# OOP Class7

## JDBC: Java Database Connectivity

# Introduction

- **JDBC** is a Java standard that provides the interface for connecting from Java to relational databases. The JDBC standard is defined by Sun Microsystems and implemented through the standard java.sql interfaces. This allows individual providers to implement and extend the standard with their own JDBC drivers. JDBC stands for **Java Database Connectivity**, which is a standard Java API for database -independent connectivity between the Java programming language and a wide range of databases.

- The JDBC library includes APIs for each of the tasks commonly associated with database usage:
- Making a connection to a database
- Creating SQL or MySQL statements
- Executing that SQL or MySQL queries in the database
- Viewing & Modifying the resulting records

- JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

# Design of JDBC

- JDBC is designed to provide a database-neutral API for accessing relational databases from different vendors. Just as a Java application does not need to be aware of the operating system platform on which it is running, so too JDBC has been designed so that the database

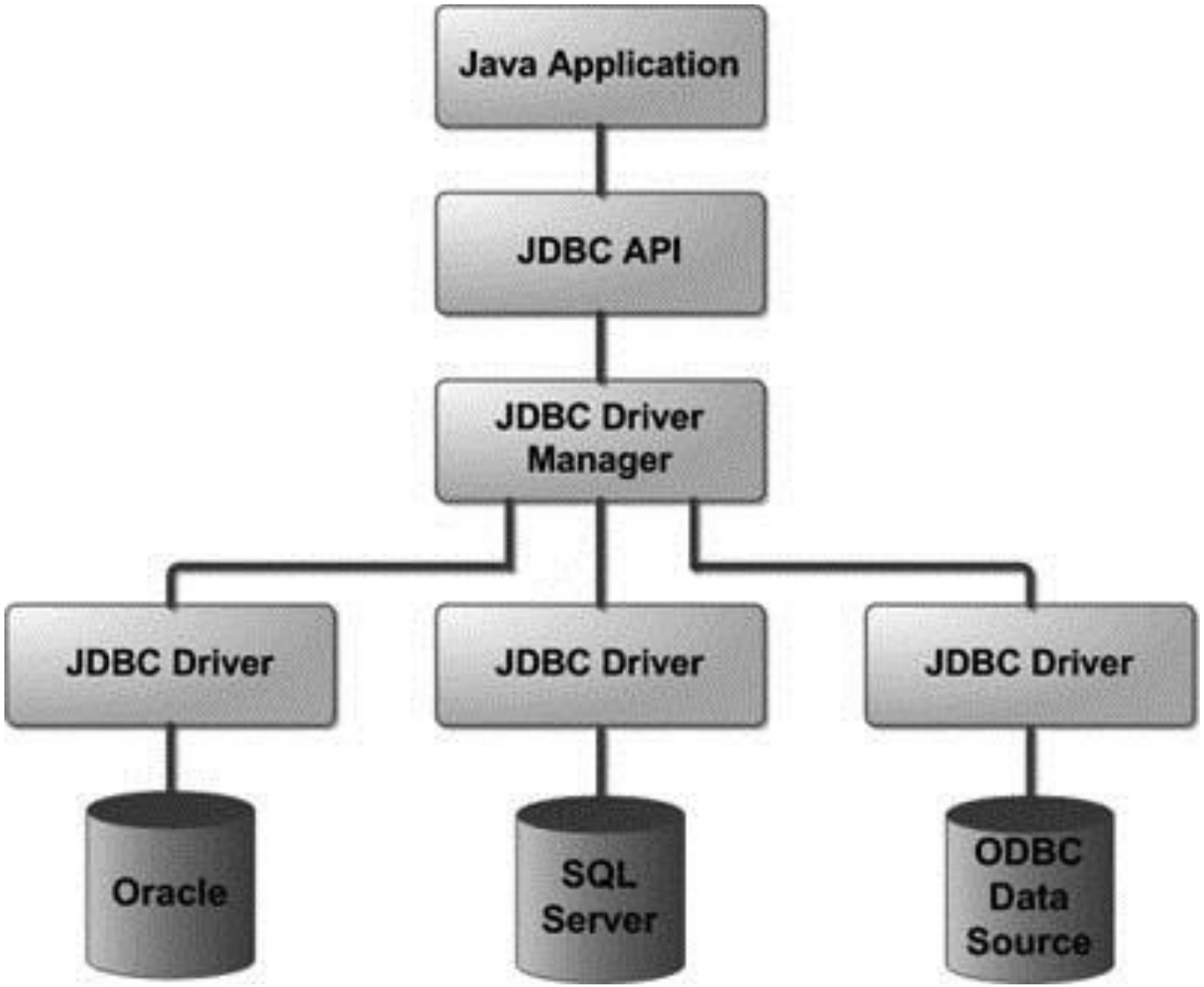
- This is not to say that JDBC cannot be used with another type of database. In fact, there are JDBC drivers that allow the API to be used to connect to both high-end, mainframe databases, which are not relational, and to access flat files and spreadsheets as databases (which are definitely not relational). But the reality is that JDBC is most commonly used with relational databases.

- The technical definition of a relational database is a database that stores dataas a collection of related entities. These entities are composed of attributes that describe the entity and attributes.
- Uses SQL

- JavaSoft's JDBC consists of two layers: the **JDBC API** and the **JDBC Driver Manager API.**
- The **JDBC API** is the top layer and is the programming interface in Java to *structured query language* (SQL) which is the standard for accessing relational databases.
- The JDBC API communicates with the **JDBC Driver Manager API**, sending it various SQL statements. The manager communicates (transparent to the programmer) with the various third party drivers (provided by Database vendors like Oracle) that actually connect to the database and return the information from the query.

