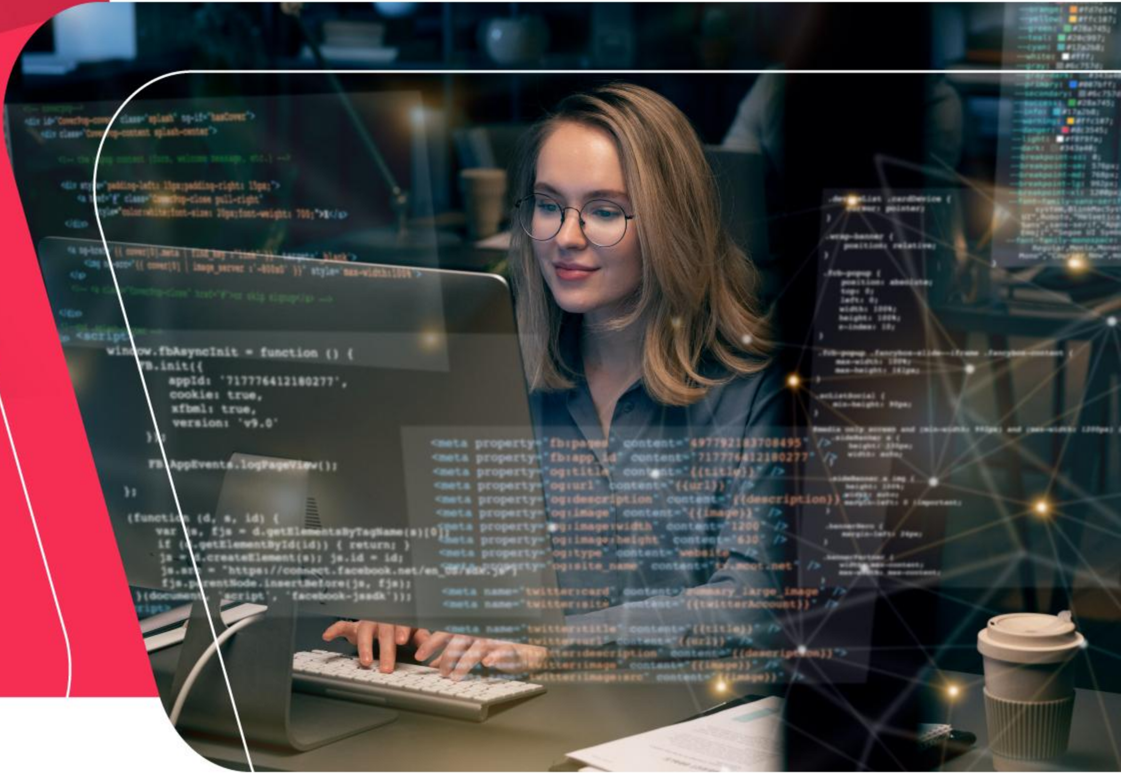
TIA-Academy

Java Bootcamp

Day 28



Technologies will be Use

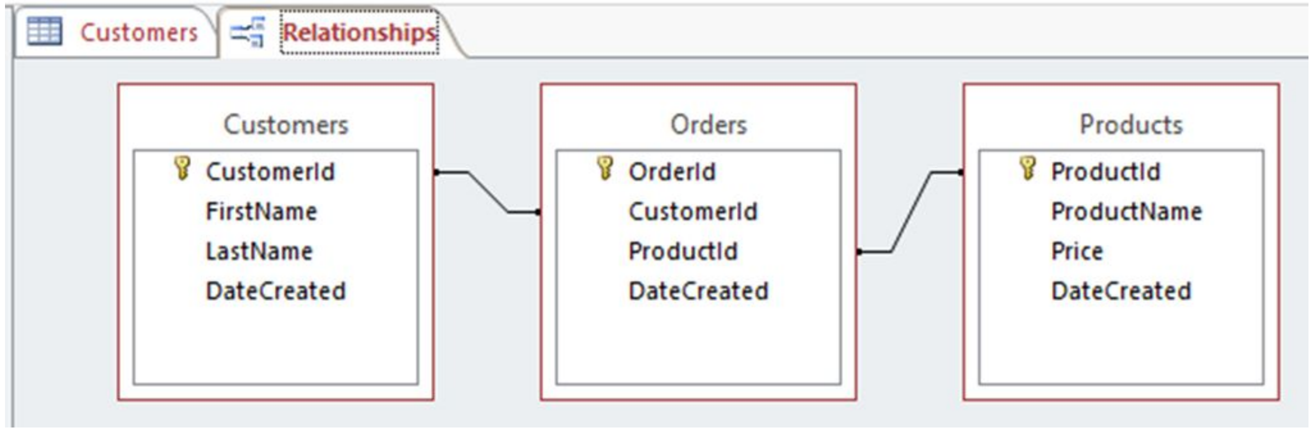
- JDK 8/**11**/15
- JRE 8/**11**/15
- **Intellij IDEA Community Edition**
- JAVA 3rd Party Library (Network, DB, etc)
- **MySQL Server Community**
- MySQL Workbench or SQLyog (Windows)

