Automation Testing

Transaction Management

Learning Objectives

By the end of this lesson, you will be able to:

- Illustrate transaction management
- List the types of transaction
- Illustrate the functions and methods of transaction
- List the applications and real-world examples of transactions in JDBC

Transaction in JDBC

Transaction in JDBC

A transaction refers to a logical unit of work. JDBC transaction makes sure that a set of SQL statements are executed as a unit.

Types of Transaction in JDBC

ACID Properties of Transaction Management

ACID stands for atomicity, consistency, isolation, and durability.

Consistency: The data must be consistent before and after the transaction.

Durability: Successful transactions occur even if the system fails.

Isolation: Multiple transactions occur independently without any interference.

Atomicity: The entire transaction takes place at once or does not happen at all.

Functions of Transaction Management

The most important functions in transaction management are:

Commit	Commit() makes the changes permanent in the database. The users cannot undo or revoke the changes.
Rollback	Rollback() undoes the changes till the last commit or mentioned savepoint.
Savepoint	Savepoint helps to create a checkpoint in a transaction and allows users to perform a rollback to that particular savepoint.

Methods Of Transaction in JDBC

Methods of Transaction Management

The connection interface provides five methods of transaction:

- setAutoCommit()
- Commit()
- Rollback()
- setSavePoint()
- releaseSavepoint()

setAutoCommit() Method

The value of the Autocommit() method is set to True by default. After the execution of the SQL statement, it will be committed automatically.

Sample Code

```
connection.setAutoCommit(true);
```

Commit() Method

The Commit() method is used to make the changes in the data, which are made by the SQL statement.

Sample Code

```
Con.commit();
```

Rollback() Method

The rollback() method is used to undo the changes till the last commit has happened.

Sample Code

```
Con.rollback();
```

setSavingpoint() Method

When the user sets a savepoint in the transaction, they can use the rollback() method to undo all the changes till the savepoint or after the savepoint.

Sample Code

```
Savepoint savePoint=
con.setSavepoint(“Mysavepoint”);
```

releaseSavepoint() Method

The releaseSavepoint() method accepts the name of the savepoint and releases or deletes the specified savepoint.

Sample Code

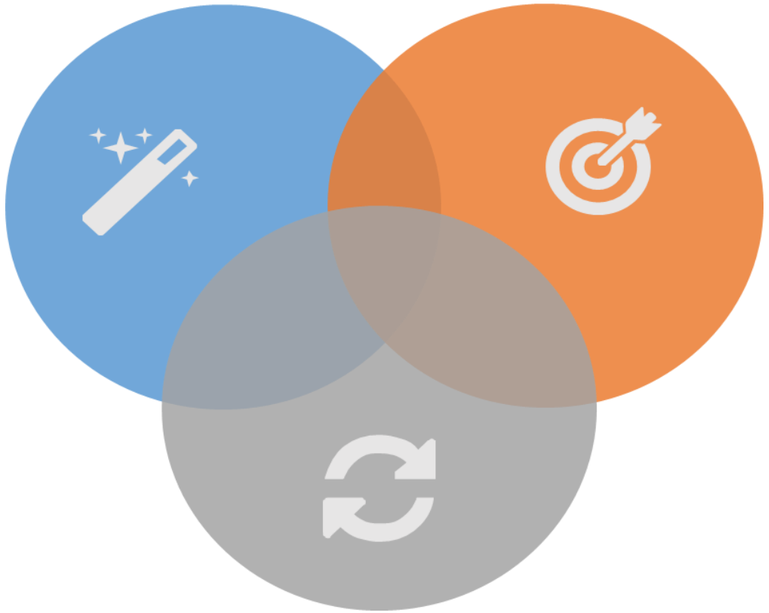
```
Release[savepoint] savepoint_name
```

Applications And Real-World Usages

JDBC API

JDBC provides universal data access from the Java programming language.
JDBC manages these three programming activities:

Connect to a data source, like a database.



Send queries and update statements to the database.

Retrieve and process the results received from the database in answer to the query.

Sample Code for JDBC API

This is the sample code of JDBC API:

Sample Code

```
Public void connectToAndQueryDatabase(String username,String password)
{

       Connection con = DriverManager.getConnection
    (“jdbc:myDriver:myDatabase”,username,password);

       Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(“SELECT a, b, c from TABLE1”);

    while(rs.next())
    {
      int x = rs.getInt(“a”);
      String s = rs.getString(“b”);
       float f = rs.getfloat(“c”);
    }
}
```

Explanation of the Code

DriverManager connects to a database driver and logs into the database.