# JDBC Architecture:

- The JDBC API supports both two-tier and three-tier processing models for database access but in general JDBC Architecture consists of two layers:
- **JDBC API:** This provides the application-to-JDBC Manager connection.
- **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.
- The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.


Java Application
JDBC API
JDBC Driver Manager
JDBC Driver
JDBC Driver
JDBC Driver
Oracle
SQL Server
ODBC Data Source

# Common JDBC Components:

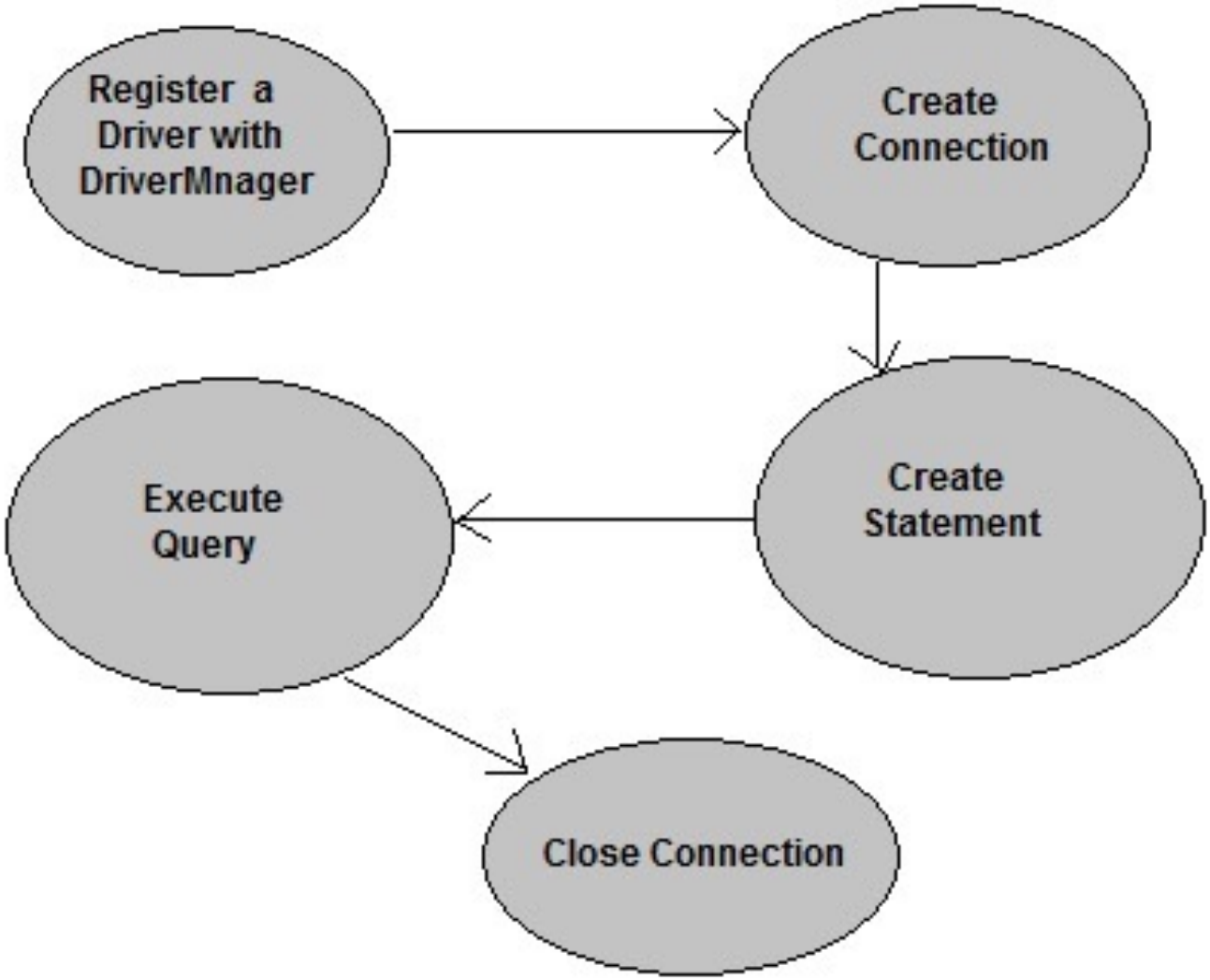
- **DriverManager:** This class manages a list of database drivers.
- **Driver:** This interface handles the communications with the database server.
- **Connection :** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement :** You use objects created from this interface to submit the SQL statements to the database.
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It
- **SQLException:** This class handles any errors that occur in a database application.

# 2) Native-API driver

- The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.
- Native API driver uses native API to connect a java program directly to the database. Native API id s C, C++ library, which contains a set of function used to connect with database directly. Native API will be different from one database to another database. So this Native API driver is a database dependent driver.

# Steps to connect a Java Application to Database

- The following 5 steps are the basic steps involve in connecting a Java application with Database using JDBC.
- Register the Driver
- Create a Connection
- Create SQL Statement
- Execute SQL Statement
- Closing the connection


Register a Driver with DriverMnager
Create Connection
Create Statement
Execute Query
Close Connection

- In this example we are using MySql as the database. So we need to know following informations for the mysql database:
- **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
- **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sujata** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sujata is the database name. We may use any database, in such case, you need to replace the sujata with your database name.
- **Username:** The default username for the mysql database is **root**.
- **Password:** Password is given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

# Step 1: Import JDBC Packages

- import java.sql.*;
- import oracle.jdbc.driver.*;
- import oracle.sql.*;

# Step 2:**Load and Register the JDBC Driver**

- Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
- Class.forName("oracle.jdbc.driver.OracleDriver");

# Step 3: **Connecting to a Database**

- The getConnection() method is an overloaded method that takes
- Three parameters, one each for the URL, username, and password.
- Only one parameter for the database URL. In this case, the URL contains the username and password.