DATABASE RELATION

TIA-Academy

What Is a Database Relationship?

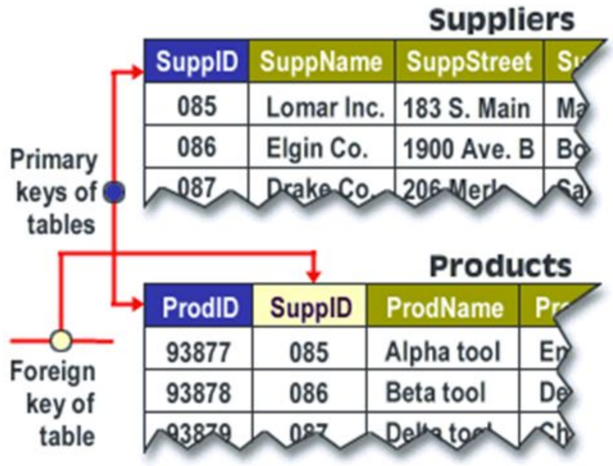
- A relationship is established between two database tables when one table uses a **foreign key** that references the **primary key** of another table.
- This is the basic concept behind the term relational database.



TIA-Academy

What Is a Primary Key?

- What is a primary key? In the world of databases, the primary key of a relational table **uniquely identifies each record in the table**.
- Databases use keys to **compare, sort, and store records, and to create relationships between records**.



Suppliers

SuppID	SuppName	SuppStreet	Su
085	Lomar Inc.	183 S. Main	Ma
086	Elgin Co.	1900 Ave. B	Bo
087	Drake Co.	206 Merl	Sa

Products

ProdID	SuppID	ProdName	Pr
93877	085	Alpha tool	En
93878	086	Beta tool	De
93879	087	Delta tool	Ch

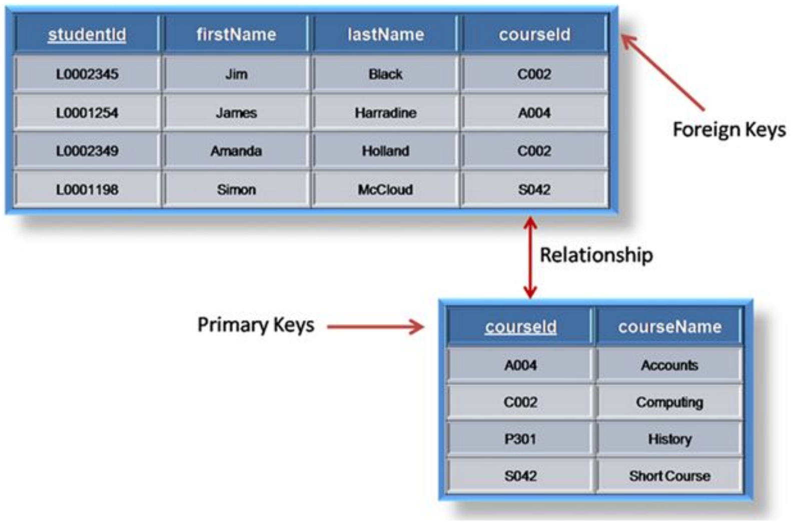
What Is a Primary Key?

- Choosing the primary key in a database is one of the most important steps in the process.
- It can be a normal attribute that is guaranteed to be unique such as Social Security number on a table with no more than one record per person or — preferably — it can be generated by the database management system such as a globally unique identifier, or GUID, in Microsoft SQL Server.
- Primary keys may consist of a single attribute or multiple attributes in combination.

TIA-Academy

What Is a Primary Key?