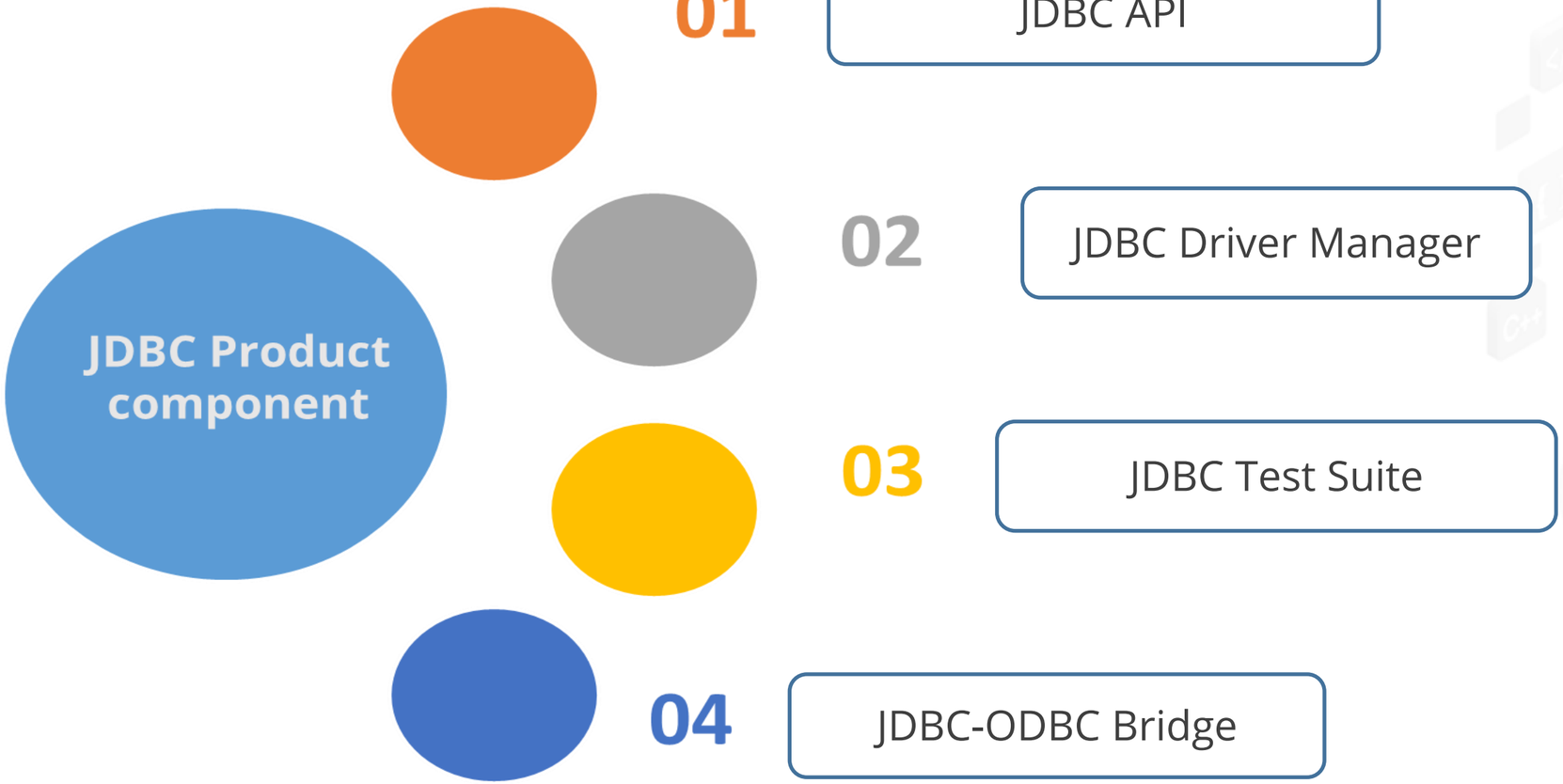
Then **Statement object** carries your SQL language query to the database.

Later, the **ResultSet object** retrieves the result of the query.

At last, it executes a simple **while loop**, which retrieves and displays those results.

JDBC Product Components

JDBC includes four components:



JDBC Product Components

JDBC API

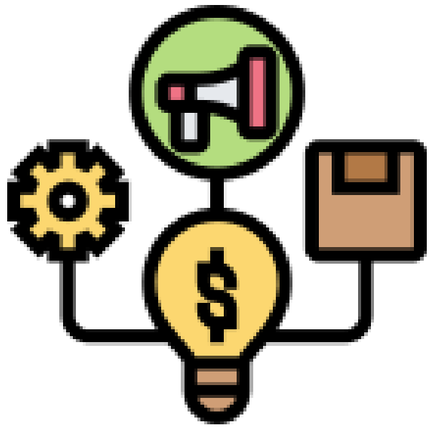
It provides a programmatic access to relational databases from java programming language. By using JDBC API, applications can execute SQL statements, retrieve results, and propagate changes back to an underlying data source.

The JDBC 4.0 API is divided into two packages: java.sql and javax.sql

JDBC Product Components

JDBC Driver Manager

This class defines objects that can connect java applications to a JDBC driver.

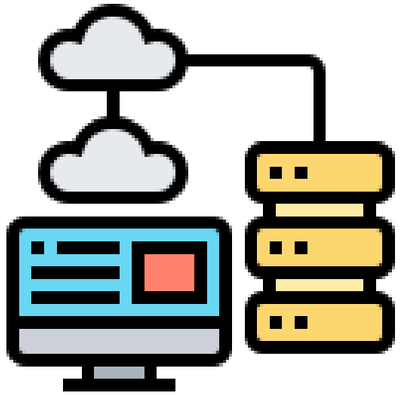


The standard extension packages java.naming and javax.sql are used as a **Datasource** object to establish a connection with a data source.

JDBC Product Components

JDBC Test Suite

The JDBC driver test suite helps to determine that JDBC drivers will run the program, and they exercise many important features in the JDBC API.