- Connection conn = DriverManager.getConnection(URL, username, passwd);
- Connection conn = DriverManager.getConnection(URL);

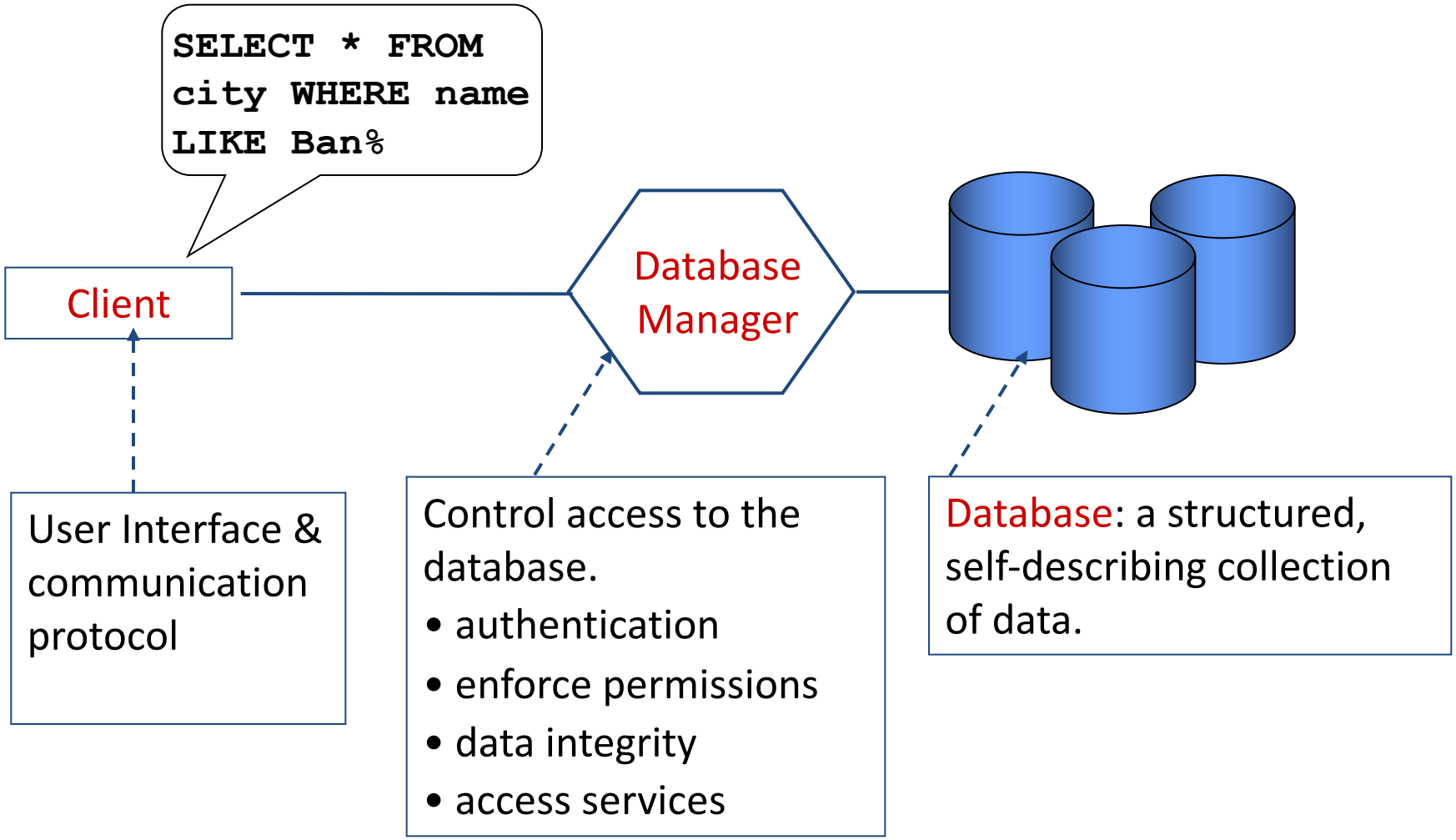
# Step 4: **Querying the Database**

- Statement sql_stmt = conn.createStatement();

SQL in a single slide:

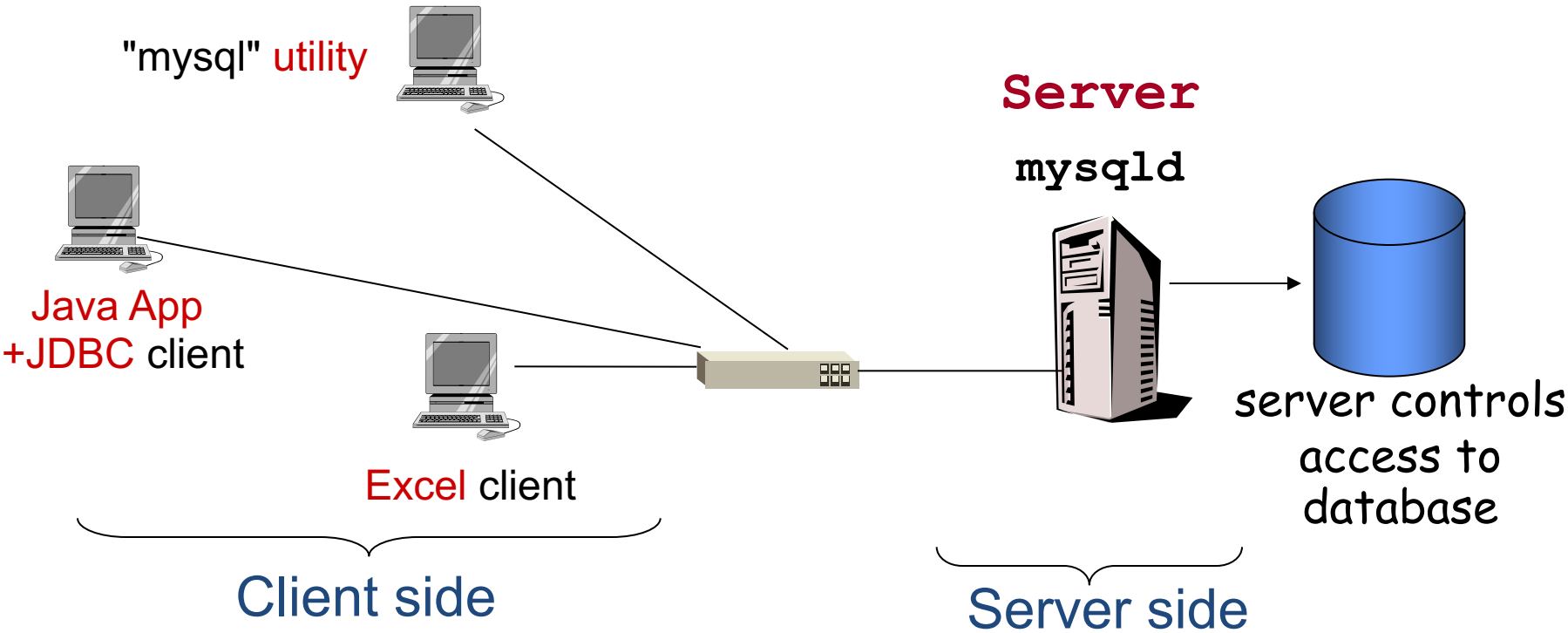
- 123

# Database Management System



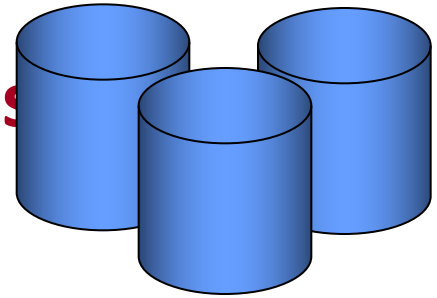
# Client - Server Databases

- Database Server is a separate *process* on a host.
- Clients can be on any machine.
- Many programs may be clients using a standard API.



# Structure of a Database

- A database *system* may contain many **databases**.
- posed of s

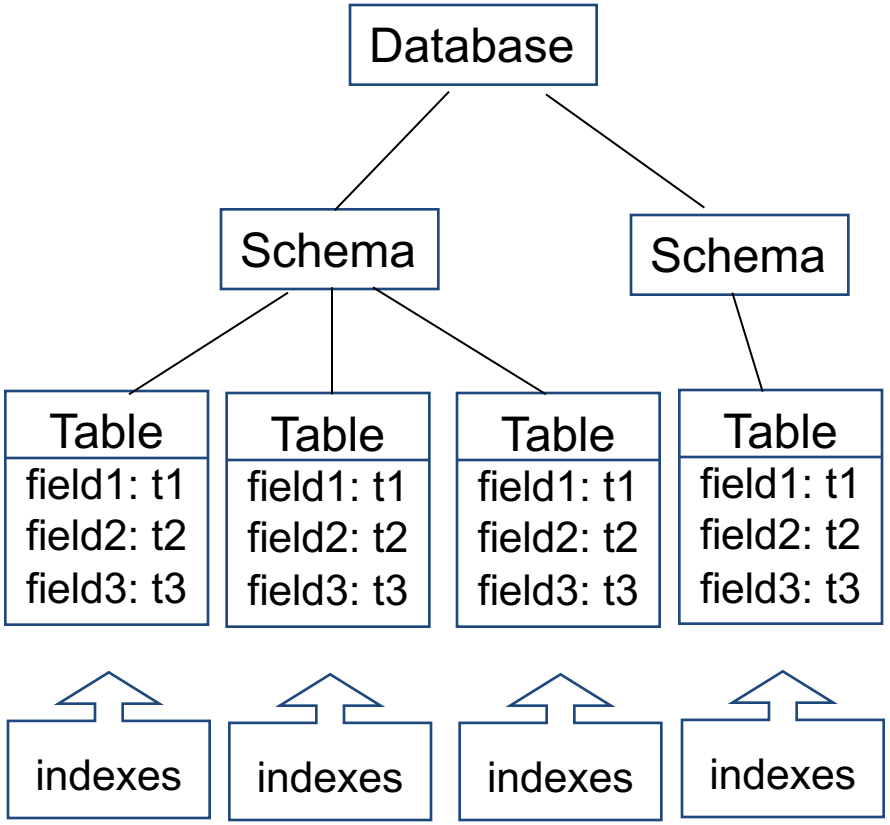


```
sql> SHOW databases;
+--------------+
| Database     |
+--------------+
| mysql        |
| test         |
| bank         |
| world        |
+--------------+
```

*MySQL only shows databases that a user has permission to access.*

```
sql> USE bank;
sql> SHOW tables;
+----------------+
| Tables_in_bank |
+----------------+
| accounts       |
| clients        |
+----------------+
```

# A Database Structure




A database contains schema, which describe the organization of the database.