- Primary keys are the unique links to related information in other tables where the primary key is used. **It must be entered when a record is created, and it should never be changed.**
- Each table in the database has a column or two specifically for the primary key.



studentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042

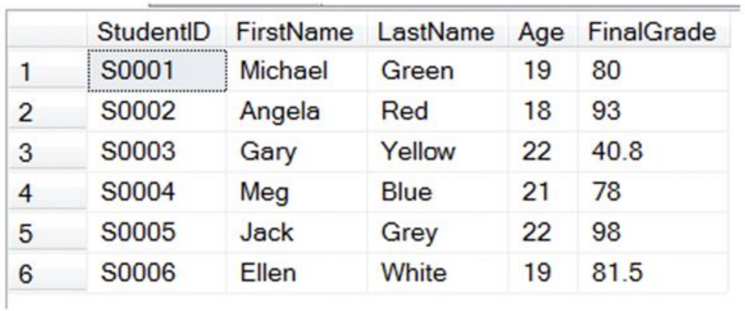
courseId	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short Course

TIA-Academy

Primary Key Example

- Imagine you have a **STUDENTS table that contains a record for each student at a university.**
- The student's unique student ID number is a good choice for a primary key in the STUDENTS table.
- The student's **first and last name are not good choices** because there is always the chance that more than one student might have the same name.

Student table:



	StudentID	FirstName	LastName	Age	FinalGrade
1	S0001	Michael	Green	19	80
2	S0002	Angela	Red	18	93
3	S0003	Gary	Yellow	22	40.8
4	S0004	Meg	Blue	21	78
5	S0005	Jack	Grey	22	98
6	S0006	Ellen	White	19	81.5

TIA-Academy

Primary Key Example

- Other poor choices for primary keys include ZIP code, email address, and employer, all of which can change or represent many people. The identifier used as a primary key must be unique.
- Even Social Security Numbers can change when the Social Security Administration reassigns a number to someone who has been affected by identity theft. Some people don't even have a Social Security Number.
- However, because both of those cases are **rare**. Social Security Numbers can be a good choice for a primary key.

Tips for Choosing Good Primary Keys

When you choose the right primary key, database lookups are speedy and reliable. Just remember:

- **Keep it short**. Because the **primary key is used for lookups and comparisons**, a short primary key means the database management system can process it more quickly than a long primary key.
- **Use a number** for the primary key whenever possible. **SQL Server or other database management systems process number data types faster** than character data types.
- **Keep it simple**. Don't use any special characters, embedded spaces, or a mix of upper and lower capitalization.
- **Never change** the primary key after you assign it.

Foreign Keys

- Database designers make wide use of keys when developing relational databases.
- Among the most common of these keys are **primary keys** and **foreign keys**.
- A database **foreign key is a field in a relational table that matches the primary key column of another table**.
- To understand how a foreign key works, let's take a closer look at the idea of a relational database.

TIA-Academy

Some Basics of Relational Databases

- In a relational database, **data is stored in tables containing rows and columns**, making it easy to search and manipulate.
- There is some serious math behind the concept of a relational database (relational algebra, proposed by E.F. Codd at IBM in 1970), but that's not the topic of this article.