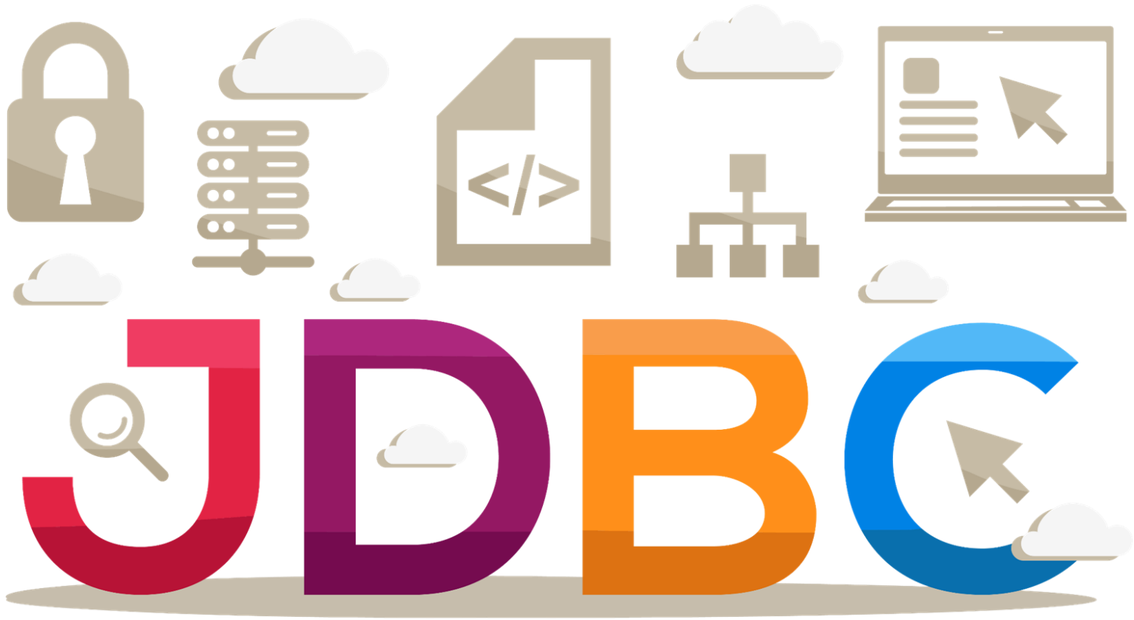


JDBC-ODBC Bridge

The Java Software bridge provides JDBC access via ODBC drivers. The ODBC driver is most appropriate on a corporate network where client installations are not mandatory.

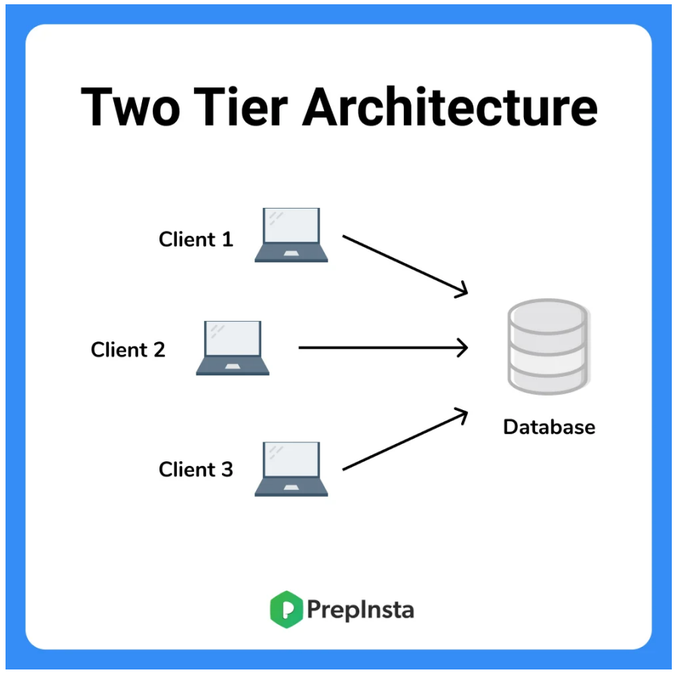
JDBC Architecture

The JDBC API supports two- and three-tier processing models for database access.



Two-Tier Architecture

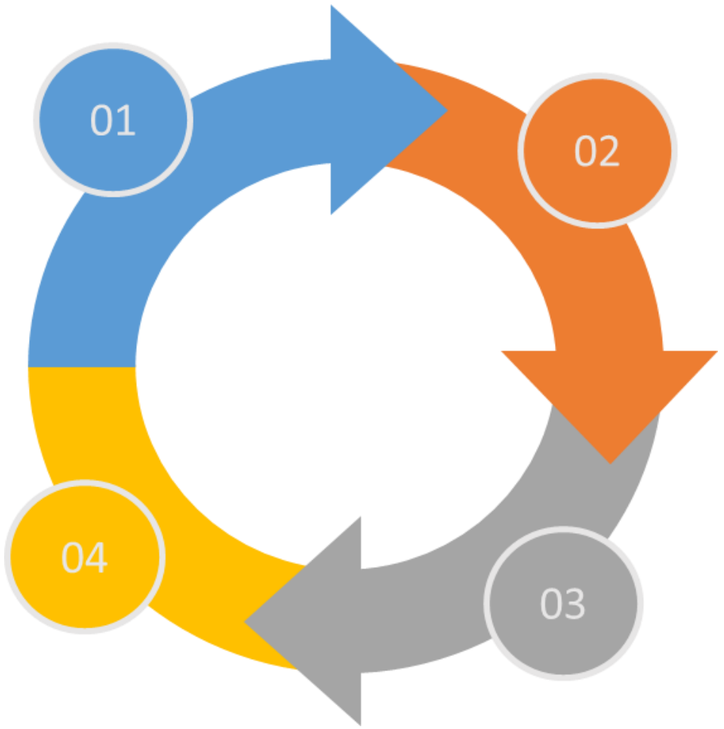
The two-tier architecture is a software in which an interface runs on a client and data structure gets stored on a server.