A schema can contain:

tables - containing data

index files - for fast lookup of data

stored procedures, constraints, triggers, and more

# Contents of a Table

- A table contains the actual data in records (rows).
- A record is composed of fields (columns).
- Each record contains one set of data values.

records (rows)

| ID | Name | CCode | District | Populatn |
|---|---|---|---|---|
| 3320 | Bangkok | THA | Bangkok | 6320174 |
| 3321 | Nonthaburi | THA | Nonthaburi | 292100 |
| 3323 | Chiang Mai | THA | Chiang Mai | 171100 |

fields (columns)

# Key field for Identifying Rows

- A table contains a *primary key* that uniquely identifies a row of data.
- Each record must have a distinct value of primary key
- The primary key is used to relate (join) tables

ID is the primary key in City table.

| ID | Name | CCode | District | Populatn |
|---|---|---|---|---|
| 3320 | Bangkok | THA | Bangkok | 6320174 |
| 3321 | Nonthaburi | THA | Nonthaburi | 292100 |
| 3323 | Chiang Mai | THA | Chiang Mai | 171100 |

# Structure of a Table

Every field has:

- a name
- a data type and length

To view the structure of a table use:

DESCRIBE *tablename*

```
sql> DESCRIBE City;
+-------------+-----------+-----+-----+---------+----------------+
| Field       | Type      | Null| Key | Default | Extra          |
+-------------+-----------+-----+-----+---------+----------------+
| ID          | int(11)   | NO  | PRI |         | auto_increment |
| Name        | char(35)  | NO  |     |         |                |
| CountryCode | char(3)   | NO  |     |         |                |
| District    | char(20)  | NO  |     |         |                |
| Population  | int(11)   | NO  |     | 0       |                |
+-------------+-----------+-----+-----+---------+----------------+
```

# Structure of a Table

"`SHOW columns FROM tablename`"
shows the same information.

```
sql> SHOW columns FROM City;
+-------------+-----------+-----+-----+---------+----------------+
| Field       | Type      | Null| Key | Default | Extra          |
+-------------+-----------+-----+-----+---------+----------------+
| ID          | int(11)   | NO  | PRI |         | auto_increment |
| Name        | char(35)  | NO  |     |         |                |
| CountryCode | char(3)   | NO  |     |         |                |
| District    | char(20)  | NO  |     |         |                |
| Population  | int(11)   | NO  |     | 0       |                |
+-------------+-----------+-----+-----+---------+----------------+
```

Fields may have a default value to use if a value is not assigned explicitly.

# 4 Basic Database Operations

The 4 most common operations:

| | |
|---|---|
| SELECT | query (search) the data |
| INSERT | add new records to a table(s) |
| UPDATE | modify existing record(s) |
| DELETE | delete record(s) from a table |

**What is CRUD?**

Programmers call these operations "CRUD".

What does CRUD stand for?

# Querying Data in a Table

SELECT displays field values from a table:

```
SELECT field1, field2, field3 FROM table ;
```

- displays ALL rows from the table.
- use LIMIT *number* to limit how many results.

```
sql> SELECT accountNumber, balance FROM accounts;
+---------------+---------------+----------+---------+
| accountNumber | accountName   | clientID | balance |
+---------------+---------------+----------+---------+
| 11111113      | P.Watanapong  | 00001001 |  300000 |
| 11111114      | CPE Fund      | 00001002 | 1840000 |
+---------------+---------------+----------+---------+
```

# SELECT statement with *

- Display values for all fields in table:

```
SELECT * FROM tablename ;
```

```
sql> SELECT * from accounts;
+---------------+---------------+----------+----------+
| accountNumber | accountName   | clientID | balance |
+---------------+---------------+----------+----------+
| 11111113      | P.Watanapong  | 00001001 |  300000 |
| 11111114      | CPE Fund      | 00001002 | 1840000 |
+---------------+---------------+----------+----------+
```

# Qualifying SELECT

- Select columns from a table that match some criteria:

```
SELECT field1, field2, field3
     FROM  table
     WHERE condition
     ORDER BY field1,... [ASC|DESC];
```

**Example: cities with population > 5 M**

```
sql> SELECT * FROM City
        WHERE population > 5000000
        ORDER BY population DESC;
```

# Strings in SQL

- Use single quote mark around String constants.

```
SELECT * FROM Country
         WHERE name = 'Thailand';
```

```
SELECT * FROM City
         WHERE Name = 'Los Angeles';
```

# WHERE conditions

| | |
|---|---|
| `name = 'Bangkok'` | equality test |
| `name LIKE 'Bang%'` | pattern match |
| `population >= 100000`<br>`population < 500000`<br>`gnp <> 0` | relations<br><br><br><> is not equals |
| `grade IN ('A','B','C','D','F')` | contained in set |
| | |
| | |
| | |

# Other Functions in SQL

Functions can have *arguments*, just like C, Java, etc.

```
SUM( expression )
MAX( expression )
MIN( expression )
COUNT( expression )
```

```
sql> SELECT MAX(SurfaceArea) FROM country;
1075400.00  (sq.km.)
```

WRONG: *This will NOT find the largest country!*

```
sql> SELECT MAX(SurfaceArea), Name FROM country;
1075400.00   Afghanistan
```

# UPDATE statement

Change values in one or more records:

```
UPDATE table
    SET field1=value1, field2=value2
    WHERE condition;
```

```
sql> UPDATE city
       SET population=40000
       WHERE name='Bangsaen' AND countrycode='THA';

Query OK, 1 row affected (0.09 sec)
```

| name | countrycode | district | population |
|---|---|---|---|
| 11111111 | THA | Chonburi | 40000 |

# UPDATE multiple columns

You can change multiple columns:

```
UPDATE table
    SET field1=value1, field2=value2
    WHERE condition;
```

Example: Update  population and GNP of Thailand

```
sql> UPDATE country
       SET population=68100000, gnp=345600
       WHERE code='THA';

Query OK, 1 row affected (0.09 sec)
```

Source: CIA World Factbook (on the web)

# Warning: UPDATE is immediate!

- Changes occur immediately. (Can't undo w/o trans.)

