SPRING WITH DATABASE

* SQL

1. INTRUDUCTION

* JDBC

1. INTRODUCTION
2. CRUD OPERATION
3. DEVELOPMENT APPROACH
4. BATCH UPDATION
5. TRANSACTION MANAGEMENT
6. DATASOURCE & CONNECTION POOLING

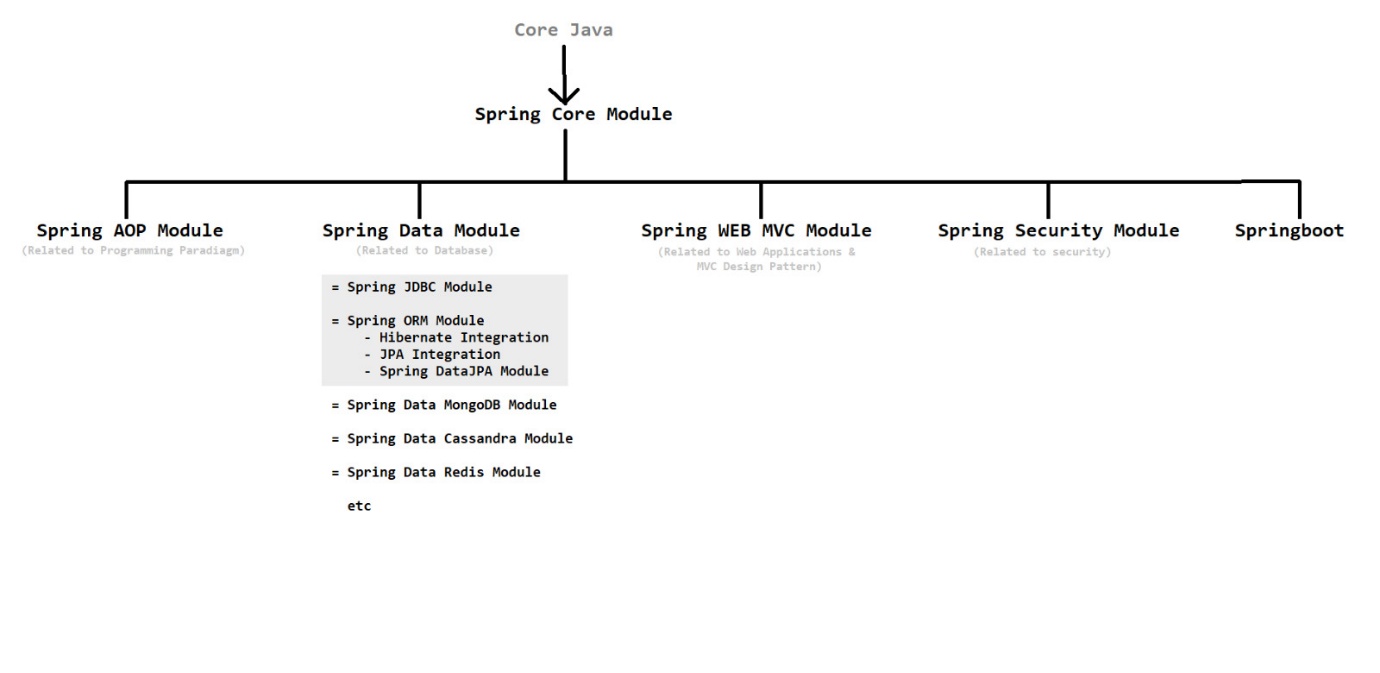
* SPRING JDBC
* HIBERNATE
* SPRING HIBERNATE
* JPA
* SPRING JPA
* SPRING DATA JPA

**SQL**

* DATABASE INTRODUCTION

1. A database is an organized collection of structured information or data typically stored electronically in computer system
2. A database is usually controlled by "Database Management System (DBMS)"
3. Few examples of DBMS are :- MySQL , Microsoft SQL Server , PostgreSQL , SQLite , MongoDB , Cassandra , DB2
4. To store the data in database we have to use one query language i.e. SQL, Oracle etc

* DIAGRAM



* SQL INTRODUCTION

1. SQL is a query language which is used to access or manipulate the database i.e. we can insert, update, delete, select etc. the data or information from database
2. Database is a collection of data in structured form

* DATA TYPES

1. String data-types : VARCHAR(size), CHAR(size), TEXT(size), MEDIUMTEXT(size), BLOB, CLOB etc
2. Numeric data-types : INT, FLOAT, DECIMAL, DOUBLE etc
3. Date & Time data-types : DATE, DATETIME, TIME, TIMESTAMP etc

* OPERATOR

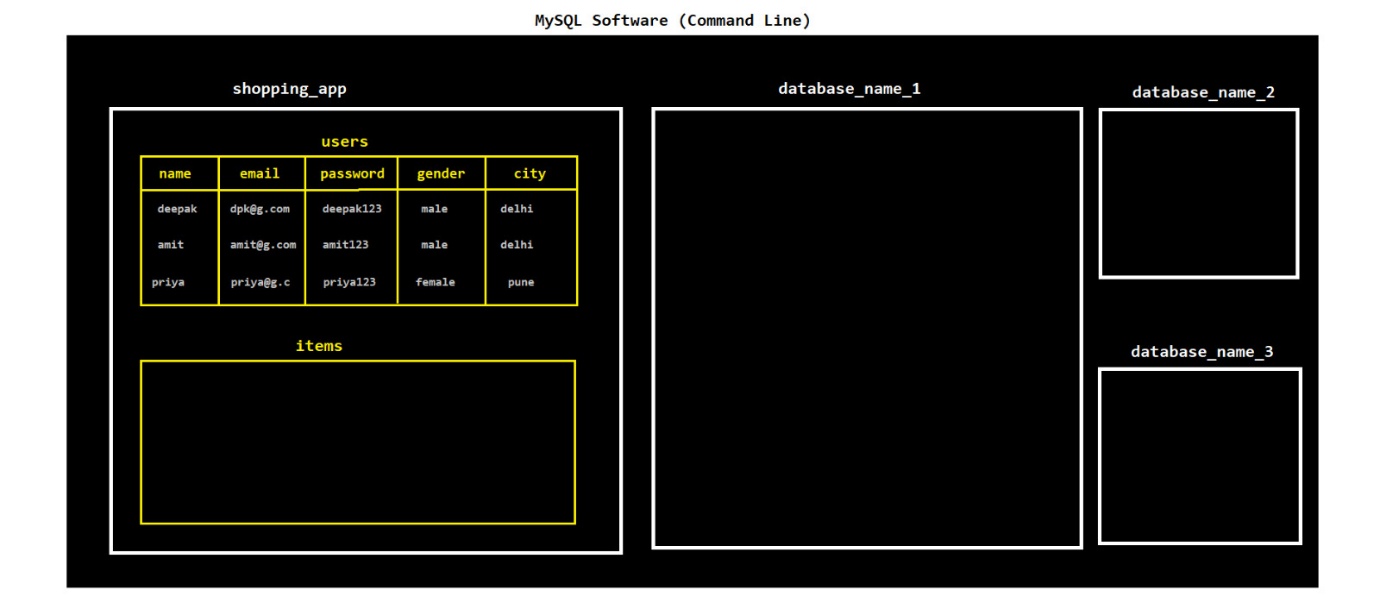
1. SQL Arithmetic Operators : +, -, \*, /, %
2. SQL Comparison Operators : =, !=, >, <, >=, <= etc
3. SQL Logical Operators : AND, OR, NOT, LIKE, BETWEEN etc

* COMMAND TYPE

There are various types of commands some of them are as follow :-

1. DDL (Data Definition Language) :- CREATE , ALTER , DROP , RENAME , …
2. DML (Data Manipulation Language) :- INSERT , UPDATE , DELETE , …
3. DQL (Data Query Language) :- SELECT , …
4. TCC (Transaction Control Commands) :- COMMIT , ROLLBACK , SAVEPOINT , …

* DIAGRAM



* DDL

Following are the some of the command used in DDL :-

1. SHOW DATABASES;
2. SHOW TABLES;
3. CREATE DATABASE database\_name;
4. DROP DATABASE database\_name;
5. USE database\_name;
6. CREATE TABLE table\_name(column\_name DATA-TYPE(size), column\_name DATA-TYPE(size), -, -, -);
7. CREATE TABLE table\_name(ID int NOT NULL, column\_name VARCHAR(100), column\_name INT, PRIMARY KEY(ID));
8. DESC table\_name;
9. ALTER TABLE old\_table\_name RENAME To new\_table\_name;
10. ALTER TABLE table\_name ADD column\_name DATA-TYPE(size);
11. ALTER TABLE table\_name DROP COLUMN column\_name;
12. DROP TABLE table\_name;

* DQL

Following are the some of the command used in DQL :-

1. SELECT \* FROM table\_name;
2. SELECT \* FROM table\_name WHERE column\_name='value';
3. SELECT col\_name, col\_name, - , - FROM table\_name;
4. SELECT col\_name, col\_name, - , - FROM table\_name WHERE column\_name='value';
5. SELECT col\_name, col\_name, - , - FROM table\_name WHERE col\_name='value' AND col\_name ='value';

* DML

Following are the some of the command used in DML :-

1. INSERT INTO table\_name VALUES('value1', 'value2', 'value3', -, -);
2. INSERT INTO table\_name(col\_name, col\_name, col\_name) VALUES('value1', 'value2', 'value3');
3. UPDATE table\_name SET col\_name='value1', col\_name='value2', -, - WHERE col\_name='value3';
4. DELETE FROM table\_name WHERE column\_name='value';

* NOTE POINTS

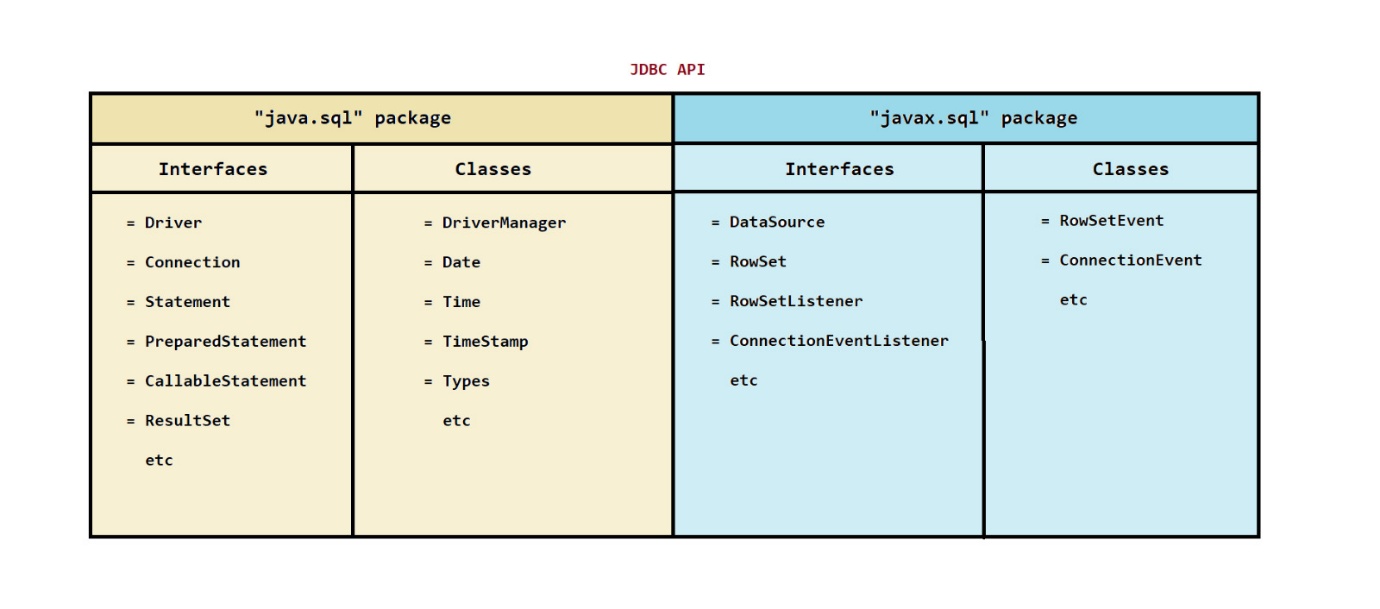
1. BLOB – Binary Large Object. It is used to store images , videos and files as they are large in size. CLOB – Character Large object. It is used to store large characters.
2. INSERT me values utni hi rhni chahiye jitne tables me column hai. Sirf specific col m insert Krna h to use insert with col name OR insert with blank string for empty field.
3. Bina where k update use Kiya to saare rows pr changes implement ho jayenge.
4. Ager sirf kuch columns chahiye then instead of \* in select specify the col name.
5. To check the current MySQL connection use STATUS; or SELECT USER(), @@hostname, @@port;

**JDBC**

**INTRODUCTION**

* JDBC

1. JDBC stands for Java Database Connectivity
2. JDBC is a technology which is used to interact the java application with database. **[** It is an API that provides a set of classes & interface that allows java application to interact with database **]** JDBC is also an API (pre-defined interfaces, classes and packages).
3. Diagram



