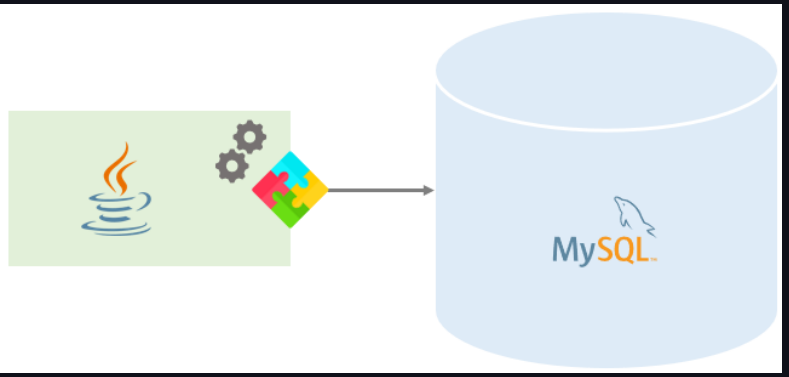


Thin Driver or Direct-to-Database Driver

This driver communicates directly with the database server without the need for a middle-tier server.

It is also platform-independent and provides better performance than the Type-3 driver.

The Type-4 driver is often the preferred choice for modern Java applications when connecting to databases.



Connecting to a Database

MySQL JDBC driver

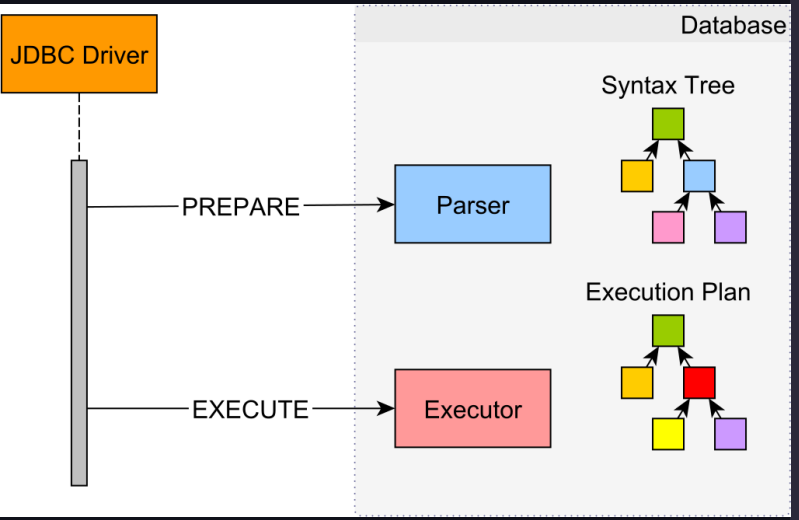
To connect to MySQL from Java, you must use the JDBC driver from MySQL. The MySQL JDBC driver is called *MySQL Connector/J*. You find the latest MySQL JDBC driver under the following

URL:http://dev.mysql.com/downloads/connector/j.

- **Find the JDBC Driver:**
- **Download the JDBC Driver JAR**
- **Add the JAR to Your Project**

- `Class.forName("com.mysql.cj.jdbc.Driver");`

Connecting to a Database



Prepared Statement

A PreparedStatement in Java is an interface used to execute precompiled SQL queries. It extends the Statement interface and provides additional functionality for parameterized queries, which can help prevent SQL injection attacks and improve performance for repeated queries.

Retrieving data

Retrieving data from a database in Java typically involves using JDBC (Java Database Connectivity) to establish a connection, create and execute a SQL query, and then process the results. Here's a step-by-step guide on how to retrieve data from a database in Java:

Import Required Libraries:

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

Establish a Database Connection:

```
Connection connection = null;
try {
    String jdbcURL = "jdbc:mysql://localhost:3306/mydatabase";
    String username = "yourUsername";
    String password = "yourPassword";

    connection = DriverManager.getConnection(jdbcURL, username, password);
} catch (SQLException e) {
    e.printStackTrace();
}
```

Create a SQL Query:

```
String sql = "SELECT * FROM employees WHERE department = ?";
```

Prepare and Execute the Query:

```
try {
    PreparedStatement preparedStatement = connection.prepareStatement(sql);
    preparedStatement.setString(1, "HR");
    preparedStatement.setIneger(2, 100);  // Set parameter values if necessary
    ResultSet resultSet = preparedStatement.executeQuery();
} catch (SQLException e) {
    e.printStackTrace();
}
```

Process the Results:

```
try {
    while (resultSet.next()) {
        int employeeId = resultSet.getInt("employee_id");
        String firstName = resultSet.getString("first_name");
        String lastName = resultSet.getString("last_name");
        int age = resultSet.getInt("age");

        // Process or display the retrieved data
        System.out.println("Employee ID: " + employeeId);
        System.out.println("First Name: " + firstName);
        System.out.println("Last Name: " + lastName);
        System.out.println("Age: " + age);
        System.out.println();
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

Close Resources:

```
try {
    resultSet.close();
    preparedStatement.close();
    connection.close();
} catch (SQLException e) {
    e.printStackTrace();
}
```

Import Required Libraries:

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

Establish a Database Connection:

```
Connection connection = null;
try {
    String jdbcURL = "jdbc:mysql://localhost:3306/mydatabase";
    String username = "yourUsername";
    String password = "yourPassword";

    connection = DriverManager.getConnection(jdbcURL, username, password);
} catch (SQLException e) {
    e.printStackTrace();
}
```

Inserting data to table

To insert data into a database using JDBC in Java, you can follow these steps:

Create a SQL Query:

```
String sql = "INSERT INTO employees (first_name, last_name, age) VALUES (?, ?, ?)";
```

Prepare and Execute the Query:

```
try {
    PreparedStatement preparedStatement = connection.prepareStatement(sql);
    preparedStatement.setString(1, "John"); // Set parameter values
    preparedStatement.setString(2, "Doe");
    preparedStatement.setInt(3, 30);

    int rowsInserted = preparedStatement.executeUpdate();
    if (rowsInserted > 0) {
        System.out.println("A new employee was inserted successfully.");
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

```
conn.setAutoCommit(false);
```

Commit & Rollback

Once you are done with your changes and you want to commit the changes then call commit() method on connection object as follows −

```
conn.commit( );
```

Otherwise, to roll back updates to the database made using the Connection named conn, use the following code −

```
conn.rollback( );
```

Handling Transactions

Handling transactions in Java with MySQL involves using JDBC (Java Database Connectivity) to manage the transaction boundaries. Here's how you can handle transactions in Java with MySQL:

If your JDBC Connection is in *auto-commit* mode, which it is by default, then every SQL statement is committed to the database upon its completion.

That may be fine for simple applications, but there are three reasons why you may want to turn off the auto-commit and manage your own transactions −