Three-Tier Architecture

The three-tier architecture is a well-established software application architecture that organizes applications into three logical and physical computing tiers:

User



Presentation Layer

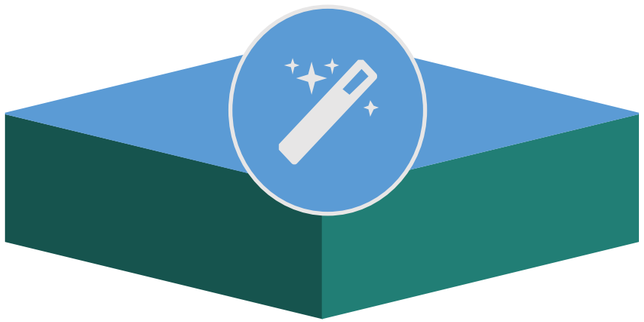
Database Layer

Business Layer

Components of Three-Tier Architecture

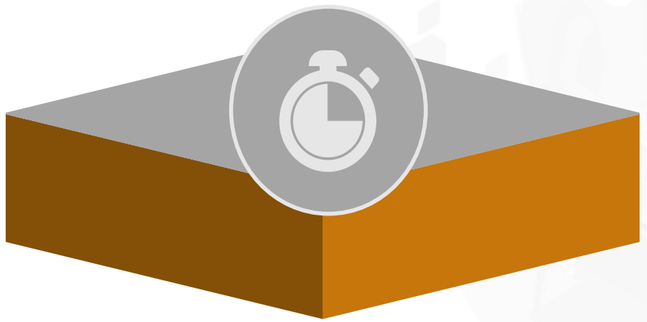
The three components of three-tier architecture are:

Presentation tier



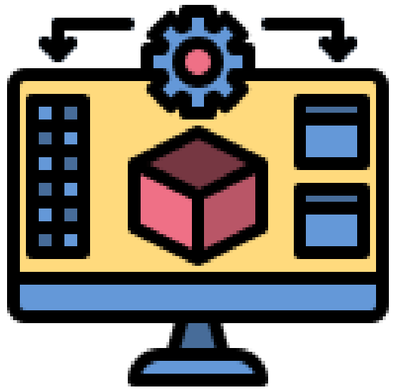
Application tier.

Data tier.



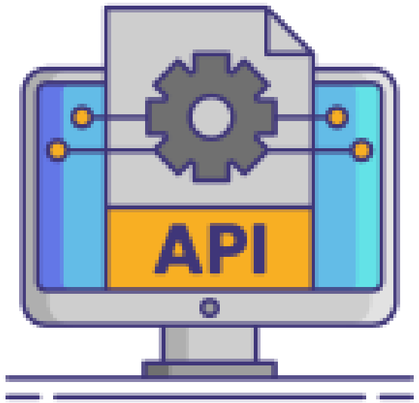
Presentation Tier

The user-interface and communication layer of the application, where the end user interacts with the application.



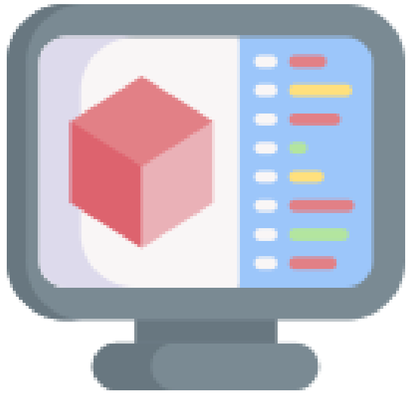
Application Tier

The application tier, also known as the logic tier, is the heart of the application. The application tier can also add, delete, or modify data in the data tier..