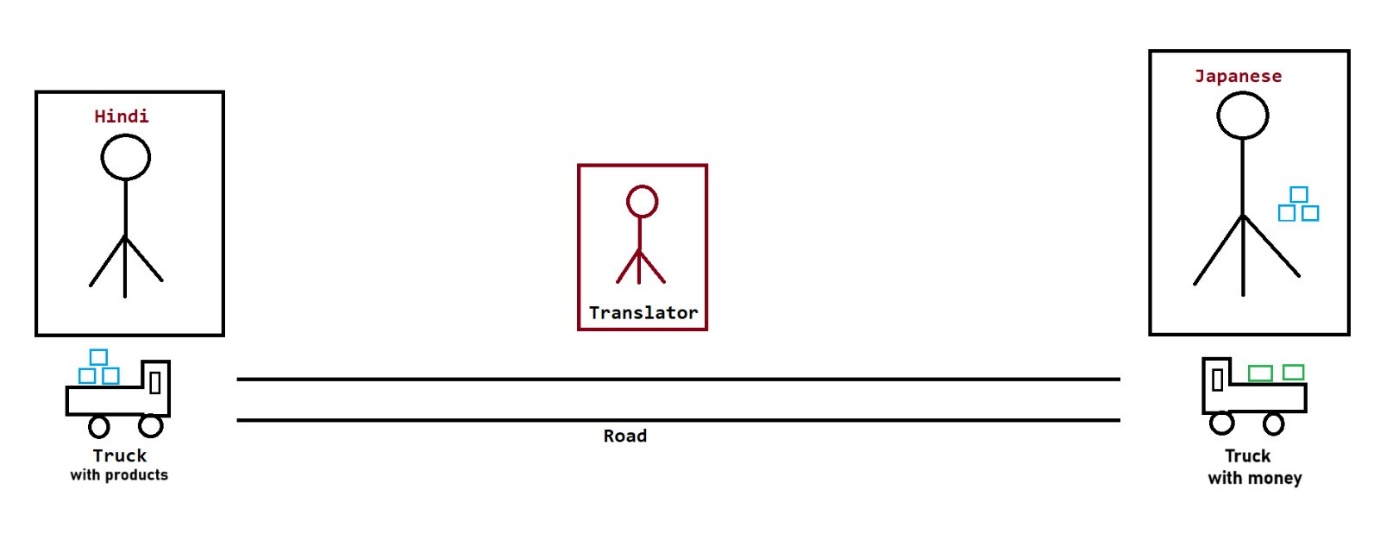
1. JDBC is an abstraction which is provided by Sun-Microsystems and implemented by database vendors (and provide them in the form of jar files) **[** JDBC API was created by SunMicrosystem and the implementation for those API are provided by DB Vendor (in form of jar files) hence we also call JDBC an Abstraction. **]**

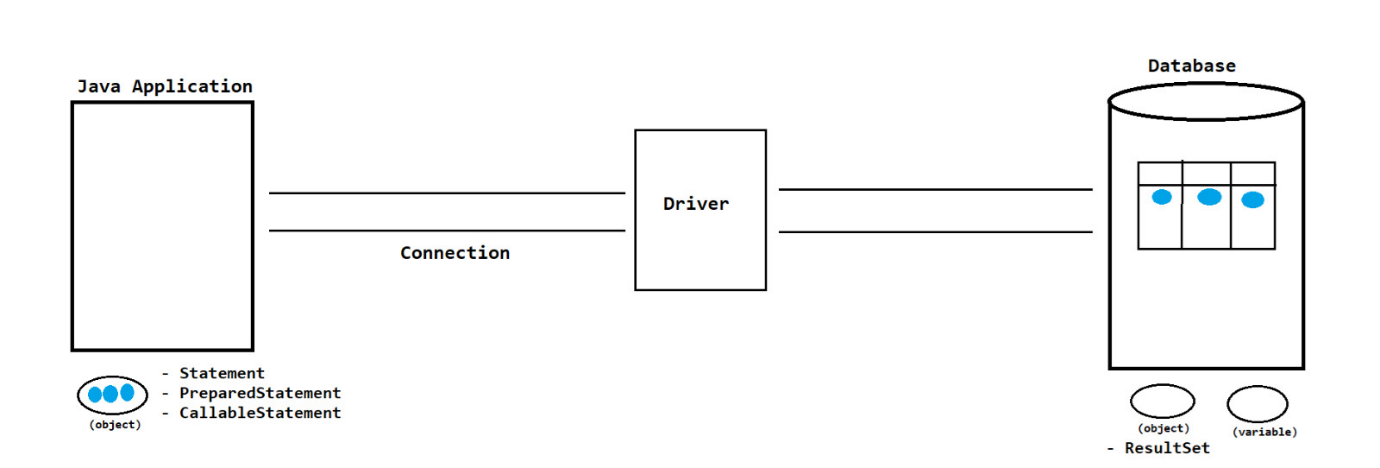
* JDBC COMPONENT

1. Component

There are 5 component in the JDBC that are Driver , DriverManager , Connection , Statement and ResultSet

1. Diagram





1. Driver

* It is responsible to translate JDBC specific calls to DB specific calls and vice versa.
* Hence it is used to convert the java calls to database specific calls and database specific calls to java calls

1. DriverManager

* DM is a class present in java.sql package which is responsible for managing the driver by registering and unregistering it and to establish connection with the DB with the help of Driver.

1. Connection

* By using Connection, Java Application can communicate with Database.
* It is used to create connection between java application and database.

1. Statement / PreparedStatement / CallableStatement

* By using Statement Object we can send our SQL Query to the Database and we can get Results from Database.
* It is used to send the SQL Queries (with data) from java application to database and get the result

1. ResultSet

* Whenever we are executing Select Query, Database engine will provide Result in the form of ResultSet. Hence ResultSet holds Results of SQL Query.
* It is used to store the output from database.
* Types of Drivers

1. JDBC-ODBC Bridge Driver

* Internally this Driver will take Support of ODBC Driver to communicate with Database.
* Type-1 Driver converts JDBC Calls (Java Calls) into ODBC Calls and ODBC Driver converts ODBC Calls into Database specific Calls.

1. Native API Driver

* Type-2 Driver internally uses Vendor specific Native Libraries to Communicate with Database.
* Type-2 Driver converts JDBC Calls into Vendor specific Native Library Calls, which can be understandable directly by Database Engine.

1. Network Protocol Driver

* Type-3 Driver converts JDBC Calls into Middleware Server specific Calls. Middleware Server can convert Middleware Server specific Calls into Database specific Calls.

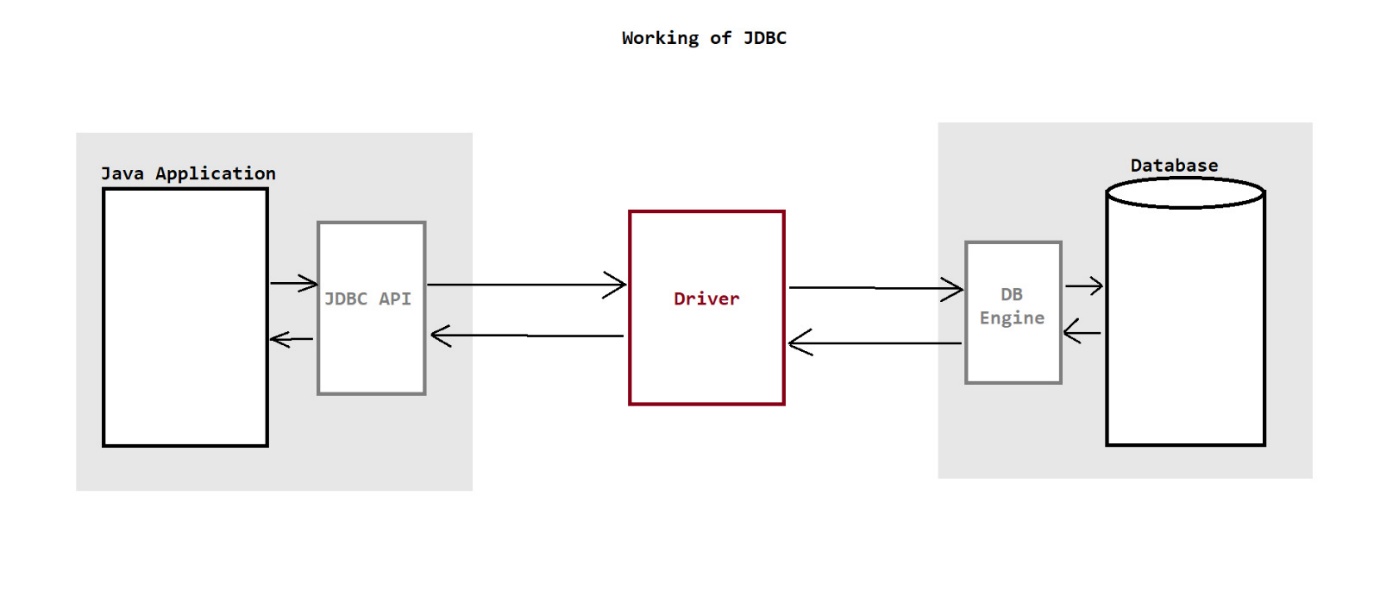
1. Pure Java Driver / Thin Driver.

* This Driver converts JDBC Calls directly into Database specific Calls using Database specific Native Protocols that communicates with the Database.
* This Driver won’t require any Native Libraries at Client side and it is developed only in Java.
* STEPS TO CREATE DATABASE CONNECTION

1. Load and register driver -- Class.forName("Driver ClassName");
2. Establish the connection between java application and database -- DriverManager.getConnection("url", "username", "password");

**CRUD OPERATION**

* JDBC WORKING



* JDBC STEPS

1. Load and register Driver Class

* Class.forName("Driver ClassName");

1. Establish Connection between Java Application and Database

* Connection con = DriverManager.getConnection("url", "username", "password");

1. Create Statement / PreparedStatement / CallableStatement Object

* Statement st = con.createStatement();
* PreparedStatement ps = con.prepareStatement();
* CallableStatement cs = con.prepareCall();

1. Send and execute SQL Query

* int count = ps.executeUpdate();//used for insert, update and delete
* ResultSet rs = ps.executeQuery();//used for select

1. Process Result from ResultSet

* --- Perform The Required Action ---

1. Close Connection

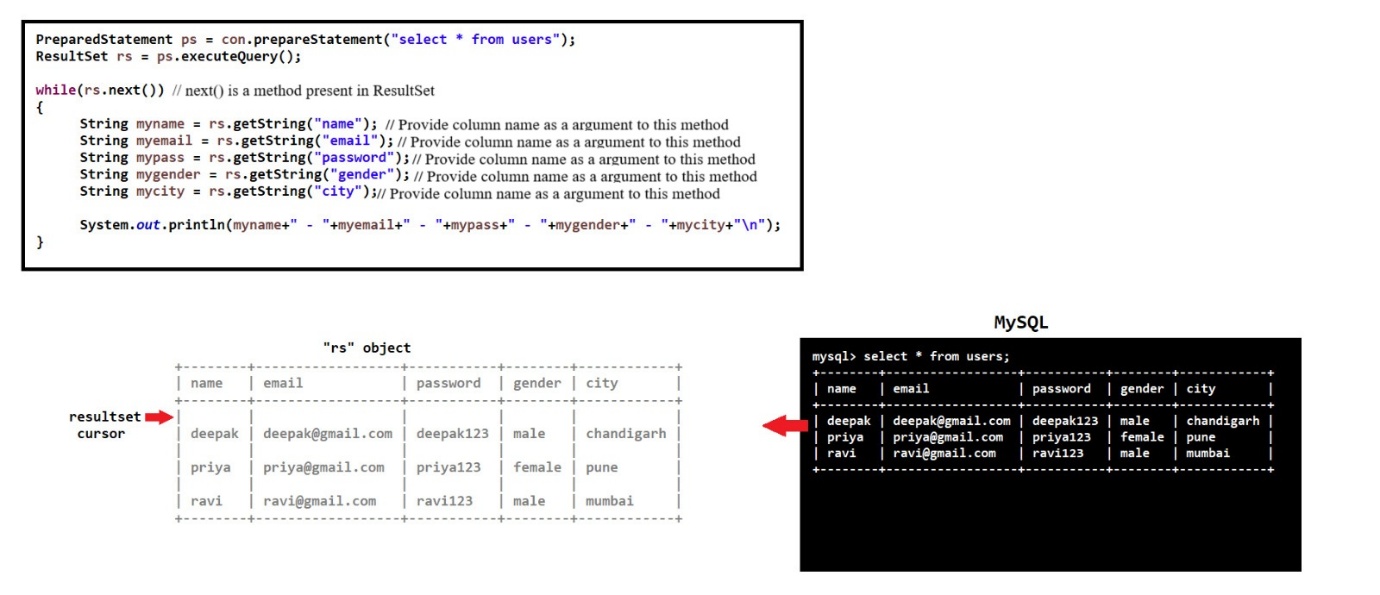
* rs.close
* ps.close();
* con.close();
* CRUD OPERATION

1. C : Create (Insert)

* AUTO\_INCREMENT ek MySQL attribute hai jo kisi column ke value ko automatically increase karta hai jab bhi naye row insert ki jati hai. In this case insert karte waqt aapko us column mein manually value dene ki need nahi hoti. Mostly primary key columns mein use hota hai.