**Be Careful!** If you forget the WHERE clause it will change *all* the rows in the table!

```
sql> UPDATE country SET HeadOfState='Obama';
        /* Oops! I forgot "WHERE ..." */
+------+----------------+-------------+--------------+
| Code | Name           | Continent   | HeadOfState  |
+------+----------------+-------------+--------------+
| AFG  | Afghanistan    | Asia        | Obama        |
| NLD  | Netherlands    | Europe      | Obama        |
| ALB  | Albania        | Europe      | Obama        |
| DZA  | Algeria        | Africa      | Obama        |
| ASM  | American Samoa | Oceania     | Obama        |
| AND  | Andorra        | Europe      | Obama        |
| AGO  | Angola         | Africa      | Obama        |
```

Obama rules!

# Deleting Records

- DELETE one or more records

  ```
  DELETE FROM tablename WHERE condition;
  ```

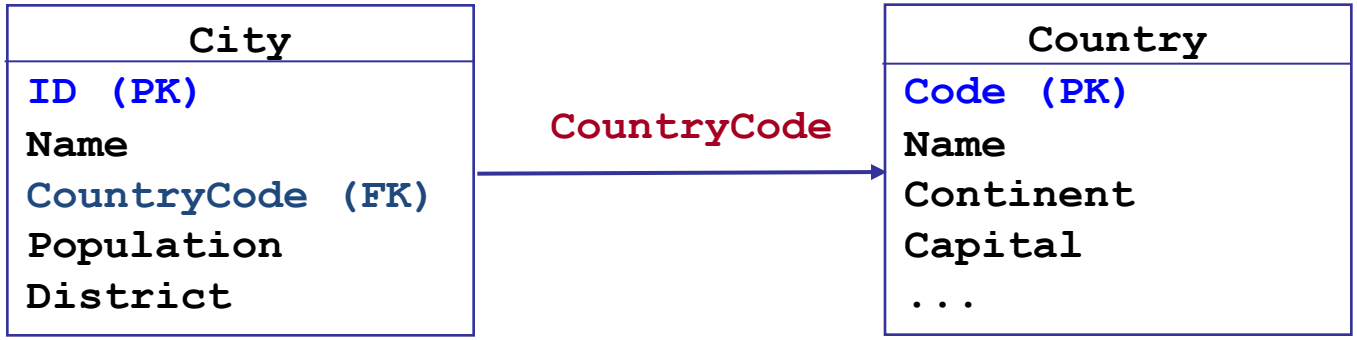
Example: Delete all cities with zero population

```
sql> DELETE FROM City WHERE population <= 0;
Query OK, 5 rows deleted.
```

# Keys

Every table should have a primary key that uniquely identifies each row.

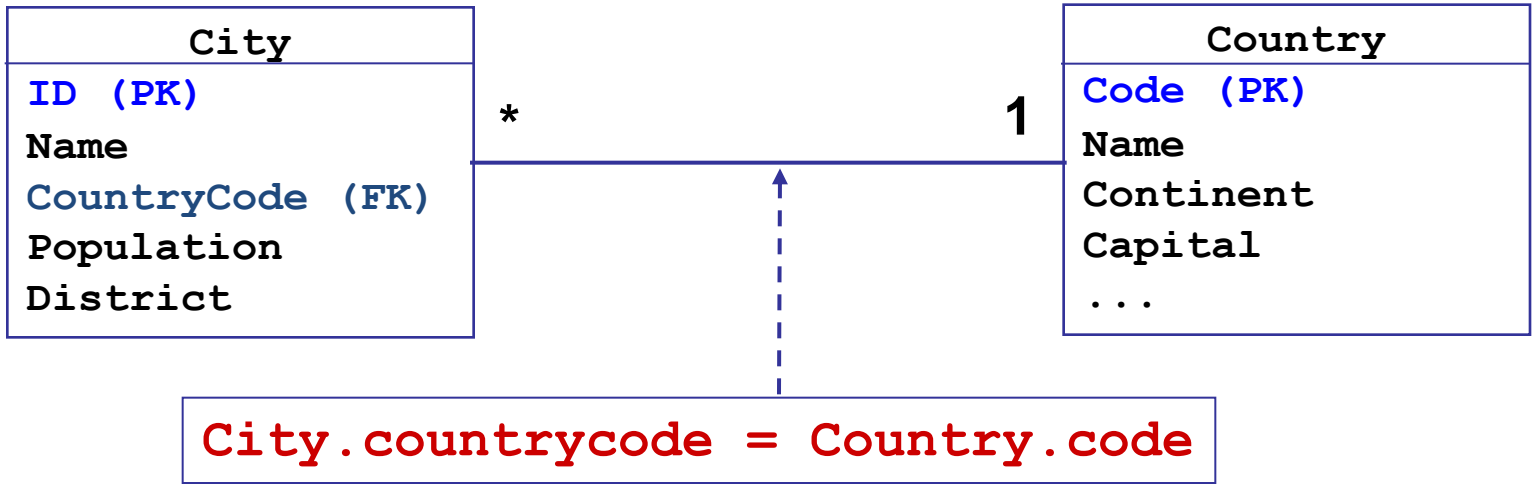
```
sql> DESCRIBE Country;
+-------------+-----------+-----+-----+---------+----------------+
| Field       | Type      | Null| Key | Default | Extra          |
+-------------+-----------+-----+-----+---------+----------------+
| Code        | char(3)   | NO  | PRI |         |                |
| Name        | char(52)  | NO  |     |         |                |
| ...         |           |     |     |         |                |
```



# Joining Tables

- Relate or "join" tables using a condition.
- Use "`table.field`" to qualify a field name:

**`Country.code`** **`Country.name`**



# Logical operations

- OR

```
SELECT * FROM City WHERE
      District='Songkhla' OR District='Bangkok';
```

- AND

```
SELECT Name, SurfaceArea FROM Country WHERE
 Continent = 'Africa' AND SurfaceArea > 1000000;
```

- NOT

```
SELECT * FROM Accounts WHERE
     NOT AvailableBalance = 0;
```

# Subqueries

- Use the result of one query as part of another query.