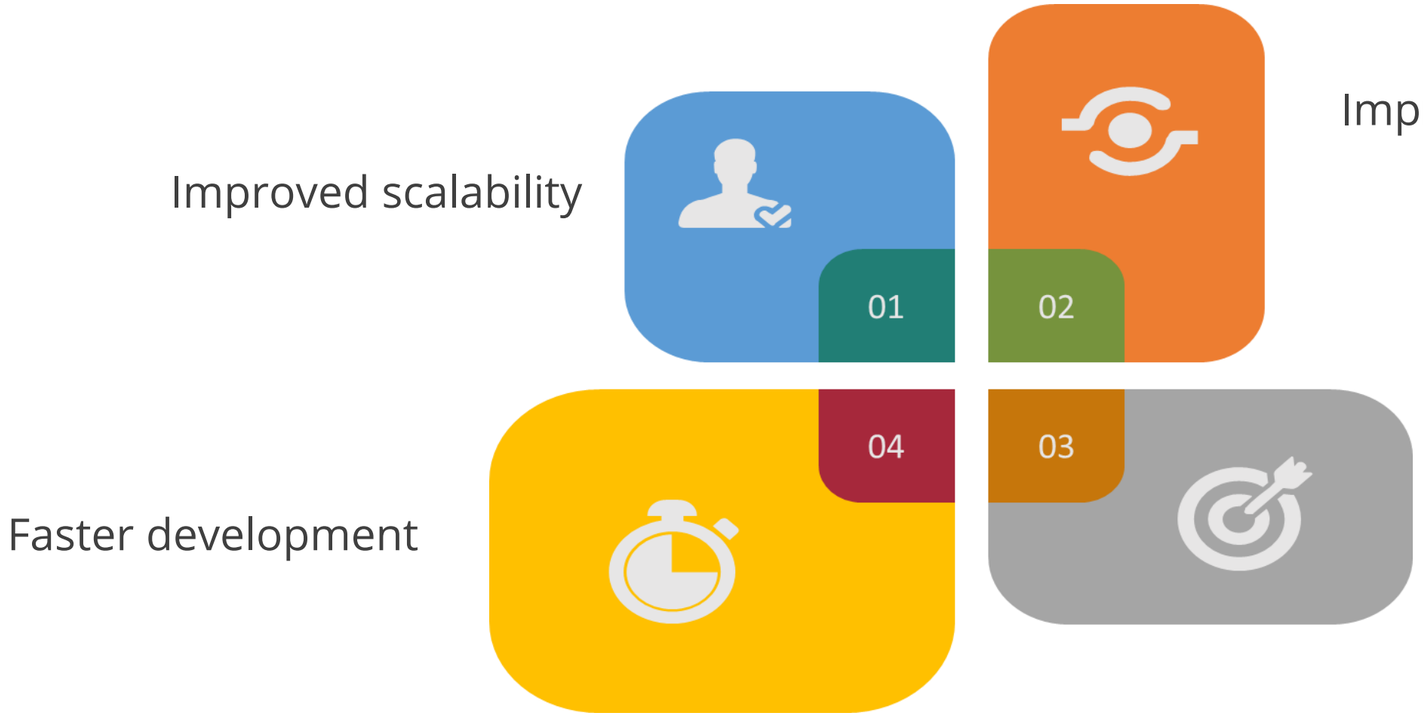
Data Tier

It is also known as database tier. The information processed by the application is stored and managed



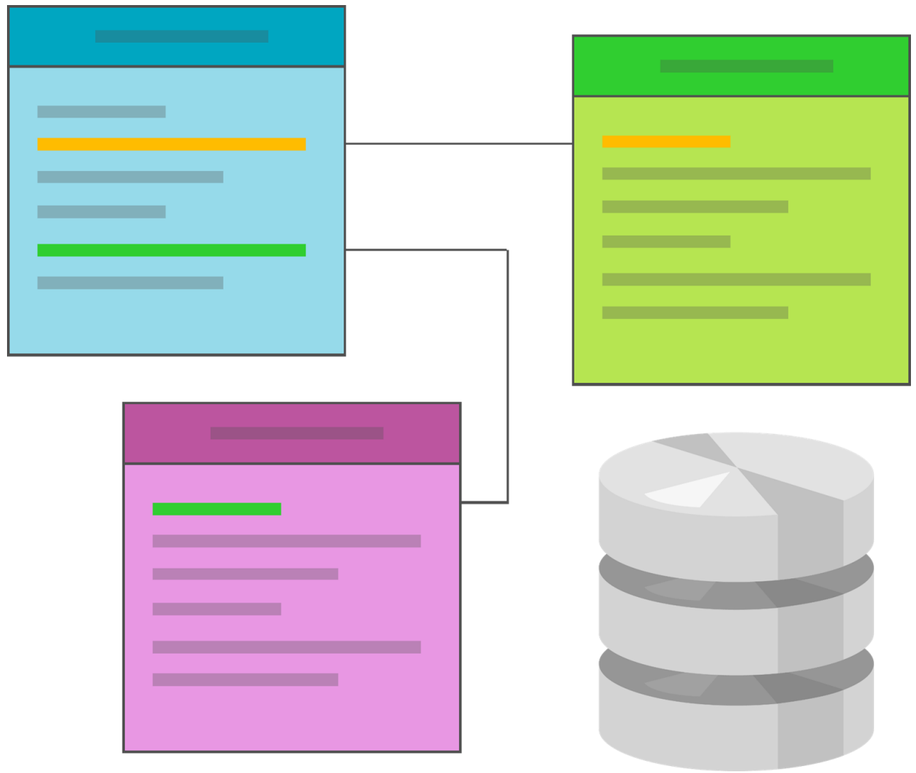
Benefits of Three-tier Architecture

The benefits of three-tier architecture are:



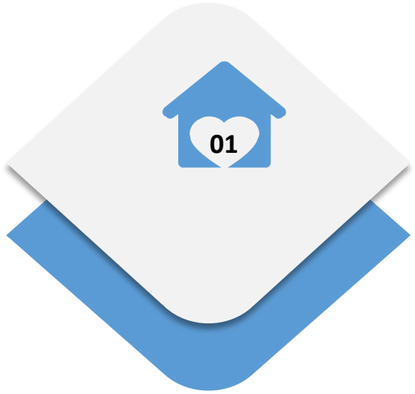
Relational Database

A relational database is a collection of data items with pre-defined relationships between them. These items are organized as a set of tables with columns and rows.