1. R : Retrieve (Select)

* ResultSet :- It is an interface which contains the data and manages the cursor to access the data.
* ResultSet database k uper kiye gaye operation ka result store krke rkhta hai.
* Method of ResultSet :-
* Navigation Methods :- next() , previous() , beforeFirst() , afterLast() , first() , last() , abosolute(-) , relative(-) , etc
* Getter Methods :- getString(-) , getInt(-) , getXXX(-) , etc
* Types of ResultSet cursor :-
* Forward-Only ResultSet :- This type of resultset allows the traversal of data only in forward direction. It's the most memory-efficient type. Syntax :- ResultSet.TYPE\_FORWARD\_ONLY
* Scrollable ResultSet :- This type of resultset allows the traversal of data in forward and backword direction. These are of two types :- ResultSet.TYPE\_SCROLL\_INSENSITIVE and ResultSet.TYPE\_SCROLL\_SENSITIVE
* How to use type :- PreparedStatement ps = con.prepareStatement("select query", ResultSet.TYPE\_FORWARD\_ONLY);



1. U : Update

* TIMESTAMP ek data type hai jo automatic current date+time save kar sakta hai jab koi row insert ya update hoti hai. If DEFAULT CURRENT\_TIMESTAMP attribute is used with these, to MySQL khud hi current date & time fill karega hence don't mention these field in INSERT.

1. D : Delete

* TEXT ek data type hai jo long text ya paragraph jaise data ko store karne ke liye use hota hai like address , comment.
* NOTE POINTS

1. Statement m SQL injection attack hone chances hence use PreparedStatement or CallableStatement.
2. Don't provide hard coded values in statement argument instead use position parameters? and take values from user.
3. For insert , update , delete can use executeupdate method but for select use executequery.
4. Internally RS k ander ek cursor hota hai jo initially first row se pahle positioned rhte hai. ResultSet ek Interface hai jo ess cursor maintain krta hai.
5. When next() is executed the cursor goes to next row ( now first row ) and wah uss row k data ko fetch krne k liye start/ready ho jayega but if next row pura empty raha to wah false return krga. If Reverse order me print Krna hai DB entries (row3 , row2 , row1) ko to use afterlast + previous.

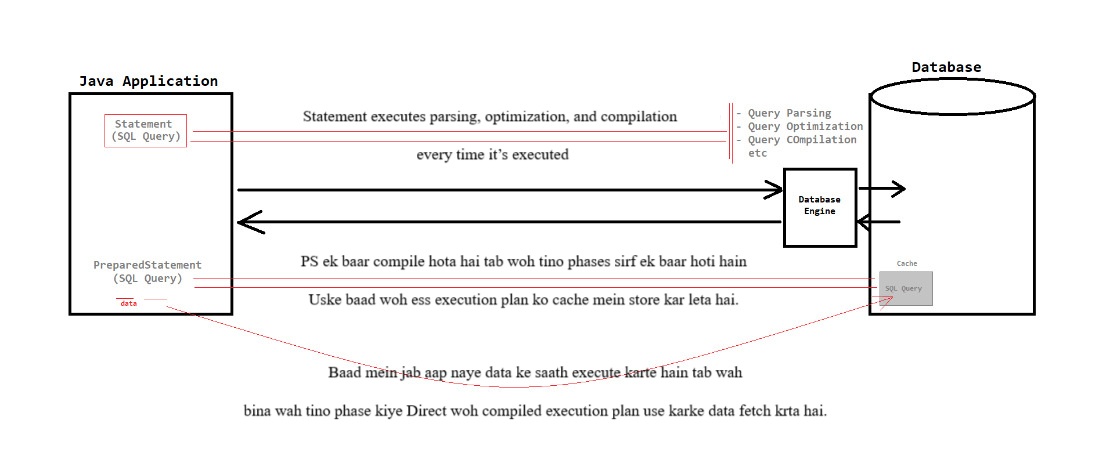
**DEVELOPMENT APPROACH**

* HOW TO USE JDBC IN DEVELOPMENT

1. Use try-catch-finally block
2. Use try-with-resources
3. Use static method approach (Preferred try-resource + utilityresource-class)
4. Above all approches are not efficient. For database connections we use "connection pooling" concept.
5. JDBC is less used for enterprise applications because there are a lot of drawbacks of JDBC and to remove these drawbacks we use frameworks i.e. hibernate, JPA, Data JPA etc

* STATEMENT & PREPAREDSTATEMENT

1. Diagram



1. PreparedStatement

* PreparedStatement objects are used to execute parametrized SQL queries
* PreparedStatement are best suited for dynamic queries which involves user inputs
* PreparedStatement performance is fast as compared to Statement
* PreparedStatement are more secured.
* PS ek baar compile hota hai and goes through all three phases , here it stores this execution plan [ note that ess samay wah positional param k sath compile hua tha ]. Phir jab hum positional param ka data dete h to PS wah execution plan ka use krke query execute kr deta hai and compile nahi krta.

1. Statement

* Statement objects are used to execute simple SQL queries and we can't use any positional parameters
* Statement are best suited for static queries that do not involve any user inputs. Yaha query hum statement me nahi executeXXX() k ander provide krte hai.
* Statement performance is low as compared to PreparedStatement
* Statement are less secured

1. Why use PreparedStatement

* Fast performance
* More secure
* Improves code readability and maintainability
* SQL INJECTION ATTACK

1. SQL Injection is a type of cybersecurity attack that targets the database and is used to manipulate or gain unauthorized access to the data stored within the database
2. The root cause of SQL injection is mixing of "SQL query" and "data"
3. It occurs only in Statement interface
4. Attack Procedure in case of Statement :- In the one field we use a random value with a logical OR expression which should return true for that whole field and fill the rest field with random value. Remember provide string expression such that it should adjust with code string quotation marks.
5. How PreparedStatement protects from SQL Injection Attack ?

* Because "SQL Query" and "input data" are sent separately to the database server.
* User inputs naya sql query nahi bana pati kuki syntax fix hai. user input directly as a field ki value treat hoti h not as a sql command
* Prepared statements ka main maksad SQL query ki structure (ya syntax) ko fix karna hota hai aur user input ko alag se data ke roop mein bind karna hota hai, jisse malicious input ko executable SQL code ke roop mein dekhna mushkil ho jata hai.
* NOTE POINTS

1. Always provide db rsrc in utility/resrc file to eliminate redundancy/repetition.
2. Ager koi resource automatic close krwane hai to use try with resource and place that resources in the try(--Here--)
3. Helper Method -- Helper method waise methods hote hain jo koi common, baar‐baar hone wala kaam ek jagah encapsulate kar dete hain, taaki baaki code zyada clean aur reusable ho.
4. Database Engine SQL queries ko compile krke execute krwata hai.
5. Standard SQL ke hisaab se string literals ko quotes (jaise 'yoy@gmail.com') mein enclose karna padta hai.

**BATCH UPDATION**

* Introduction

It is a batch of updates grouped together and sent to the database in one batch rather then sending them one by one.

* Advantage

1. Application performance will be improved.
2. Network traffic will be reduced.

* Disadvantage

1. It can only be used for "insert, update and delete" SQL query, not for select SQL query
2. If any single SQL query gets an error then it will disturb the flow of program (error wale ko chod ke baki sab value insert kr dega but program me error throw karega\*).

* Procedure

For batch updations we have 3 methods and these methods are present in Statement and PreparedStatement interface.

* addBatch(String query)
* executeBatch() int[]
* clearBatch()
* Note Points

1. Regarding PreparedStatement the setString() is for string data type , setInt() is for integer data type , ...
2. Use pure batch approach when

* Usecase :- Jab 1 million rows load karne ho aur business rule ye allow karta ho ki agar beech ka 500-record chunk fail ho, toh us chunk ko retry/skip karke baaki commit ho jaye.
* Isme auto-commit = true rehne do (ya fir har chunk apni auto-commit pe chale), taaki agar ek row/ek chunk fail ho, toh baaki rows pe effect na pade.
* agar 500-record chunk fail ho, tumhe BatchUpdateException ke getUpdateCounts() se exact kaunsa record/chunk fail hua, pata chalega. Uske baad us chunk ko retry/skip kar sakte ho.

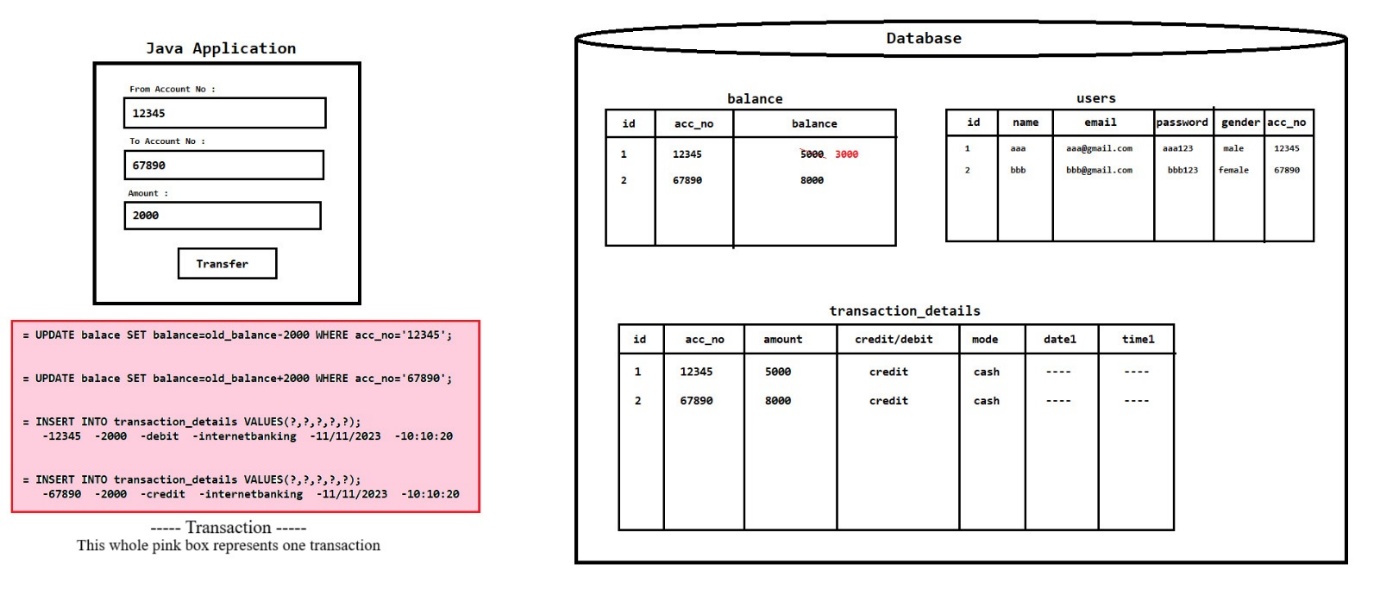
1. Statement.SUCCESS\_NO\_INFO and Statement.EXECUTE\_FAILED – these are predefined constant

* Jab aap executeBatch() call karte ho, to aapko ek int[] array milta hai jismein har ek element batata hai ki corresponding command ka kya effect tha.
* Agar value positive hai, to iska matlab hai ki woh command ne utne rows affect kiye.
* Agar value Statement.SUCCESS\_NO\_INFO hai, to command to execute ho gayi lekin affected rows ka count pata nahi hai. Usually iska value -2 hota hai.
* Aur agar value Statement.EXECUTE\_FAILED hai, to woh particular command fail ho gaya. Iska value typically -3 hota hai.

**TRANSACTION MANAGEMENT**

* INTRODUCTION

1. A group of several SQL operations as a single unit where SQL operations are executed on the principal of either all or none.
2. Diagram



1. Properties of transaction :- It follows the ACID property
2. A (Atomicity) : Either perform all operations or none
3. C (Consistency) : The database state must be stable [Like :- debit & credit operation k baad total amount same rhna chahiye user table ka]
4. I (Isolation) : All the transactions must be executed independently, one transaction must not give effect to another transaction
5. D (Durability) : The changes of successful transaction occurs even if the system/database failure occurs

* TRANSACTION IN MYSQL DB

1. Scenario : Ek transaction k ander multiple sql cmd hote hai and jab wah saare sql cmd exe ho jata h tabhi transaction exe/complete ho jata h. But hum chahte hai ki jab tk transaction complete na ho tab tk jo uske ander ke sql cmd k effect h wah temp rhe not prmnt after comp prmnt ho jaye. Esse Fayda yh hoga suppose aadhe sql cmd exe ho gye but uske baad wale m error aa gaya to hum bakio ko rollback krkr transaction failed show kr sake
2. MySQL k ander chize by default permanently store hoti hai
3. SELECT @@AUTOCOMMIT; : to check the autocommit status. If its 1 then it will store the data permanenetly and if value is 0 then it will store the data temporary
4. SET @@AUTOCOMMIT=0; : It will change the autocommit value to 0 (temporary)
5. COMMIT; : It will store the temporary data permanenetly
6. ROLLBACK; - It will rollback (remove) the temporary data

* TM IN JDBC

1. For transaction management, we have 3 methods :-

setAutoCommit(boolean)

commit()

rollback()

1. These methods are present in Connection interface
2. Syntax

Connection con = ------

try {

con.setAutoCommit(false);

//SQL operations 1

//SQL operations 2

if(condition true) {

con.commit(); }

else {

con.rollback(); } }

catch(Exception e) {

con.rollback();

e.printStackTrace(); }

* NOTE POINTS

1. commit and rollback tabhi kaam karenge jb autocommit false pr set hoga.
2. Use Transaction + Batch :- Jab “All or None” atomicity maintain karni ho (poori batch ek hi unit ki tarah ho). Recommended batch size: ~1000 commands per transaction. Batch size bahut zada mat rkhna warna rollback cost badhega.