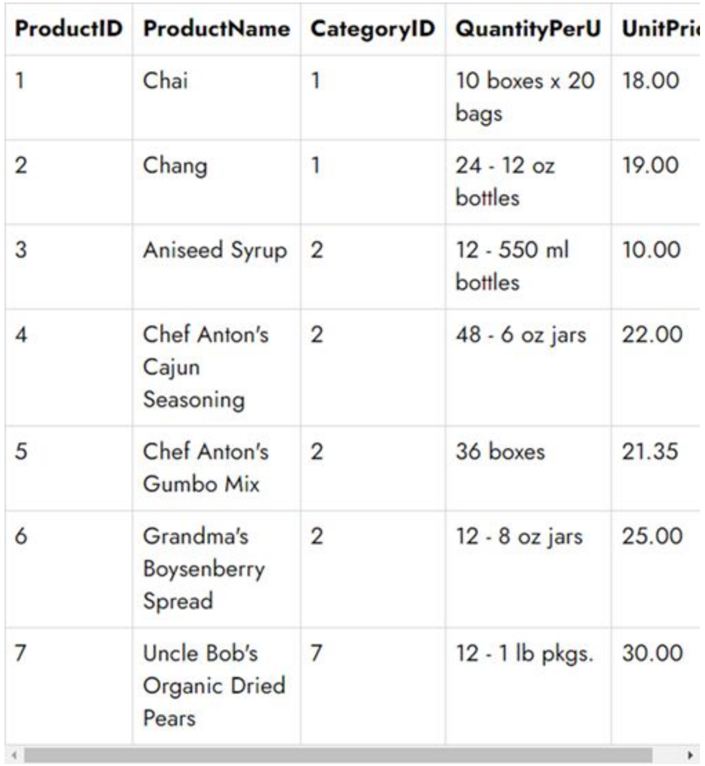
TIA-Academy

Some Basics of Relational Databases

- For practical purposes (and non-mathematicians), **a relational database stores related data in rows and columns**.
- Further — and here's where it gets interesting — **most databases are designed so that the data in one table can access the data in another table**.
- This ability to create relationships between tables is the real power of a relational database.

Some Basics of Relational Databases

- Most tables, especially those in large, complex databases, have primary keys.
- Tables that are designed to access other tables must also have a foreign key.
- To use the commonly-cited Northwinds database, here is an excerpt from a Product table:



ProductID	ProductName	CategoryID	QuantityPerU	UnitPri
1	Chai	1	10 boxes x 20 bags	18.00
2	Chang	1	24 - 12 oz bottles	19.00
3	Aniseed Syrup	2	12 - 550 ml bottles	10.00
4	Chef Anton's Cajun Seasoning	2	48 - 6 oz jars	22.00
5	Chef Anton's Gumbo Mix	2	36 boxes	21.35
6	Grandma's Boysenberry Spread	2	12 - 8 oz jars	25.00
7	Uncle Bob's Organic Dried Pears	7	12 - 1 lb pkgs.	30.00

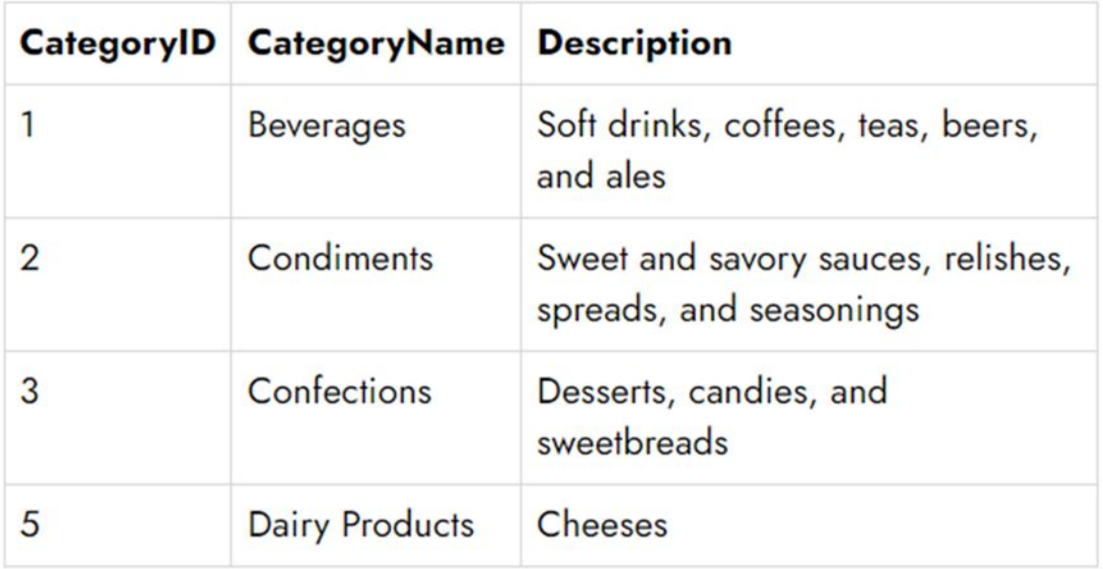
Using Foreign Keys

- The **ProductID** column is this table's **primary key.** It assigns a unique ID to each product.
- This table also contains a **foreign key column, CategoryID**. Every product in the Product table links to an entry in the Categories table that defines that product's category.

TIA-Academy

Using Foreign Keys

- Note this excerpt from the database's Categories table:



CategoryID	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweetbreads
5	Dairy Products	Cheeses

Northwind Database's Categories Table Excerpt

Using Foreign Keys

- The column **CategoryID is this column's primary key**. (It has no foreign key because it has no need to access another table.) **Every foreign key in the Product table links to a primary key in the Categories table.**
- For example, the product Chai is assigned a category "Beverages", while Aniseed Syrup is in the category Condiments.
- This kind of linking creates a myriad of ways to use and re-use data in a relational database.