Integrity Rules

Relational tables follow certain integrity rules to ensure that the data they contain stay accurate and accessible.



First, the rows in a relational table should all be distinct. If there are duplicate rows, there can be problems in identifying the correct one.

Second integrity rule in relational model is that column values must not be repeating groups or arrays.

The third aspect of data integrity involves the concept of null values.

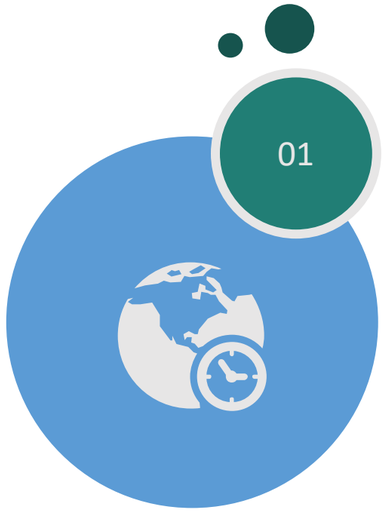
Relational Database Concepts

The department table illustrates some of the relational database concepts.

Dept_no	Dept_name	Location	City
1001	Accounting	New York	Rochester
1002	Research	France	Paris
1003	Sales	London	Camden
1004	Operations	USA	San Antonio
1005	Science	India	Bangalore

It has four rows and six columns, with each row representing a different department. The primary key for this table would generally be the department number because each one is guaranteed to be different.

Types of Relational Database



Select Database

Where Clauses

SELECT Statement

SQL is a language designed to be used with relational databases. SELECT statement is used in RDBMS. It is also known as query, which is used to get information from the table.

Sample Code

```
SELECT Dept_no, Dept_name
FROM Department;

SELECT * FROM Department;
```

WHERE Clauses

The where clause in a SELECT statement provides the criteria for selecting values. The keyword LIKE is used to compare strings.

Sample Code

```
SELECT Dept_no, Dept_name
FROM Department
WHERE Dept_name LIKE 'Research';
```

Joins

The Joins in SQL can easily perform the following:

There must be one column that appears in both the tables in order to relate them with each other.

Suppose, after retrieving the names of the employees who have company cars, one wanted to find out who has which car.

A distinguishing feature of relational database is that it is possible to get data from more than one table, which is called as Join.

Example of Joins

The following code asks for the first and last name of the employees who have company cars and model names of those cars. FROM clause lists both the tables as the requested data is contained in both of them.

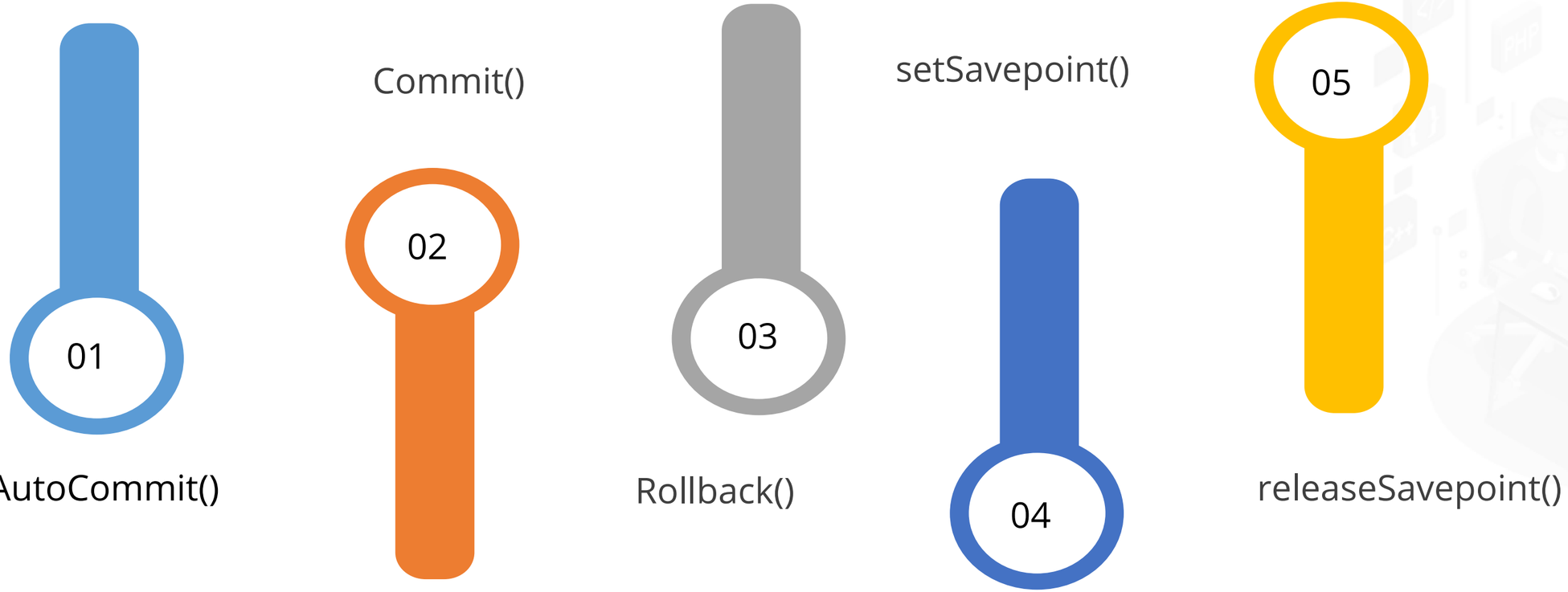
Sample Code

```
SELECT Employees.First_Name,Last_name,
Cars.Model,
From Employees,Cars
Where Employees.Car_No = Cars.Car_Name
```