**DATASOURCE & CONNECTION POOLING**

* CONNECTION OBJECT

We can get connection object by 2 ways :-

1. DriverManager (class) - "java.sql" package
2. DataSource (interface) - "javax.sql" package

* DRIVER MANAGER

1. DriverManager is a class which is present in "java.sql" package
2. How to get connection object using DriverManager :-

Connection con = DriverManager.getConnection("URL", "USERNAME", "PASSWORD");

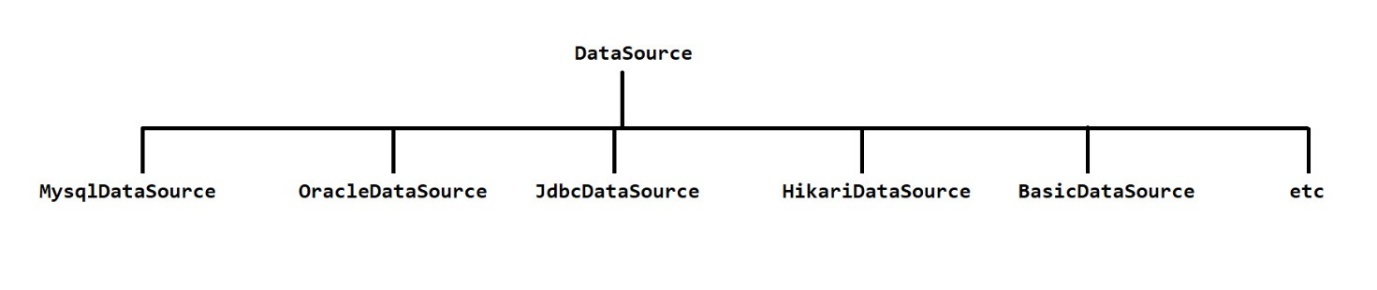
1. DriverManager takes a lot of time to open database connection in a network which will slow down the application performace
2. Whenever there is increase in the number of clients, the performance of the application will be decreased
3. The connection object created using DriverManager is not reusable, thus whenever we need connection object, it will again take time to create

* DATASOURCE

1. It is an interface which is present in "javax.sql" package
2. It defines a standardized way to obtain database connection
3. Some basic implementation of DataSource are :-

* MysqlDataSource : used for MySQL database
* OracleDataSource : used for Oracle database
* JdbcDataSource : used for H2 database (open-source, in-memory, lightweight database written in java)

1. Above provided implemented classes does not have in-built connection pooling feature
2. Diagram

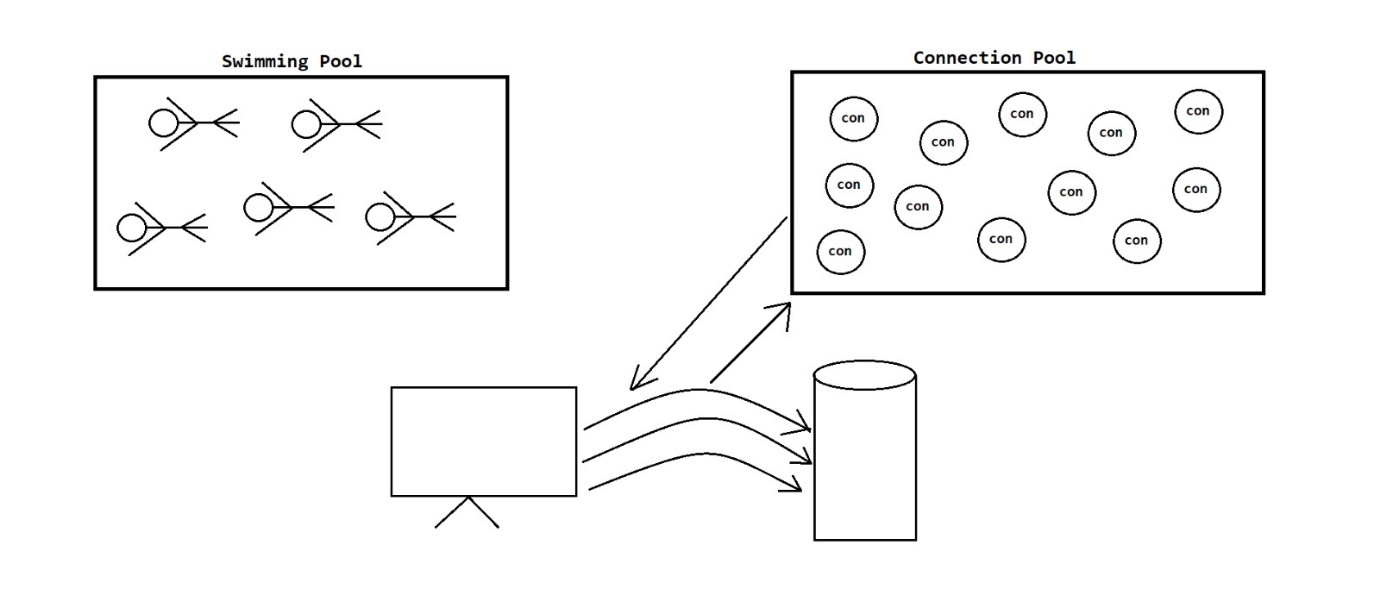


* DATASOURCE OVER DRIVERMANAGER

1. Easy configuration
2. Easy to switch across different databases
3. Automatic driver loading
4. Cleaner and more readable code
5. "Connection Pooling" is the most important advantage of DataSource

* CONNECTION POOL

1. A connection pool is a group of reusable connection objects for a particular database
2. When an application requests for connection object, it obtains one from the pool
3. When an application closes the connection, the connection object returns back to the pool
4. Diagram



* CONNECTION POOLING

1. Connection Pooling is a feature or technique that involves managing and reusing existing database connection objects in connection pool, instead of creating new connections from scratch every time an application interacts with the database
2. This improves the application performance and resource utilization by minimizing the time and resource needed to establish the database connection.
3. Real World Example : Cooking in Restaurants

* PROVIDING CONNECTION POOLING

1. Using third-party connection pooling libraries

* HikariCP
* HikariCP ek super-fast JDBC connection pool library hai.
* Known for its high performance and lightweight nature, making it suitable for modern application.
* "HikariDataSource" is an implemented class for DataSource
* Apache Commons DBCP
* A widely used connection pooling library with configurable options for connection management
* "BasicDataSource" is an implemented class for DataSource
* C3P0
* It offers features like connection testing and customization options
* "ComboPooledDataSource" is an implemented class for DataSource
* BoneCP
* It is also lightweight connection pooling library designed for speed and efficiency
* "BoneCPDataSource" is an implemented class for DataSource

1. Application server-provided connection pooling

* Many application servers i.e. Apache Tomcat, WildFly etc come with built-in connection pooling capabilities.
* JMX AND MBEAN

1. JMX (Java Management Extensions) ek standard hai, jisse Java apps runtime mein metrics aur operations expose kar paate hain.
2. MBean (Managed Bean)—yeh ek Java object hota hai jisme kuch attributes (properties) aur operations (methods) define hoti hain. Iske ander managed bean hote hai.
3. Types of MBeans: Standard MBeans – fixed interface + implementation. MXBeans – simplified type mapping for monitoring. Dynamic MBeans – runtime par attribute/operation change kar sakte hain
4. JVM ke andar ek MBeanServer hota hai (via ManagementFactory.getPlatformMBeanServer()), jahan saare MBeans register hote hain.
5. HikariCP khud provide karta hai ek HikariPoolMXBean. Jab tum auto ya manual registration code use karte ho, to ye MXBean JVM ki MBeanServer mein register ho jata hai.
6. Fir JConsole, VisualVM, ya Prometheus JMX exporter jaise tools se tum real-time pool metrics dekh sakte ho—connection leaks pakadne, pool sizing tune karne, ya spikes monitor karne ke liye perfect.
7. ActiveConnections , IdleConnections , ThreadsAwaitingConnection , TotalConnections Ye log HikariCP ke MXBean ke attributes hain , jo JMX MBeanServer me register hone par JMX attributes ban jate hain aur tum inhe code se ya JMX tools se access kar sakte ho!

* NOTE POINTS

1. JDBC does not provide in-built connection pooling feature but we can integrate third-party connection pooling libraries with JDBC.
2. Spring framework ke ander bhi inbuild connection pooling ka feature nahi hota hum third party library ya phir application server ka hi use krke connection pooling implement krte hai.
3. driver manager < data source (provides better performance than DM) < data source with connection pooling (provides better performance than DM + Conn Pool feature).

**SPRING JDBC**

**INTRODUCTION**

* Spring JDBC Module

1. Spring JDBC provides mechanism to connect the spring application with database and execute SQL queries
2. It internally uses JDBC API (but solves the problem of Plain-JDBC)
3. We have to use spring-jdbc dependency for spring JDBC module
4. Spring provides some classes which are as follows :-

* DriverManagerDataSource
* JdbcTemplate
* NamedParameterJdbcTemplate
* DriverManagerDataSource

1. It is an implemented class of DataSource interface which is present in "org.springframework.jdbc.datasource" package
2. It is used for :- Database configurations , Driver loading , Connection creation , etc.

* Plain-JDBC and Spring-JDBC

|  |  |
| --- | --- |
| Plain-JDBC | Spring-JDBC |
| Connection object must be created manually. | JdbcTemplate automatically provides the connection object. |
| Requires these steps manually:  i. Load driver (Class.forName)  ii. Create Connection  iii. Create Statement  iv. Execute query  v. Close resources (boiler-plate code). | Boiler-plate code (steps i, ii, iii, v) is handled automatically by JdbcTemplate; only the query step is variable. |
| Must handle checked exceptions with try-catch or throws. | Converts checked exceptions into runtime exceptions — less boilerplate. |
| SELECT query results are stored in ResultSet. | Results can be stored in collections like List, Map, or Bean objects. |
| ResultSet is non-synchronized and cannot be transferred over the network. | Collections and Bean objects are synchronized and can be transferred over the network. |
| Manual code needed to map ResultSet to Beans or Collections. | Uses RowMapper and pre-defined methods for easy mapping. |
| Provides limited support for transaction management. | Provides strong support for transaction management. |

* Note Points

1. JdbcTemplate ka use karte time manually Connection object ka use karne ki zarurat nahi padti. JdbcTemplate khud hi internally DataSource se Connection ko manage karta hai, handle karta hai aur close bhi karta hai.
2. Test purpose ya debugging ke liye connection ki availability check krne k liye (i.e., connection successful ho ya nahi) hum jTemp.getDataSource().getConnection() use kr skte hai but production ke time iska use nahi hai.