Example: Which country has the largest population?

```
SELECT Name, Population
    FROM country
    WHERE Population =
    ( SELECT max(population) FROM country);
```

*Subquery*

*To use subqueries in MySQL you need version 4.1 or newer.*

# Data Definition Commands

**These commands alter the structure of a database**

**CREATE** create a Table, Index, or Database

**ALTER** modify structure of a Database or Table

**DROP** delete an entire Table, Index, or Database

**RENAME** rename a Table

# Creating a Table

To add a new table to a database:

```
CREATE TABLE tablename
        (field1, field2, ... )
     options ;
```

```
sql> CREATE TABLE CUSTOMER (
      accountNumber VARCHAR(8) NOT NULL,
      clientID      VARCHAR(40) NOT NULL,
      balance       DOUBLE  DEFAULT '0',
       availableBalance DOUBLE DEFAULT '0'
       ) ;
Query OK, 0 rows affected.
```

# Productivity Hint

- Type the "CREATE TABLE" statement into a file.
- "source" the file in mysql: **`source filename;`**

File: **`/temp/create-table.sql`**

```
CREATE TABLE CUSTOMER (
   accountNumber CHAR(10) NOT NULL,
   clientID      VARCHAR(40) NOT NULL,
   balance       DOUBLE  DEFAULT '0',
   availableBalance DOUBLE DEFAULT '0',
   PRIMARY KEY( clientID )
    ) ;
```

```
sql> SOURCE /temp/create-table.sql;
Query OK, 0 rows affected.
```

# Deleting Records a Table

You *must* specify a "WHERE" clause for rows to delete.

If there is no "WHERE", it ***deletes all rows*** !!

```
DELETE FROM tablename
       WHERE condition ;
```

```
-- first use SELECT to verify condition
sql> SELECT * FROM city
     WHERE name="Bangsaen";
sql> DELETE FROM city
     WHERE name="Bangsaen";
Query OK, 1 row affected.
```

# Resources

## MySQL

- http://dev.mysql.com/tech-resources/articles/

## Learning SQL

- http://www.w3schools.com/sql/
  nice tutorial and command reference

## Step 5: **Executing the Query and Returning a ResultSet**

- ResultSet rset = sql_stmt.executeQuery ("SELECT empno, ename, sal, deptno FROM emp ORDER BY ename");

- Alternatively, the SQL statement can be placed in a string and then this string passed to the executeQuery() function. This is shown below.
- String sql = "SELECT empno, ename, sal, deptno FROM emp ORDER BY ename";
- ResultSet rset = sql_stmt.executeQuery(sql);

## Step 6: **Closing the ResultSet and Statement**

- rset.close(); (ResultSet object)
- sql_stmt.close(); (Statement Object)
- conn.close(); (Connection object)

# Example

```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

# JDBC Statements

- Once a connection is obtained we can interact with the database. The JDBC *Statement, CallableStatement,* and *PreparedStatement* int erfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database.

| Interfaces | Recommended Use |
| --- | --- |
| Statement | Use the for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters. |
| PreparedStatement | Use the when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime. |
| CallableStatement | Use the when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters. |

# 1. The Statement Objects

- **Creating Statement Object**

```
Statement stmt = null;
try
{
stmt = conn.createStatement( ); . . .
}
catch (SQLException e)
{ . . . }
```

# Closing Statement Object

```
Statement stmt = null;
try
{
stmt = conn.createStatement( ); . . .
}
catch (SQLException e)
{ . . . }
finally
{
stmt.close();
}
```

# Closing PreparedStatement Object

```
PreparedStatement pstmt = null;
try
{
String SQL = "Update Employees SET age = ? WHERE id = ?";
pstmt = conn.prepareStatement(SQL); . . .
}
catch (SQLException e)
{ . . . }
finally
{
pstmt.close(); }
```

- These type and mode are predefined in ResultSet Interface of Jdbc like below which is static final.

**Type:**

- public static final int TYPE_FORWARD_ONLY=1003
- public static final int TYPE_SCROLL_INSENSITIVE=1004
- public static final int TYPE_SCROLL_SENSITIVE=1005

**Mode:**

- public static final int CONCUR_READ_ONLY=1007
- public static final int CONCUR_UPDATABLE=1008

- **Example**
- Statement stmt=con.CreateStatement(1004, 1007); **or**
- Statement stmt=con.CreateStatement(ResutlSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);

# Batch Processing in JDBC

- Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast.
- The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing.
- **Advantage of Batch Processing**
- Fast Performance

# Methods of Statement interface

The required methods for batch processing are given below:

| Method | Description |
| --- | --- |
| void addBatch(String query) | It adds query into batch. |
| int[] executeBatch() | It executes the batch of queries. |