Syntax of Transaction in JDBC

Syntax of Transaction

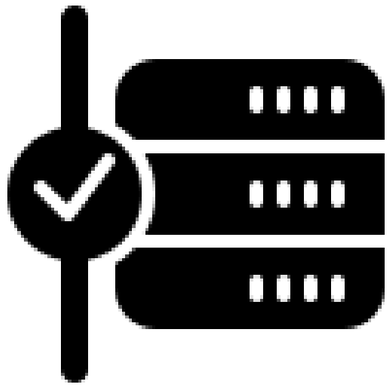
The syntax of transactions are:



setAutocommit()

Syntax

```
public void setAutocommit(boolean value);
```



Values

true to enable autocommit mode for the connection and **false** for disabling it

Example of setAutocommit()

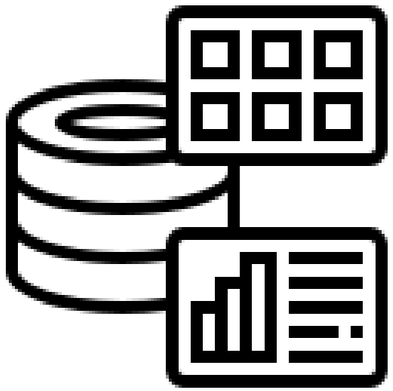
This is the example of setAutocommit() method:

Sample Code

```
Import java.sql.Connection;
Import java.sql.DriverManager;
Import java.sql.Statement;
 public class BatchProcessing_Statement {
   public static void main(String args[])throws Exception {
   String mysql= "jdbc:mysql://localhost/sampleDB";
    Connection con= DriverManager.getConnection(mysqlUrl,"root", "password");
  System.out.println("connection established");
   // create table Dispatches(Product_Name VARCHAR(255), Name_of _the_Customer
VARCHAR(255), Month_of_Dispatch VARCHAR(255, Price INT,Location VARCHAR(255);
     Statement stmt = con.createStatement();
    con.setAutoCommit();
    String insert1= "INSERT INTO Dispatches(Product_Name, Name_of
_the_Customer,Month_of_Dispatch,Price,Location)VALUES" + "('Laptop',
'John','June','20000','Bangalore');
      stmt.addBatch(insert1);
      stmt.executeBatch();
      con.commit();
    System.out.println("Records inserted");
   }
}
```

commit()

To commit the transaction, use the commit() method as:



Syntax

```
con.commit()
```

Example of commit()

This is the example of commit() method:

Sample Code

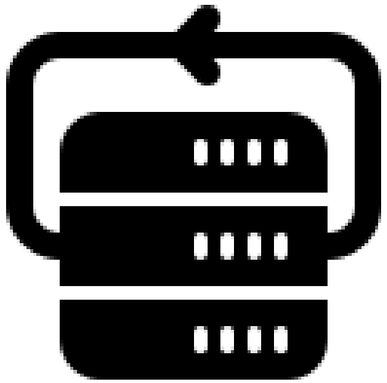
```
Import java.sql.Connection;
Import java.sql.DriverManager;
Import java.sql.Statement;
 public class BatchProcessing_Statement {
   public static void main(String args[])throws Exception {
   String mysql= "jdbc:mysql://localhost/sampleDB";
    Connection con= DriverManager.getConnection(mysqlUrl,"root", "password");

  System.out.println("connection established");

   // create table Dispatches(Product_Name VARCHAR(255), Name_of _the_Customer
VARCHAR(255), Month_of_Dispatch VARCHAR(255, Price INT,Location VARCHAR(255);
     Statement stmt = con.createStatement();
    con.setAutoCommit(false);
    String insert1= "INSERT INTO Dispatches(Product_Name, Name_of
_the_Customer,Month_of_Dispatch,Price,Location)VALUES" + "('Laptop',
'John','June','20000','Bangalore');
      stmt.addBatch(insert1);
      stmt.executeBatch();
      con.commit();
    System.out.println("Records inserted");
   }
}
```

rollback()

To commit the transaction, use the rollback() method as:



Syntax

```
//setting the savePoint//
con.rollback("MysavePoint");
```

Example of rollBack()

This is the example of rollback() method:

Sample Code

```
Import java.sql.Connection
Import java.sql.DriverManager;
Import java.sql.Statement;
 public class BatchProcessing_Statement {
   public static void main(String args[])throws Exception {
   String mysql= "jdbc:mysql://localhost/sampleDB";
    Connection con= DriverManager.getConnection(mysqlUrl,"root", "password");

  System.out.println("connection established");

   // create table Dispatches(Product_Name VARCHAR(255), Name_of _the_Customer VARCHAR(255),
Month_of_Dispatch VARCHAR(255, Price INT,Location VARCHAR(255);
     Statement stmt = con.createStatement();
       con.setAutoCommit(false);
       Resultset rs = con.createStatement(Resultset.TYPE_SCROLL_SENSITIVE,
Resultset.CONCUR_UPDATE);
    String insert1= "INSERT INTO Dispatches(Product_Name, Name_of
_the_Customer,Month_of_Dispatch,Price,Location)VALUES" + "('Laptop',
'John','June','20000','Bangalore');
      stmt.addBatch(insert1);
      stmt.executeBatch();
      con.commit();
      con.rollback()
    System.out.println("Contents of the table");
}}
```

setSavepoint()