Database Table Relationship

- In a relational database, a relationship is formed by correlating rows belonging to different tables.
- A **table relationship is established when a child table defines a Foreign Key column that references the Primary Key column of its parent table.**

TIA-Academy

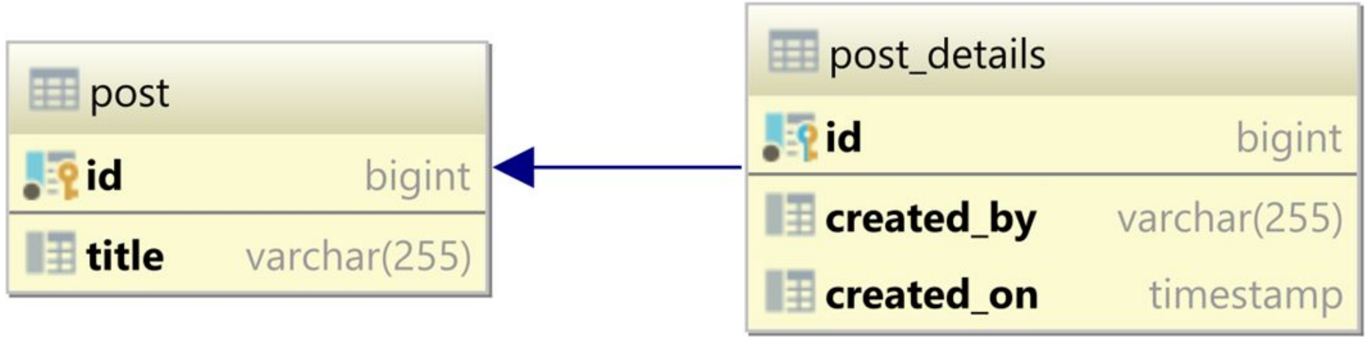
Database Table Relationship

Every database table relationship is, therefore, built on top of Foreign Key columns, and there can be three table relationship types:

- *one-to-one* requires the child table Primary Key to be associated via a Foreign Key with the parent table Primary Key column.
- *one-to-many* is the most common relationship, and it associates a row from a parent table to multiple rows in a child table.
- *many-to-many* requires a link table containing two Foreign Key columns that reference the two different parent tables.

One-To-One

- The one-to-one table relationship looks as follows:



One-To-One

- In a relational database system, a **one-to-one table relationship links two tables based on a Primary Key column in the child which is also a Foreign Key referencing the Primary Key of the parent table row.**
- Therefore, we can say that the child table shares the Primary Key with the parent table.
- In the table diagram above, the id column in the post_details table has also a Foreign Key relationship with the post table id Primary Key column:

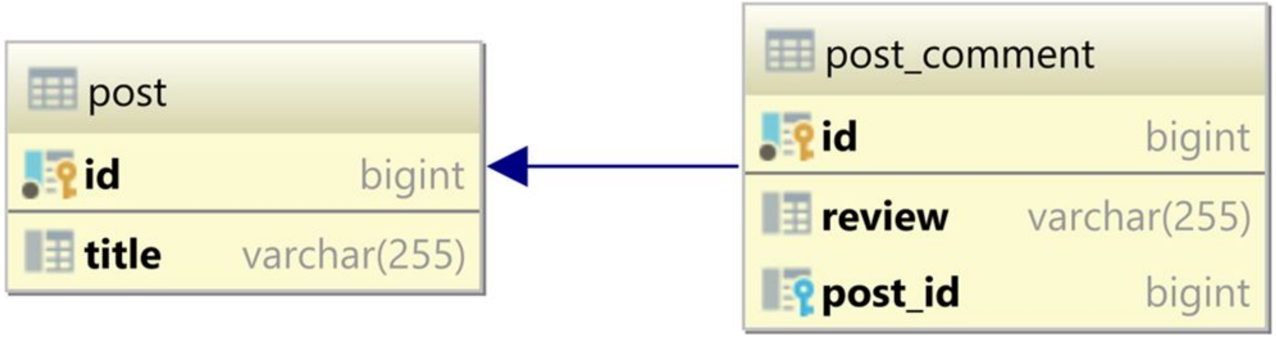
```
ALTER TABLE
    post_details
ADD CONSTRAINT
    fk_post_details_id
FOREIGN KEY (id) REFERENCES post (id)
```