**JDBCTEMPLATE**

* Introduction

1. It is the central class in Spring-JDBC support classes
2. It is used for

* Simplifies JDBC :- JdbcTemplate reduces boilerplate JDBC code making it readable and concise
* Connection Management :- It manages connections, reducing the need for manual connection handling
* Exception Handling :- Automatically translates the database exceptions into Spring's DataAccessExceptions which simplifies the error handling
* SQL execution :- Executes SQL queries i.e. insert, update, delete, select etc
* Parametrization :- Support both positional and named parameters in SQL queries
* Methods

1. update(-) --- used for insert, update and delete SQL queries
2. query(-) , queryForList(-) , queryForMap(-) , queryForObject(-) , queryForRowSet(-) --- used for select SQL queries
3. execute(-) --- used for DDL (create, drop, alter etc) SQL queries

* RowMapper

1. RowMapper is an interface which is used for mapping rows of a relational database result set to java object.
2. RowMapper is a callback interface which means it provides a callback method that is invoked for each row in the result set during data retrieval.
3. It converts result set into object
4. Method :-

* T mapRow (ResultSet rs, int rowNum) throws SQLException
* queryForObject() – for single row
* query() – for multiple rows

**NAMEDPARAMETERJDBCTEMPLATE**

* Introduction

1. It is the class which is provided by Spring framework as the part of Spring-JDBC module
2. It provides "higher-level abstraction" and more convenient way to work with SQL queries
3. A "higher-level abstraction" means that it provides user-friendly way to interact with the system as compared to lower-level abstraction
4. In JdbcTemplate, we provide "Positional Parameters" which is not user-friendly because if there are a lot of positional parameters then it will create confusion. To remove this confusion we use NamedParameterJdbcTemplate in which we use "Named Parameters"

* Ways to achieve

We can provide "Named Parameters" by 2 ways :-

1. By using Map <interface>

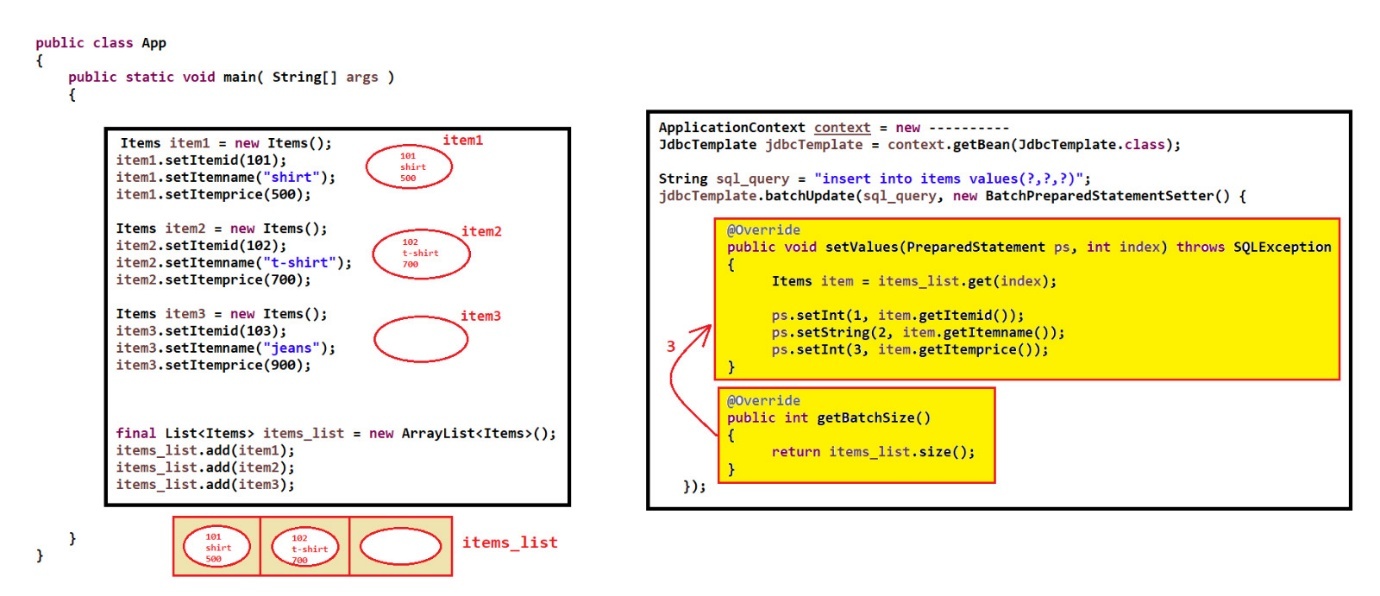
* We can use any implemented class of Map interface for eg. HashMap etc

1. By using SqlParameterSource <interface>

* We can use these implemented classes :- MapSqlParameterSource or BeanPropertySqlParameterSource

**BATCH UPDATE**

* Introduction
* Batch update me hum ek sath multiple query ko wrap krte hai then unhe ek sath hi execute kr dete hai
* Batch update is the process by which we group multiple sql queries together and execute them at the same time as a single unit rather than one by one.
* We can use batch update only for insert , update & delete only and not for select
* It is a batch of updates grouped together and sent to the database in one batch rather then sending them one by one.
* Method
* batchUpdate(-) ;
* Diagram



* Note Points

1. jdbcTemplate.batchUpdate ( Sting sqlCmd, BatchPreparedStatementSetter object )

* Provide parametirized sql commands and then to provide the required object.
* First import the BatchPreparedStatementSetter (yah ek interface hai & ess class k meth callback nature wale hai ) then create an anonymous to implement the interface and create the object
* invokes your getBatchSize() once to Determine how many statements to execute. [Jitni size return ki hogi from getBatchSize() utni baar setValue() execute hoga]
* Then it enters a loop from i = 0 to i < batchSize and i++ where it calls batchSetter.setValues(preparedStatement, i); + adds to batch iteratively. Then finally executes the batch.
* Since setValue ek call back method hai hence wah apne aap execute hoga & index ki value apne aap increment karega (i=0; i<batchLength; i++)

**CONNECTION POOLING**

* Connection Pool

1. A connection pool is a group of reusable connection objects for a particular database
2. When an application requests for connection object, it obtains one from the pool
3. When an application closes the connection, the connection object returns back to the pool.

* Connection Pooling

1. Connection Pooling is a feature or technique that involves managing and reusing existing database connection objects in connection pool, instead of creating new connections from scracth every time an application interacts with the database
2. This improves the application performance and resource utilization by minimizing the time and resource needed to establish the database connection
3. Real World Example : Cooking in Restaurants

* Ways To Achieve
* HikariCP [third-party connection pooling library]

1. Using third-party connection pooling libraries like HikariCP
2. Known for its high performance and lightweight nature, making it suitable for modern application
3. "HikariDataSource" is an implemented class for DataSource
4. sirf hikari cp wala dependency add kr dena baki required wali chize(log4j) maven khud kr lega
5. Flow

* jtemp ka update meth hikari datasource ko req send krta hai ki usko conn obj chahiye taki wah given query ko exe kr sake hence hikari datasource usko conn object provide kr dega.
* Then update meth execute ho jane k baad wah conn obj ko hikari datasource ko return krwa dega and hikari pool k ander conn ko return kr dega (a.k.a conn closed).
* Than during exe of another obj jtemp.update() meth it will again send request to hikari datasource where it will req a conn from pool which it will receive and again the same thing will happen.
* Application server-provided connection pooling
* Many application servers i.e. Apache Tomcat, WildFly etc come with built-in connection pooling capabilities.
* Note Points

1. Spring does not provide in-built connection pooling feature but we can integrate third-party connection pooling libraries with Spring.
2. Hikari datasource create krte time driver class load krne ki need nahi hoti kya ? -- Nahi hikari internally khud se hi sambhal leta hai.
3. Limit s jada conn crt krke exe kiya to thoda time wait krke exception raise kr dega -- hence either inc size or close the some other.
4. context.getBean ( Bean-Name , Class-Name ) ; Second argument me jis class ka naam doge spring given named bean ko uss class type m cast krke reference me store karega
5. hikariConfig.setRegisterMbeans(true); Ager nahi kiya to bhi chalega error nahi aayega.

**DESIGN PATTERN**

* Design Pattern Book
* A book i.e. "Design Patterns : Elements of Reusable Object-Oriented Software" was written in 1994 and its best selling book for design patterns
* This book was written by 4 authors i.e. Erich Gamma, Ralph Johnson, John Vlissides and Richard Heml and they all 4 are known as Gang Of Four (GOF)
* They categorized all the design patterns into 3 parts :-

1. Creational Design Patterns
2. Structural Design Patterns
3. Behavioral Design Patterns

* Creational Design Patterns
* These patterns focus on the process of object creation.
* They help manage the instantiation of objects, hiding the details of how objects are created and composed.
* There are 5 Creational Design Patterns :-

1. Singleton Design Pattern :- Ensures a class has only one instance.
2. Factory Method Design Pattern :- Defines an interface for creating objects.
3. Abstract Factory Design Pattern :- Provides an interface for creating related object families.
4. Prototype Design Pattern :- Creates new objects by copying an existing one.
5. Builder Design Pattern :- Separates the construction of a complex object from its representation

* Structural Design Patterns
* These patterns deal with the composition of classes and objects to form larger structures.
* They facilitate the creation of relationships between objects to build more complex systems.
* There are 7 types of Structural Design Patterns :-

1. Adapter Design Pattern :- Allows incompatible interfaces to work together.
2. Bridge Design Pattern :- Separates an object's abstraction from its implementation.
3. Composite Design Pattern :- Composes objects into tree structures to treat individual objects and compositions uniformly.
4. Decorator Design Pattern :- Dynamically adds behavior to objects without altering their class.
5. Facade Design Pattern :- Provides a simplified interface to a complex subsystem.
6. Flyweight Design Pattern :- Shares common, stateless objects to conserve memory.
7. Proxy Design Pattern :- Provides a surrogate or placeholder for another object to control access.

* Behavioral Design Patterns
* These patterns define how objects interact and communicate with each other.
* They address the responsibilities and behaviors of objects within a system.
* There are 11 types of Behavioral Design Patterns :-

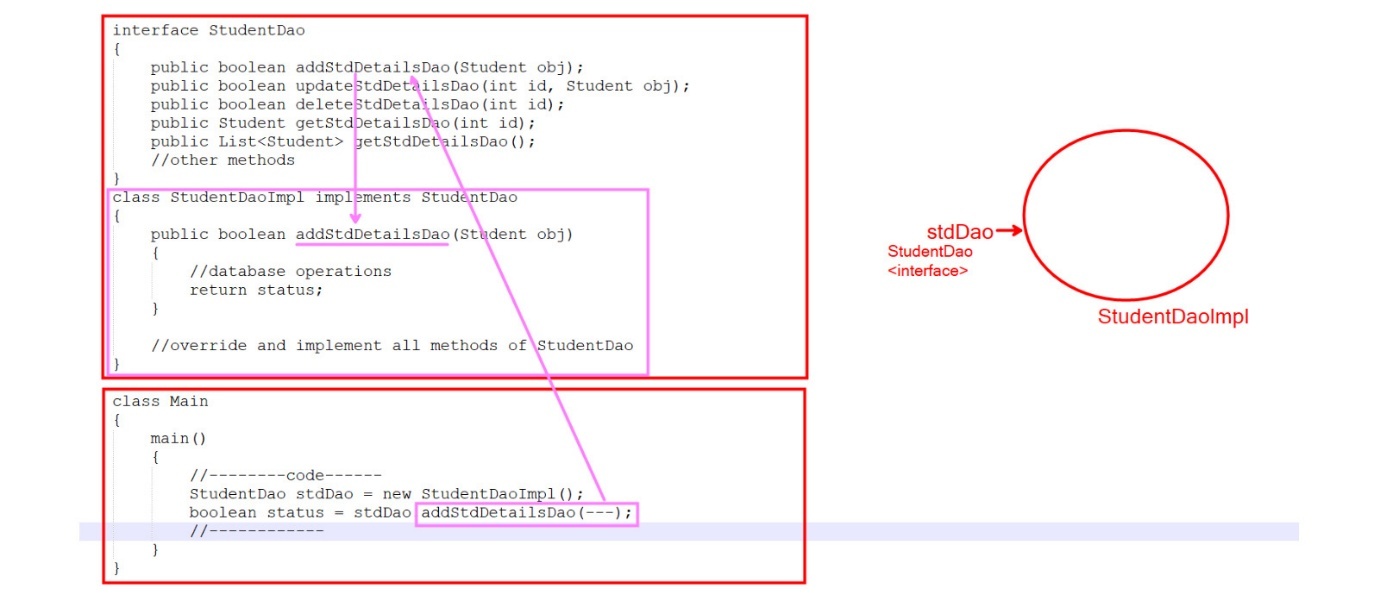
1. Chain of Responsibility Design Pattern :- Passes requests along a chain of handlers.
2. Command Design Pattern :- Encapsulates requests as objects to parameterize clients.
3. Interpreter Design Pattern :- Defines a grammar for interpreting language expressions.
4. Iterator Design Pattern :- Provides a way to access elements sequentially.
5. Mediator Design Pattern :- Defines central communication between objects.
6. Memento Design Pattern :- Captures and externalizes an object's internal state.
7. Observer Design Pattern :- Notifies observers of an object's state changes.
8. State Design Pattern :- Alters an object's behavior when its state changes.
9. Strategy Design Pattern :- Defines a family of algorithms and makes them interchangeable.
10. Template Method Design Pattern :- Defines the skeleton of an algorithm with certain steps.
11. Visitor Design Pattern and:- Separates an object structure from operations on it.