```
import java.sql.*;
class FetchRecords
{
public static void main(String args[])throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:
@localhost:1521:xe","system","oracle");
Statement stmt=con.createStatement();
stmt.addBatch("insert into user420 values(190,'abhi',40000)");
stmt.addBatch("insert into user420 values(191,'umesh',50000)");
stmt.executeBatch();//executing the batch
 con.commit();
con.close();
}}
```

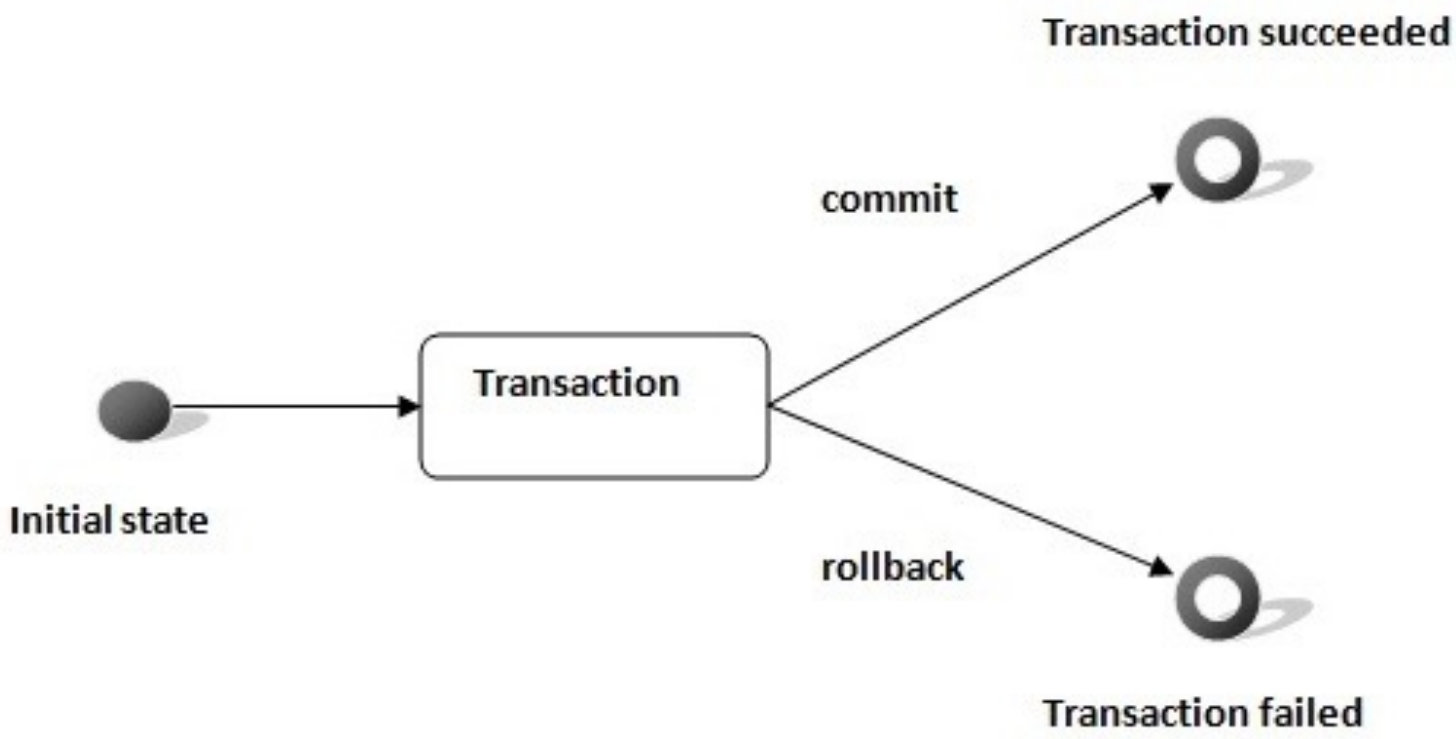
# Transaction Management in JDBC

Transaction represents **a single unit of work**. The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.

- **Atomicity** means either all successful or none.
- **Consistency** ensures bringing the database from one consistent state to another consistent state.
- **Isolation** ensures that transaction is isolated from other transaction.
- **Durability** means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

**Advantage of Transaction Mangaement**

- **fast performance** It makes the performance fast because database is hit at the time of commit.


Transaction succeeded
commit
Transaction
Initial state
rollback
Transaction failed

| Method | Description |
| --- | --- |
| void setAutoCommit(boolean status) | It is true bydefault means each transaction is committed bydefault. |
| void commit() | commits the transaction. |
| void rollback() | cancels the transaction. |

```
import java.sql.*;
class FetchRecords
{
public static void main(String args[])throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:
@localhost:1521:xe","system","oracle");
con.setAutoCommit(false);
 Statement stmt=con.createStatement();
stmt.executeUpdate("insert into user420 values(190,'abhi',40000)
");
stmt.executeUpdate("insert into user420 values(191,'umesh',500
00)");
 con.commit();
con.close();
}}
```

- 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 –
- To increase performance.
- To maintain the integrity of business processes.
- To use distributed transactions.

# 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( );

# Using Savepoints

- The new JDBC 3.0 Savepoint interface gives you the additional transactional control. Most modern DBMS, support savepoints within their environments such as Oracle's PL/SQL.
- When you set a savepoint you define a logical rollback point within a transaction. If an error occurs past a savepoint, you can use the rollback method to undo either all the changes or only the changes made after the savepoint.
- The Connection object has two new methods that help you manage savepoints –

- **setSavepoint(String savepointName):** Defines a new savepoint. It also returns a Savepoint object.
- **releaseSavepoint(Savepoint savepointName):**Deletes a savepoint.

```java
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();

   //set a Savepoint
   Savepoint savepoint1 = conn.setSavepoint("Savepoint1");
   String SQL = "INSERT INTO Employees " +
                "VALUES (106, 20, 'Rita', 'Tez')";
   stmt.executeUpdate(SQL);
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees " +
                "VALUES (107, 22, 'Sita', 'Tez')";
   stmt.executeUpdate(SQL);
   // If there is no error, commit the changes.
   conn.commit();

}catch(SQLException se){
   // If there is any error.
   conn.rollback(savepoint1);
}
```