TIA-Academy

ASSIGNMENT 01

One-To-Many

- The one-to-many table relationship looks as follows:



One-To-Many

- In a relational database system, a **one-to-many table relationship links two tables based on a Foreign Key column in the child which references the Primary Key of the parent table row**.
- In the table diagram above, **the post_id column in the post_comment table has a Foreign Key relationship with the post table id Primary Key** column:

```
ALTER TABLE
    post_comment
ADD CONSTRAINT
    fk_post_comment_post_id
FOREIGN KEY (post_id) REFERENCES post (id)
```

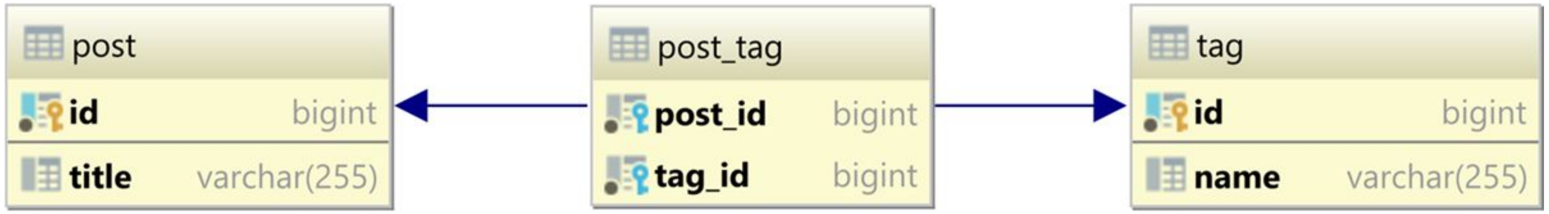
TIA-Academy

ASSIGNMENT 02

TIA-Academy

Many-To-Many

- The many-to-many table relationship looks as follows:



post	
id	bigint
title	varchar(255)

post_tag	
post_id	bigint
tag_id	bigint

tag	
id	bigint
name	varchar(255)

Many-To-Many

- In a relational database system, **a many-to-many table relationship links two parent** tables **via a child table** which contains two Foreign Key columns referencing the Primary Key columns of the two parent tables.
- In the table diagram above, the post_id column in the post_tag table has also a Foreign Key relationship with the post table id Primary Key column:

```
ALTER TABLE
    post_tag
ADD CONSTRAINT
    fk_post_tag_post_id
FOREIGN KEY (post_id) REFERENCES post (id)
```

TIA-Academy

Many-To-Many

- And, the tag_id column in the post_tag table has a Foreign Key relationship with the tag table id Primary Key column:

```
ALTER TABLE
    post_tag
ADD CONSTRAINT
    fk_post_tag_tag_id
FOREIGN KEY (tag_id) REFERENCES tag (id)
```

TIA-Academy

ASSIGNMENT 03

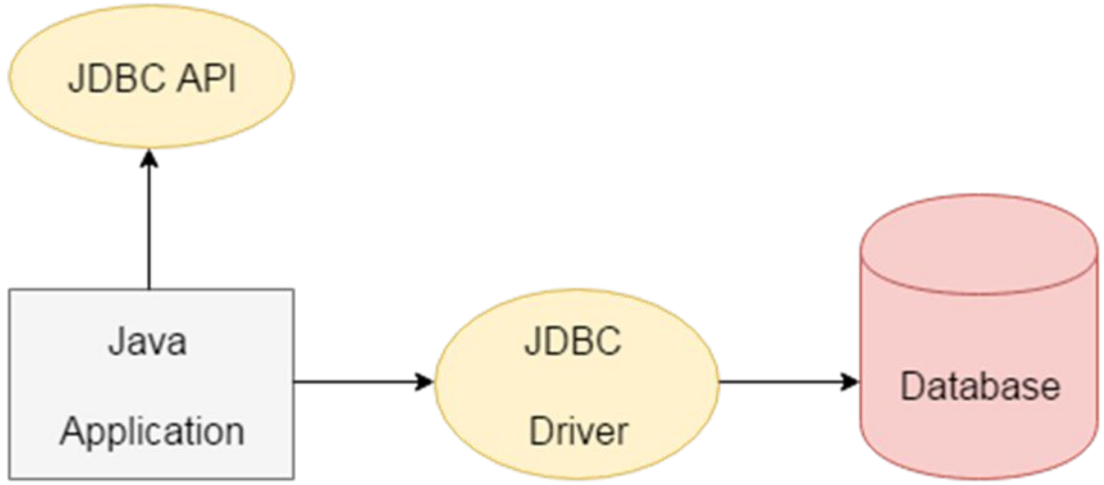
TIA-Academy

Manipulating Database with JDBC

TIA-Academy

Java JDBC

- Java JDBC is a java API to **connect and execute query** with the database. JDBC API uses jdbc drivers to connect with the database.