* Miscellaneous Design Patterns
* These are not defined in the above mentioned book
* Some examples are :-

1. Dependency Injection Design Pattern
2. DAO Design Pattern
3. MVC Design Pattern

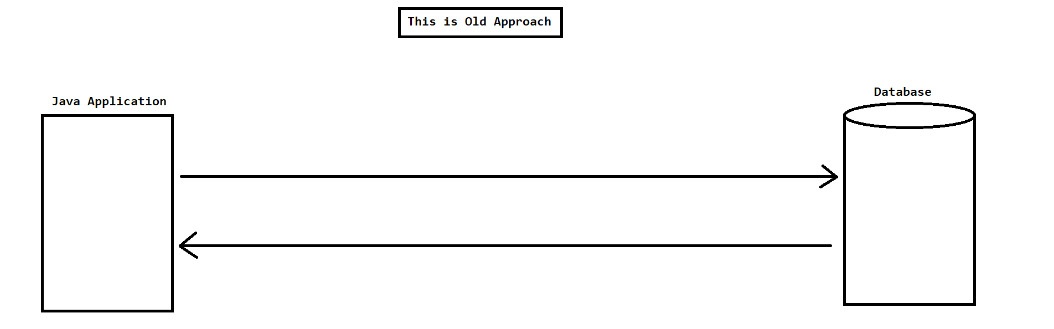
**JDBC DAO**

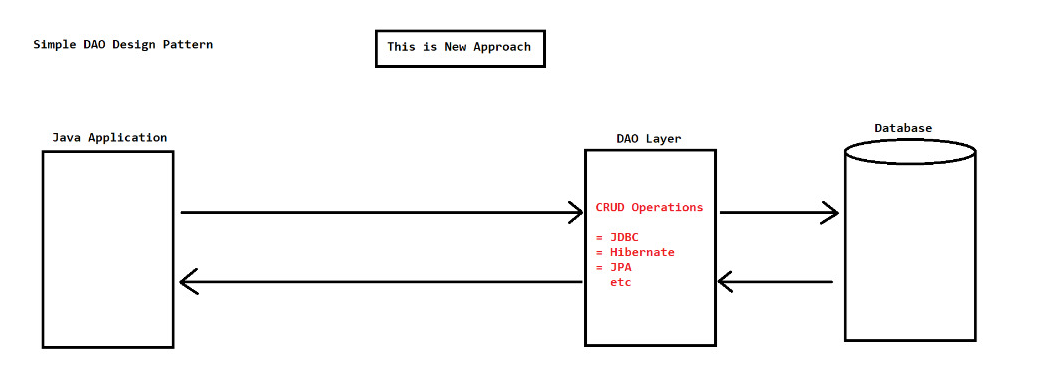
* Introduction
* Design patterns are established best practices and reusable templates for solving common software design problem
* "established best practices" means that these practices have been tried, tested and widely recognized as effective solutions
* They offer structured approaches to create relationships and interactions between classes and objects, promoting efficient, maintainable and scalable code
* For e.g. Singleton Design Pattern , Factory Design Pattern , Abstract Factory Design Pattern , Prototype Design Pattern , DAO Design Pattern , MVC Design Pattern , Dependency Injection Design Pattern
* Modular design pattern means combination of multiple design pattern.
* DAO Design Pattern
* DAO stands for "Data Access Object"
* Diagram

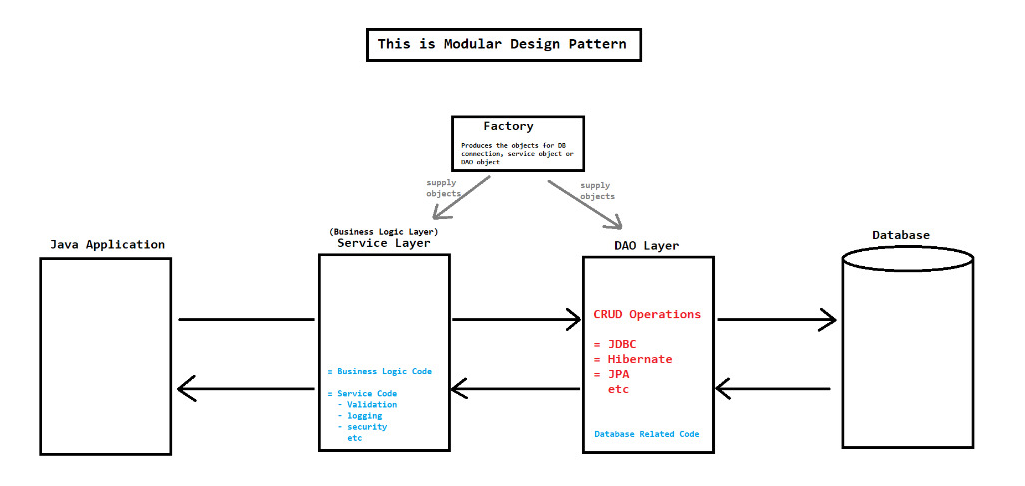


* It is used to separate the data persistence logic (database related code/logic) into a separate layer. This way, any other layer i.e. service layer etc does not know how low-level operations are performed to access the database
* Procedure

1. Create different DAO package
2. Create an interface which contains abstract methods
3. Create an implemented class and override / implement the methods
4. Object interface ka reference leke hi banana
5. Diagram







**SPRING JDBC DAO**

* Spring DAO
* Spring DAO is a concept (design pattern) to work with database technologies i.e. JDBC, Hibernate, JPA etc in an easy way
* We can achieve DAO design pattern in spring by 3 ways

1. By straight-forward and traditional way
2. By extending DAO support classes
3. By using Annotations

* Their preference order is 3>1>2
* By straight-forward and traditional way
* This approach involves the creation of custom classes to perform data access operations in a traditional way without relying on spring pre-defined classes or annotations.
* By extending DAO support classes
* Spring has provided some DAO support classes (It's name is generally in this format -- XxxDaoSupport) to simplify certain aspects of DAO development
* Spring has provided many DAO support classes and 2 main classes are :-
* JdbcDaoSupport

1. yh abstract class hai hence iska object create nahi kr skte. Following are some important method of this class
2. public final void setDataSource(DataSource dataSource){ - } // set this in config file in the DaoImp bean as it is internally rqrd by JdbcDaoSupport to crt jdbctemplate obj
3. public final void setJdbcTemplate(JdbcTemplate jdbcTemplate) { - }
4. public final JdbcTemplate getJdbcTemplate() { - } // this will provide us object of JdbcTemplate hence no need to create & set the our own JdbcTemplate
5. no need to create an object of jtemplate as it is been already provided by the JdbcDaoSupport just call the appropriate method to JdbcTemplate & set the Datasource.

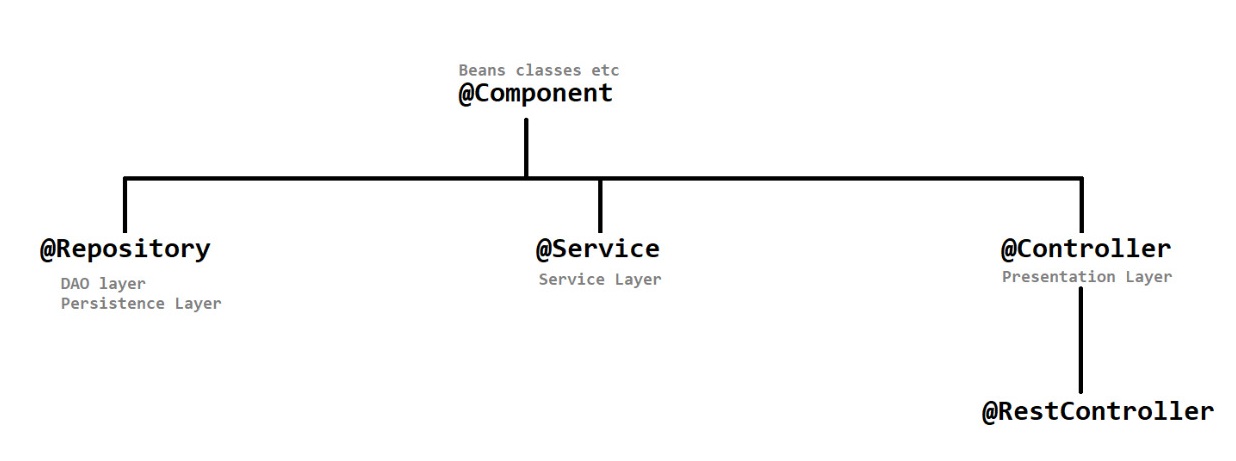
* NamedParameterJdbcDaoSupport

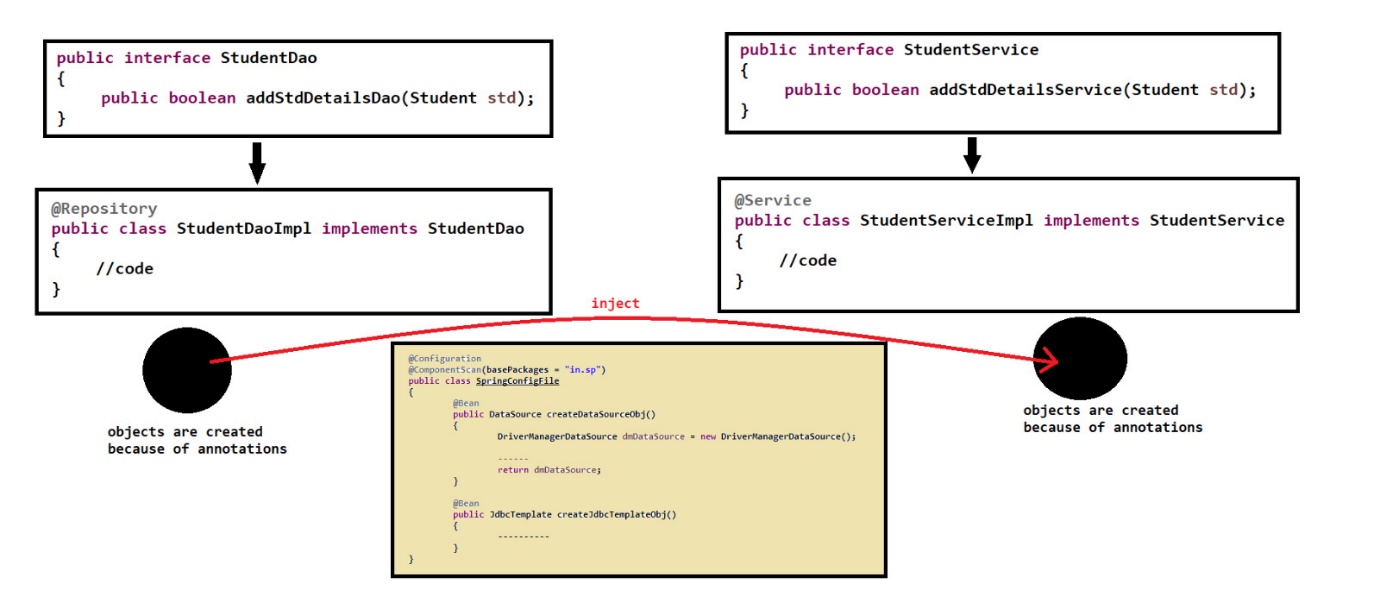
1. public NamedParameterJdbcTemplate getNamedParameterJdbcTemplate() { - }
2. just involve usage of named param and map while the rest of the things are same as JdbcDaoSupport

* This is not a standalone approach because we still need to extend and customize user defined classes to create our specific DAO's
* By using Annotations
* Spring provides some annotations which can simplify the configuration and make our DAO's more concise and declarative
* Main annotations are :-

1. @Repository :- It is used with DAO layer / persistent layer / data access layer. It is a specialization of @Component
2. @Autowired :- Used for automatic dependency injection.
3. @Transactional :- It is used for transaction management
4. @Service :- It is used with service layer. It is a specialization of @Component

* Diagram





* Note Points

1. Spring me object creation automatic hota hai hence ager kisi program me object ki need hai then just declare its reference variable in the same program. Then in the config file create a bean for this program anf use setter meth DI or Constructor DI to get the object from an existing bean.
2. Jdbc-Dao-Service me humne service-implementation class k ander instance reference variable nahi banaya tha because hum Jdbc Dao ka object manually create kr rhe the. But Spring wale approach me hum manually object create nahi krte hence instance reference variable banaya and DI use kiya configuration file me.
3. Before using any external userdefined object in a class we have to make sure that the required object data is available to use if not we can get NullPointerException. We can sure this by 2 ways.

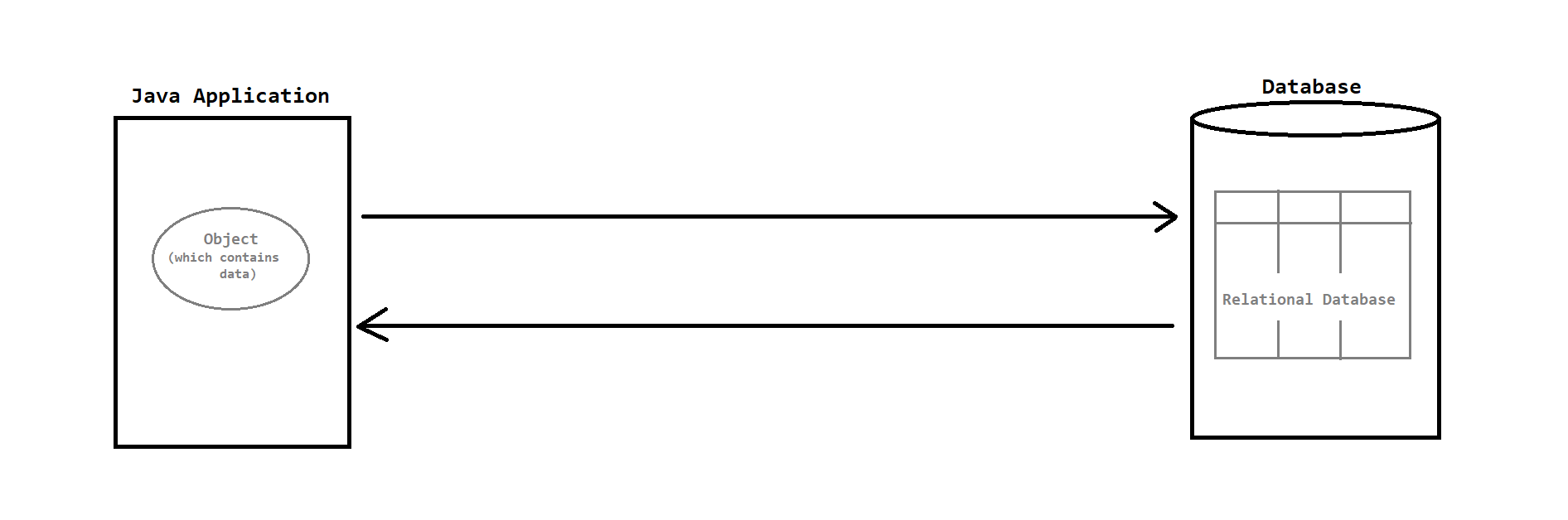
* First declare the required object reference as a parameter in method hence it will act as local variable & it will be provided by the caller.
* Second declare the object as a instance reference variable and then use DI in config file to inject the object data from existing bean.