Syntax

```
//setting the savePoint//
Savepoint savepoint= con.setSavepoint("Mysavepoint");
```

Example of setSavepoint()

This is the example of setSavepoint() method:

Sample Code

```
Import java.sql.Connection;
Import java.sql.DriverManager;
Import java.sql.Statement;
 public class BatchProcessing_Statement {
   public static void main(String args[])throws Exception {
   String mysql= "jdbc:mysql://localhost/sampleDB";
    Connection con= DriverManager.getConnection(mysqlUrl,"root", "password");

  System.out.println("connection established");

   // create table Dispatches(Product_Name VARCHAR(255), Name_of _the_Customer
VARCHAR(255), Month_of_Dispatch VARCHAR(255, Price INT,Location VARCHAR(255);
     Statement stmt = con.createStatement();
       con.setAutoCommit(false);
       Resultset rs = con.createStatement(Resultset.TYPE_SCROLL_SENSITIVE,
Resultset.CONCUR_UPDATE);
```

Example of setSavepoint()

This is the example of setSavepoint() method:

Sample Code

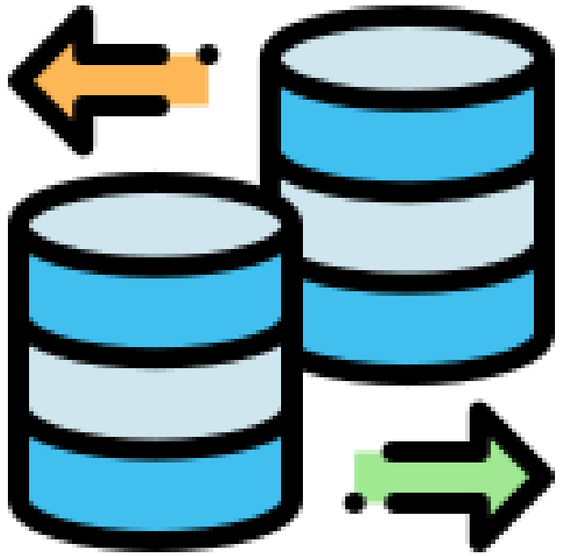
```
String insert 1= "INSERT INTO Dispatches(Product_Name,
Name_of_the_Customer,Month_of_Dispatch,Price,Location)VALUES" +
"('Laptop', 'John','June','20000','Bangalore');
String insert 1= "INSERT INTO Dispatches(Product_Name,
Name_of_the_Customer,Month_of_Dispatch,Price,Location)VALUES" +
"('Laptop', 'John','June','20000','Bangalore');

 Savepoint savepoint = con.setSavepoint("MysavePoint");
      stmt.addBatch(insert1);
      stmt.executeBatch();
      con.commit();
      con,rollback(savePoint);
    System.out.println("Contents of the table");
```

releaseSavepoint()

Syntax

```
//release of the savePoint //
con.releaseSavepoint("MysavePoint");
```



Example of releaseSavepoint()

This is the example of the releaseSavepoint() method:

Sample Code

```
Import java.sql.Connection;
Import java.sql.DriverManager;
Import java.sql.Statement;
 public class BatchProcessing_Statement {
   public static void main(String args[])throws Exception {
   String mysql= "jdbc:mysql://localhost/sampleDB";
    Connection con= DriverManager.getConnection(mysqlUrl,"root", "password");

  System.out.println("connection established");

   // create table Dispatches(Product_Name VARCHAR(255), Name_of _the_Customer
VARCHAR(255), Month_of_Dispatch VARCHAR(255, Price INT,Location VARCHAR(255);
     Statement stmt = con.createStatement();
       con.setAutoCommit(false);
       Resultset rs = con.createStatement(Resultset.TYPE_SCROLL_SENSITIVE,
Resultset.CONCUR_UPDATE);
```

Example of releaseSavepoint()

This is the example of the releaseSavepoint() method:

Sample Code

```
    String insert1= "INSERT INTO Dispatches(Product_Name,
Name_of_the_Customer,Month_of_Dispatch,Price,Location)VALUES" + "('Laptop',
'John','June','20000','Bangalore');
String insert1= "INSERT INTO Dispatches(Product_Name,
Name_of_the_Customer,Month_of_Dispatch,Price,Location)VALUES" + "('Laptop',
'John','June','20000','Bangalore');

Savepoint savepoint = con.setSavepoint("MysavePoint");
      stmt.addBatch(insert1);
      stmt.executeBatch();
      con.commit();
      con,rollback(savepoint);
    System.out.println("Contents of the table");
    rs = stmt.executeQuery("select * from Dispatch");
     System.out.println("");
    con.releaseSavepoint(savepoint);
   System.out.println( " Save point released");
   }
}
```

Key Takeaways

- To maintain the atomicity, consistency, and durability, database integrity transactions are useful.
- The relational tables follow certain integrity rules to ensure that the data they contain stay accurate and accessible.
- The transaction ensures that the other connections to the same database see either all the updates or none of them.
- The two-tier architecture is a software in which an interface runs on a client and data structure gets stored on a server.