Why use JDBC

- Before JDBC, ODBC API was the database API to connect and execute query with the database.
- But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured).
- That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

What is API

- **API (Application programming interface)** is a document that contains description of all the features of a product or software.
- It represents classes and interfaces that software programs can follow to communicate with each other.
- An API can be created for applications, libraries, operating systems, etc

TIA-Academy

JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. **Thin driver (fully java driver)**

Thin driver

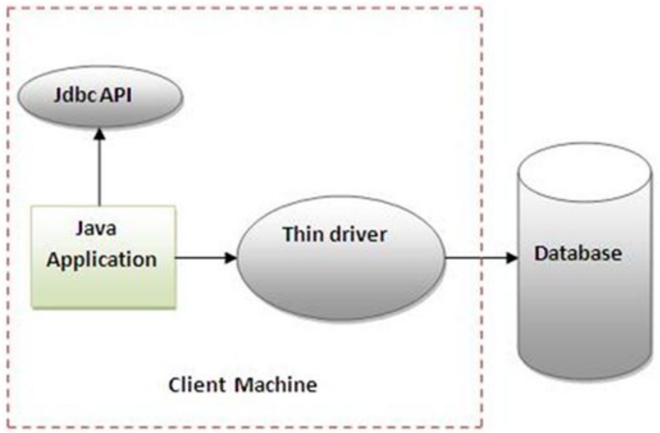


Figure- Thin Driver

- The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver.
- It is fully written in Java language.

Thin driver

Advantages:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantages:

- Drivers depends on the Database.

5 Steps to connect to the database in java

There are 5 steps to connect any java application with the database in java using JDBC.

They are as follows:

1. Register the driver class
2. Creating connection
3. Creating statement
4. Executing queries
5. Closing connection

TIA-Academy

1. Register the driver class

- The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.
- Syntax of forName() method

```
public static void forName(String className) throws ClassNotFoundException
```

- Example to register the OracleDriver class

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2. Create the connection object

- The getConnection() method of DriverManager class is used to establish connection with the database.
- Syntax of getConnection() method

```
public static Connection getConnection(String url)throws SQLException
public static Connection getConnection(String url,String name,String password)
throws SQLException
```

- Example to establish connection with the Oracle database

```
Connection con=
DriverManager.getConnection( "jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

TIA-Academy

3. Create the Statement object

- The createStatement() method of Connection interface is used to create statement.
- The object of statement is responsible to execute queries with the database.
- Syntax of createStatement() method

```
public Statement createStatement()throws SQLException
```

- Example to create the statement object

```
Statement stmt=con.createStatement();
```

TIA-Academy

3. Create the Statement object

- The createStatement() method of Connection interface is used to create statement.
- The object of statement is responsible to execute queries with the database.
- Syntax of createStatement() method

```
public Statement createStatement()throws SQLException
```

- Example to create the statement object

```
Statement stmt=con.createStatement();
```

TIA-Academy

4. Execute the query

- The executeQuery() method of Statement interface is used to execute queries to the database.
- This method returns the object of ResultSet that can be used to get all the records of a table.
- Syntax of executeQuery() method

```
public ResultSet executeQuery(String sql)throws SQLException
```

- Example to create the statement object

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next()){
    System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

5. Close the connection object

- By closing connection object statement and ResultSet will be closed automatically.
- The close() method of Connection interface is used to close the connection.
- Syntax of close() method

```
public void close()throws SQLException
```

- Example to close connection

```
con.close();
```

Connect to the mysql database in java

- For connecting java application with the mysql database, you need to follow steps to perform database connectivity.

- In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.

Connect to the mysql database in java

2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** 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 sonoo is the database name. We may use any database, in such case, you need to replace the sonoo with your database name.
3. **Username:** The default username for the mysql database is **root**.
4. **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.

Create Database

- Let's first create a table in the mysql database, but before creating table, we need to create database first.