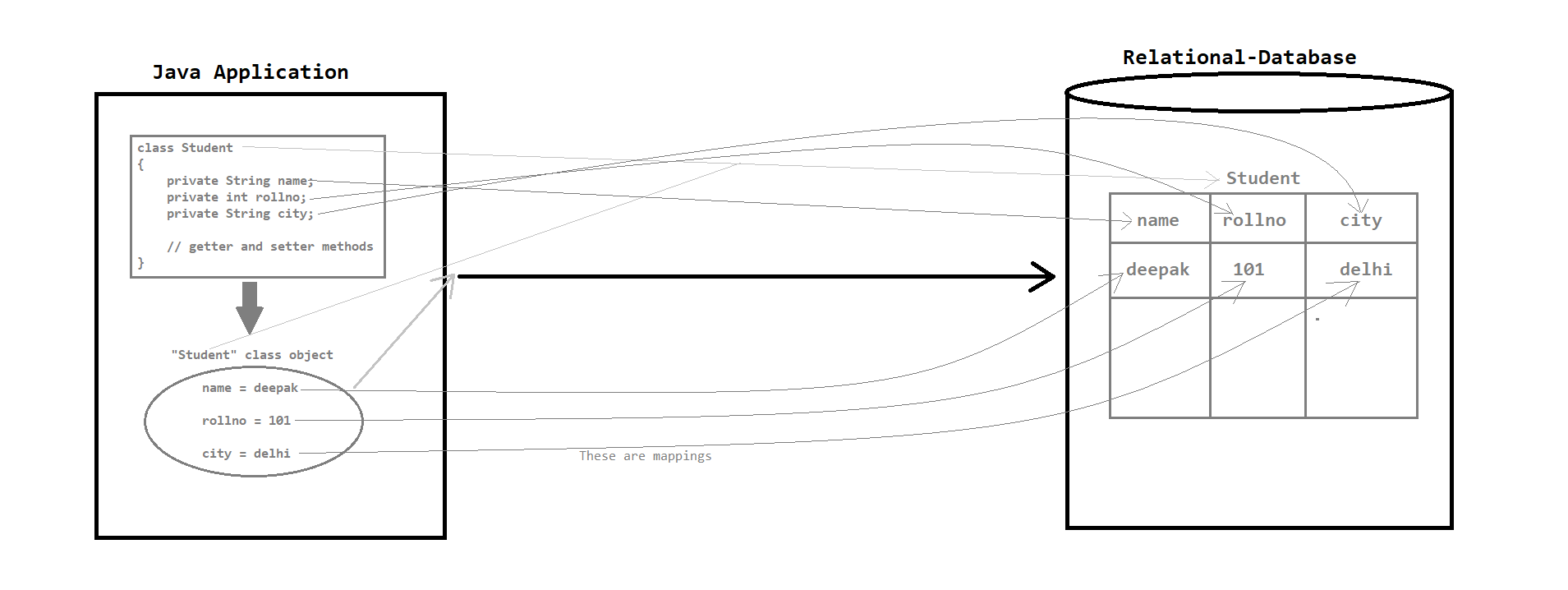
1. Parent p1 = new Child () ; Yh preferred hai. Therefore config file ka bean object ka "return-type" parent-type hi rakho but obj child ka hi banao (either using parent or child reference)

**HIBERNATE**

**BASIC TERMS**

* ORM (Object-Relational Mapping)
* Object ke data(property/variables) ko database ke data (columns & rows ) ke sath map kr deta hai.
* Class Name = Table Name ; Property Name = Column Name ; Property Value / Object Data Rows ; Number of columns = Number of Property.
* ORM is a programming technique (style/paradigm ) that allows data to be mapped between "Object" in object-oriented programming code and "Relational Database" using XML file or annotations. We can compare it with OOP, AOP etc
* It will map the "Object" and "Relational Database" in such a way that :- model class(object) corresponds to the table , properties map to the table column and property / object values become row in database table
* ORM simplifies database interactions in object-oriented programming languages, making it easier for developer to work with database
* Diagram





* Some java related ORM tools (frameworks) are :-

1. Hibernate
2. JPA (Java Persistence API) -- ORM Tool + Specification
3. TopLink
4. EclipseLink
5. MyBatis

* Data Persistency
* Data Persistency means hum data ko kahi bhi permanently store krwa sake.
* Data means information and persistency means permanent
* Data Persistency is the ability to store the data permanently in a way that it is not lost when the program or system is no longer running.
* Example :- int a = 10 ; Yh persistent data nahi hai as program khtm to yah data bhi khtm.
* To achieve data-persistency wrt java we have following techniques or technologies :-

1. Serialization and Deserialization
2. JDBC
3. ORM :- Hibernate , JPA , TopLink , EclipseLink , MyBatis

* Note Points

1. ORM is not developed by a specific individual or organization. It is a collaborative effort programming technique on which many organizations are working together
2. ORM is typically used with relational databases (MySQL, Oracle, SQL Server, PostgreSQL etc). We normally does not use ORM with NoSQL databases (MongoDB, Cassandra, Redis etc)

**HIBERNATE WORKING**

* Introduction

1. Hibernate is an open-source ORM tool or framework which can be used with Java
2. NOTE : Hibernate is not a core or official component of Java EE (Java Enterprise Edition) platform
3. Hibernate was developed by Gavin King while working in JBoss. The project was later acquired by Red Hat company.
4. Hibernate first version was released in 2001
5. It simplifies database programming in java applications by mapping java objects to database tables, making it easier to work with relational database.

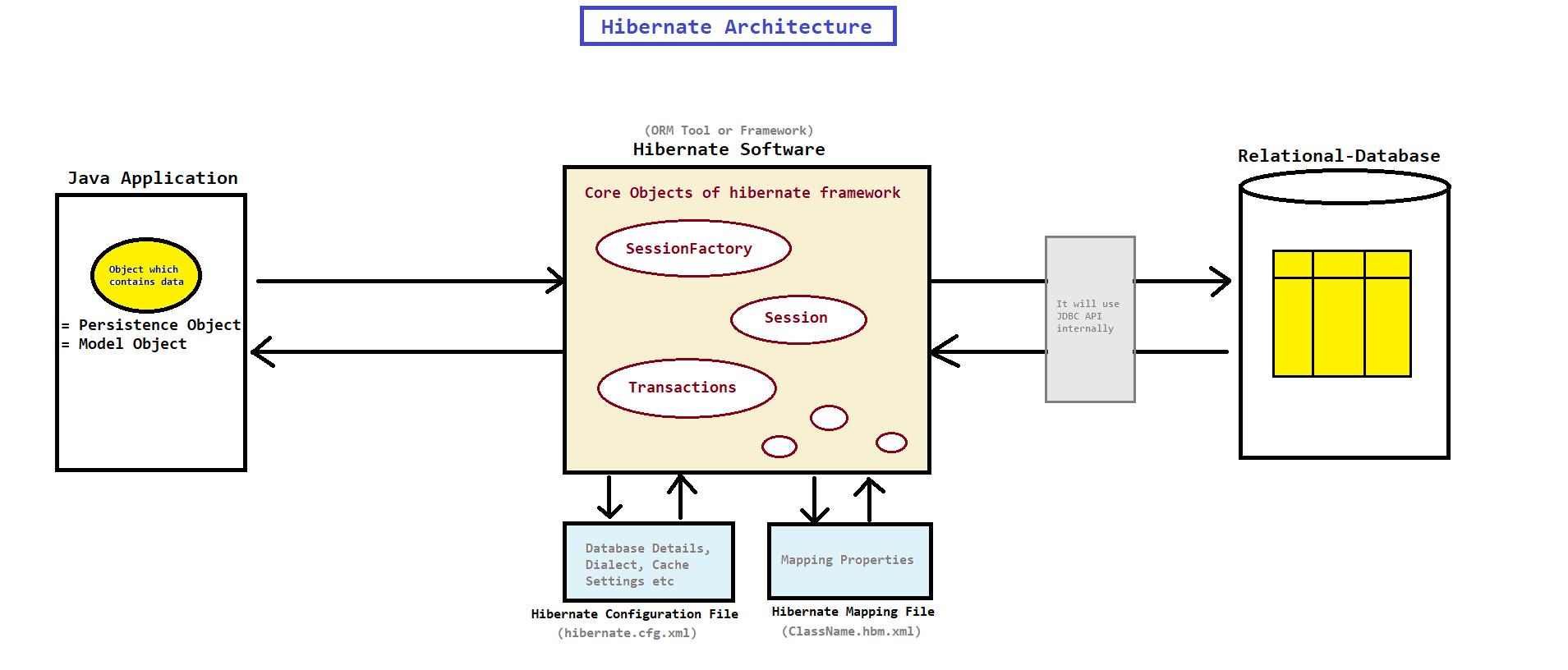
* Features

1. Database independent :- It supports various relational databases
2. Automatic Table Generation :- It can generate database schema from java classes
3. Query Language :- Hibernate provides its own query language i.e. HQL (Hibernate Query Language). It is database-independent which allows us to write queries using a syntax that hibernate translates into database-specific SQL queries
4. Caching :- It provides very good cache mechanism for better performance
5. Transaction Management :- It supports ACID transaction properties
6. Connection Pooling :- It manages and optimizes database connections
7. Integration :- It can be easily integrated with Java EE applications and Java frameworks (Spring etc)
8. Community Support :- It provides active open-source community for updates and support

* Architecture
* Persistent Object = Model Object = Java Object
* Generalized Name :- Hibernate.cfg.xml -- hibernate config file ; hibernate mapping name -- classname.hbm.xml ;
* Working

1. Object/java-app will interact with hibernate software
2. Then yaha 2 files hoti hai (config - db detail + dialect + cache setting + .. , mapping file - mapping prop tells object ko kis tarike se db k sath map krwana h) jinke sath hibernate software interact krta hai regularly
3. hibernate/hum hibernate k core object + jo obj humne diye + above file ka use krke db k sath interact krta hai.
4. Config detail ka use krke connection establish karega + ...
5. Given mapping detail ka use krkr hibernate sql query generate karega and persistent operation perform krga.
6. hibernate directly db s interact nahi krta wah via/internally JDBC ke through krta hai kaam

* Diagram



* Components

1. Application Layer

* In this layer persistence object is created
* In this layer we provide business logic and persistence operations
* In this layer hibernate software will be activated

1. Persistence Object

* It is java object that represents the data and is designed to be persistent (means it can stored or retrieve or modify the data from relational database)

1. Hibernate Configuration File

* It is typically an XML or Java file
* It contains settings and properties for hibernate i.e. database connection information, dialect, cache settings etc

1. Hibernate Mapping File

* A hibernate mapping file (usually in XML format) defines how java object are to be mapped with database tables

1. Hibernate Software

* When the client activates hibernate, the framework retrieves configuration details from hibernate configuration file and establish the connection with database
* When the client initiates persistence operation, hibernate generates the appropriate database-specific SQL query and executes it.

1. Relational Database

* The persistence object will be mapped with relation-database and new table with data will be created

**PROGRAM PROCEDURE & XML MAPPING**

* Steps to create hibernate program

1. Download & Install any IDE (Eclipse)
2. Create "Simple Java Project" or "Maven Project"
3. Add "hibernate jars" (for java project) or "provide dependencies -- hibernate core" (for maven project)
4. Create "POJO class"
5. Create "Hibernate Mapping File"

* camelCase file name is preferred
* provide xml schema which is present in hibernate-core>org.hibernate>hibernate-mapping(dtd) copy the schema declaration
* hibernate ke ander hume ek primary key banana zaruri hai (xml file me kisi property ko primary key banana h to use id tag) else error dega.