```
create database sonoo;
use sonoo;
create table emp(id int(10),name varchar(40),age int(3));
```

TIA-Academy

Example to Connect Java Application with

```
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);}
    }
}
```

TIA-Academy

MySQL Connector

- To connect java application with the mysql database mysqlconnector.jar file is required to be loaded. But, first download it :
- https://dev.mysql.com/downloads/connector/j/

TIA-Academy

MySQL Connector

Two ways to load the jar file:

1. paste the mysqlconnector.jar file in JRE/lib/ext folder:

   Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.

2. set classpath:

   There are two ways to set the classpath:

   - temporary
   - permanent

MySQL Connector

- How to set the temporary classpath
  - open command prompt and write:
  - C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;

- How to set the permanent classpath
  - Go to environment variable then click on new tab. In variable name write classpath and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;.; as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;

TIA-Academy

ASSIGNMENT 04

TIA-Academy

Transaction Management in JDBC

TIA-Academy

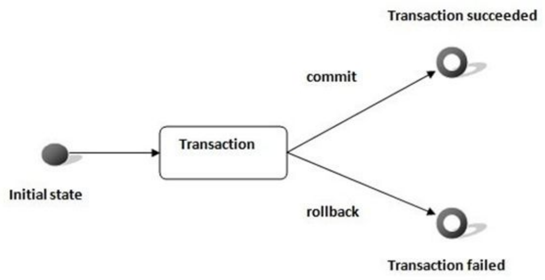
Transaction Management in JDBC

- 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.

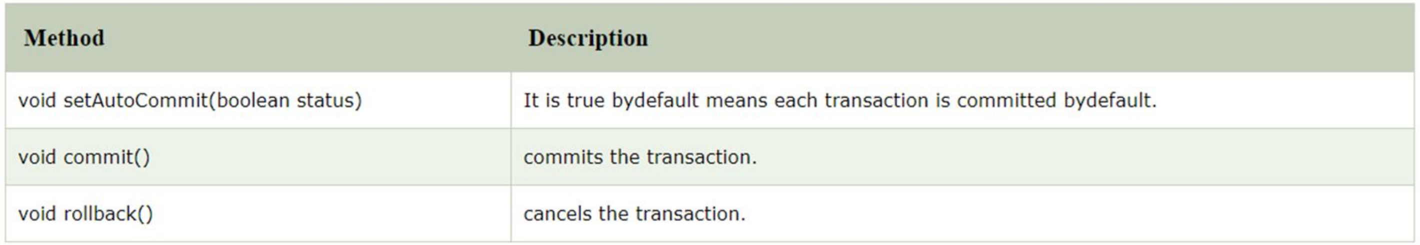
TIA-Academy

Advantage of Transaction Management

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



- In JDBC, **Connection interface** provides methods to manage transaction.



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.

Simple example of transaction management in jdbc using Statement

```
import java.sql.*;
class FetchRecords{
      public static void main(String args[])throws Exception{
      Class.forName(“mysql.jdbc.driver.OracleDriver");
      Connection con=DriverManager.getConnection("jdbc:mysql: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',50000)");

      con.commit();
      con.close();
      }
}
```

Simple example of transaction management in jdbc using Statement

```
import java.sql.*;
import java.io.*;
class TM{
        public static void main(String args[]){
        try{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
        con.setAutoCommit(false);

        PreparedStatement ps=con.prepareStatement("insert into user420 values(?,?,?)");

        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        while(true){

                System.out.println("enter id");
                String s1=br.readLine();
                int id=Integer.parseInt(s1);
```

Simple example of transaction management in jdbc using Statement

```
System.out.println("enter name");
String name=br.readLine();

System.out.println("enter salary");
String s3=br.readLine();
int salary=Integer.parseInt(s3);

ps.setInt(1,id);
ps.setString(2,name);
ps.setInt(3,salary);
ps.executeUpdate();

System.out.println("commit/rollback");
String answer=br.readLine();
if(answer.equals("commit")){
        con.commit();
}
if(answer.equals("rollback")){
        con.rollback();
}
```

Simple example of transaction management in jdbc using Statement

```
                    System.out.println("Want to add more records y/n");
                    String ans=br.readLine();
                    if(ans.equals("n")){
                    break;
                    }
          }
    con.commit();
    System.out.println("record successfully saved");

    con.close();//before closing connection commit() is called
    }catch(Exception e){System.out.println(e);}


    }

}
```

TIA-Academy

ASSIGNMENT 05 (HOME ASSIGNMENT)

TIA-Academy

Thank You