1. Create "Hibernate Configuration File"

* camelCase name file named preferred
* provide xml schema which is present in hibernate-core>org.hibernate>hibernate-comfig(dtd) copy the schema declaration.
* create session factory tag and provide db config detail; mysql-connector j wali dependency ki need hogi
* set ddl query using property tag with update as a value and auto as attribute to automatically create table
* link the mapping file using mapping tag
* property tag with name driver\_class, url, username.password inke pahle connection. laga dena

1. Create main class to execute hibernate application

* Procedure for main class

1. Create "Configuration" object

* The Configuration class is used to configure and manage hibernate settings
* How to get Configuration object :- Configuration cfg = new Configuration();
* Methods of Configuration class :-
  1. configure()// It is used to load the hibernate configuration details from an external file (hibernate-config-file)
  2. configure(String resource) // use this jab config file kisi dusere package ke ander but ager same package ke ander hai to use the above

1. Create "SessionFactory" object

* SessionFactory is an interface that represents a factory for creating "Session" objects
* It-is-used-to / internally-handle tasks like load the configuration details from configuration file, setting up connection pool, manage database connections etc
* How to get SessionFactory object :- SessionFactory sessionFactory = cfg.buildSessionFactory();

1. Create "Session" object

* Session is an interface that represents a single unit of work
* It is used to perform database operations
* How to get Session object :- Session session = sessionFactory.openSession();
* Database operations that we can perform using Session object are :-
  1. Insert operation :- save() and persist()
  2. Update operation :- update() and saveOrUpdate()
  3. Delete operation :- delete()
  4. Retrieve operation :- get() and load()

1. Create "Transaction" object

* Transaction is an interface that represents database transaction
* It is used to control and manage transactions
* hibernate saare db operation ko transaction nature wala hi consider krta hai
* How to get Transaction object :-
* First way :- Transaction transaction = session.getTransaction(); then use transaction.begin();
* Second Way :- Use Transaction transaction = session.beginTransaction(); // transaction ko get bhi krwaye and begin bhi kr dega hence upper k dono meth wale kaam yh khud hi kr lega
* transaction.commit(); kr dena last me to commit the mentioned transaction.
* Note Points

1. In JDBC, AutoCommit default value is "true" and we have to set it to "false" by "con.setAutoCommit(false);" statement. But in hibernate AutoCommit default value is "false"
2. session.save() ; table and uska data database me daal dega matble table create krke uske ander data insert krdega automatically
3. dialect ke hisab se hi sql queries generate krta hai hibernate
4. configuration file me use show\_sql property if you want to see sql cmds executed by hibernate + use format\_sql for better display
5. ager dialect mentioned nahi hai config me then hibernate given driver class k hisab s appropriate sql-technology use karega (but ager hibernate ko nahi mila to tujhe provide krna padega) if you want to use other suitable dbms-tech then use property with name dialect and mentioned it's dialect as per version which can be found in the jar file of hibernate core.
6. Dialect yah dekhta hai ki aap konsa database use kr rahe ho and uske hisab se hi sql query generate karega. To see various dialect go to hibernate jar > org.hibernate.dialect > select the appropriate dialect aper version and software.

**ANNOTATION MAPPING**

* Annotations used in Hibernate

1. @Entity

* hibernate ko bolega ki ess class ko mapping k liye consider karo
* The @Entity annotation is used to mark the java class as an entity, indicating that the instances of this class will be mapped to rows in a database table
* When we annotate a class with @Entity, hibernate recognizes it as a persistent entity and we can perform database operations (such as insert, update, delete etc) on instances of this class
* This annotation is used at the class level
* Syntax :-

@Entity

class ClassName { ----- }

1. @Table

* @Table annotation is used to map the class with database table
* This annotation is used at the class level
* Syntax :-

@Entity

@Table(name = "table\_name")

class ClassName { ----- }

1. @Id

* @Id annotation is used to specify the primary key of an entity class. It marks a field as the unique identifier for instances of that entity
* In relational database, a primary key uniquely identifies each row in a table and hibernate uses this annotation to map the java objects primary key to the database primary key
* This annotation is used with field (attribute) within the entity class
* Syntax :-

@Entity

@Table(name = "table\_name")

class ClassName {

@Id

private int empid; }

1. @Column

* @Column is used to map the field with table column
* It is used with field (attribute) within the entity class
* Agar class ko @Entity aur @Table se annotate kiya gaya hai, to @Column ka use tab zaroori nahi hota jab property name aur column name same ho.
* Lekin agar database column ka naam property name se different hai, to @Column(name = "column\_name") use karna padta hai.
* Syntax :-

@Entity

@Table(name = "table\_name")

class ClassName {

@Column(name = "column\_name")

private String propert\_name; }

* Table v/s Entity and @Table v/s @Entity

|  |  |  |  |
| --- | --- | --- | --- |
| Table (DB) | Entity (Java Class) | @Table Annotation | @Entity Annotation |
| A physical structure in the database | A Java object representing a row in a table | Maps an entity to a specific DB table | Marks a Java class as a JPA entity |
| Exists in the database | Exists in Java code (usually in a Hibernate/JPA app) | Used in Java code (with JPA/Hibernate) | Used in Java code (with JPA/Hibernate) |
| Stores data in rows and columns | Acts as an in-memory representation of DB rows | Customizes the table name and other settings | Declares the class as an entity to be persisted |
| No default behaviour table must be created. | Class name is mapped to table name by default | If not used, table name = class name | If not used, class is not treated as an entity |
| ORM maps data to this | ORM uses entities to persist and fetch data | Helps ORM find the correct DB table | Enables ORM to recognize and manage the class |

* Code to automatically generate / increment id

Here identity and IDENTITY indicates that the persistence provider must assign primary keys for the entity using a database identity column.

1. Using MySQL

col\_name datatype auto\_increment

1. Using XML Mapping

<id name="stdid" column="std\_id"> <generator class="identity" /> </id>

1. Using Annotations

@GeneratedValue(strategy = GenerationType.IDENTITY)

* Note Points

1. Config file ke mapping tag ka attribute "class" use krna "resource" nahi

**CRUD**

* Insert
* When save() method insert the record in database, it will return primary key of the saved object in serializable form - Serializable save(Object object);
* When persist() method insert the record in database, it will not return any value - void persist(Object object);
* Retrieve
* Retrieve operation me hum log transaction ka concept use nahi krte. bcoz need nahi hai commit and rollback ki as hum bus select hi kr rhe hote hai.
* get(class,pri\_key) method

1. The get() method is used to retrieve an object from the database by its primary key (identifier)
2. If the object with the given identifier is found in the database, it will return the whole actual object and initializes all the properties with data from database. This is known as "eager loading" or "early loading". sout se reference print karoge to object return karega.
3. If the object is not found in the database it will return null object

* load() method

1. The load() method also retrieves an object by its primary key but it return a proxy object rather than the actual object from the database
2. It will retrieve the values from database only when getter methods are called. This is known as "lazy loading". sout se reference print karoge to proxy reference return karega & select command execute nahi karega.
3. If the object is not found in the database it provides an exception i.e. "ObjectNotFoundException"

* Update
* direct update use kiya like update(stud) to yah given values insert kr dega and baki sab null kr dega jo given nahi thi joki acha way nahi hai
* jis object(row) ko update krna hai usko pahle get krwao session se >> required value set krdo using setter >> use update(stud) with that object
* Difference Table

|  |  |  |  |
| --- | --- | --- | --- |
| Aspect | update() | saveOrUpdate() | merge() |
| Purpose | Updates an existing record | Updates if exists, inserts if not | Copies values from a detached object into a managed one |
| Session Dependency | Needs object attached to session or fetched from session | Can work with detached objects, but still sensitive | Designed to work safely with detached objects |
| Insert if Not Exist | Throws exception if record doesn’t exist | Will insert if object is new (ID not in DB) | Same as saveOrUpdate() behavior |
| Returns | void | void | Returns a managed instance (very important) |
| Multiple Sessions Issue | Will throw NonUniqueObjectException if same entity is already in session | Same issue as update() | No problem – it resolves detached conflicts internally |
| Common Use Case | When you are sure entity is persistent & attached | When you're unsure if entity exists in DB or not | When working with detached entities (common in web apps, REST APIs) |

* Delete

|  |  |  |
| --- | --- | --- |
| Aspect | delete() | remove() |
| Availability | Hibernate-native (older API) | Introduced via JPA (EntityManager.remove()), supported by Hibernate 5+ and preferred in Hibernate 6+ |
| Return Type | void | void |
| Standard | Hibernate-specific | JPA standard method |
| Object State Required | Entity must be attached to session | Same – entity must be managed |
| Behavior | Deletes the row from DB if entity is in session | Same behavior, but better aligned with JPA and Hibernate 6 |
| Exception if Detached | May throw exception if entity is detached and not reassociated | Same – needs managed entity (use find() or merge() before) |
| Use Case | Legacy or old Hibernate code | Modern apps using Hibernate 6 / JPA compliance |

* Note Points
* XSD is the industry standard for modern XML schema validation and configuration. DTD is mostly obsolete.
* Hibernate design ke mutaabiq primary identifier ko update karna allowed nahi hai.

**SPRING HIBERNATE**

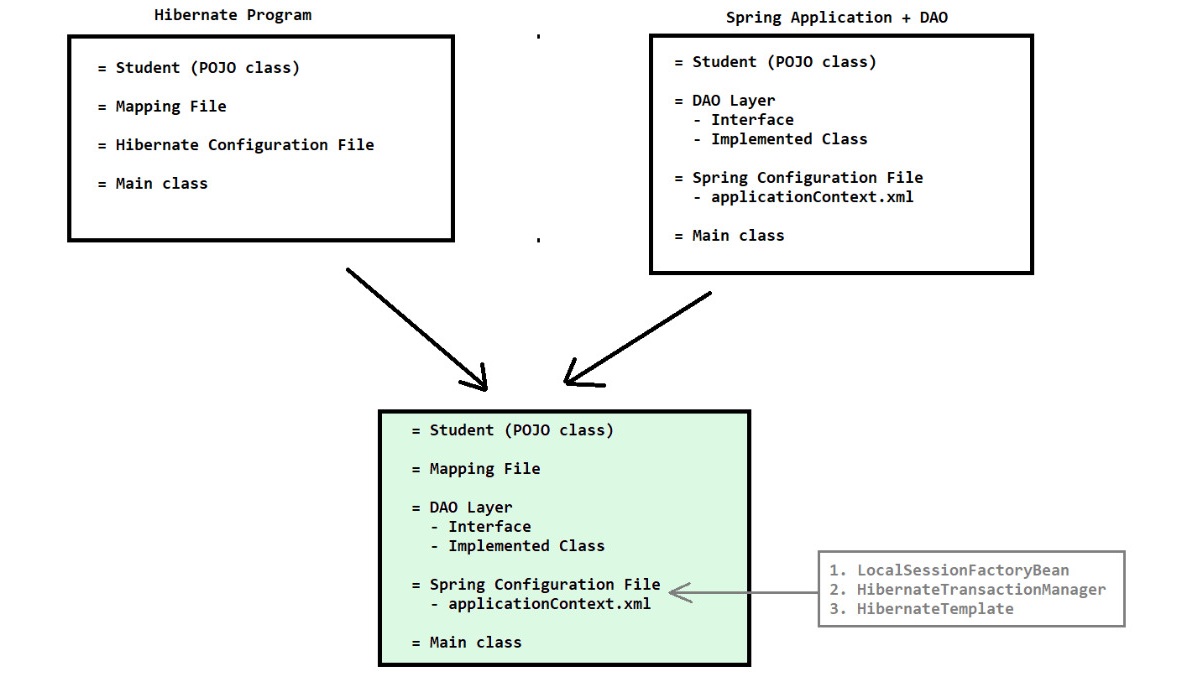
* Spring with Hibernate
* Spring with Hibernate combines the strength of both frameworks to create efficient, maintainable and scalable java application
* Using hibernate with spring will simplifies the configuration, enhances transaction management etc by removing the boiler-plate code i.e. creating Configuration, SessionFactory, Session & Transaction objects
* To achieve spring with hibernate integration, spring has provided 3 classes :-

1. LocalSessionFactoryBean
2. HibernateTransactionManager
3. HibernateTemplate

* Requirments –

1. Spring Core 5.3.10 , Spring Context 5.3.10 , Spring ORM 5.3.10 , Hibernate Core Relocation 5.5.7 , Jakarta Persistence API (JPA) 3.1.0 , (JTA) Jakarta transaction API 2.0.1 , MySQL Connector J 8
2. Use Javax package not Jakarta , JDK 17 (eclipse wala use kr lena)
3. Use mentioned version only they are dependent on each other hence incompatibility between their interaction may occur on incorrect selection. ClassNotFound MethodNotFound , … such errors can arise on incorrect selection.

* Diagram



* LocalSessionFactoryBean
* Spring context file ke ander hum config provide karnge and hum session factory ka object get krwange localsessionfactorybean k through.
* It is a class which simplifies the process of configuring hibernate within spring application
* Use :-

1. Configuration :- It allows us to define and configure hibernate properties such as database connection details, dialect, mapping resources etc in spring context file in more simplified manner.
2. SessionFactory creation :- It is responsible for creating and initializing the hibernate SessionFactory based on the provided configurations

* The SessionFactory is a component in hibernate that manages the lifecycle of database connections and provides a central point for creating and managing database transaction.
* By using LocalSessionFactoryBean, we can obtain a fully configured SessionFactory instance to work with hibernate in our spring application
* session factory database connection & transaction ke liye bahut important hota hai.
* Property file like values ke liye hum <props> ka use krte hai ; hibernate.prop\_name krke key & value use krna.
* java config file ko mapping se link krne kliye use property with name packagesToScan
* HibernateTransactionManager
* Used for hibernate/database transaction
* It is a class that provides an abstraction layer for database transactions when using hibernate as a persistence framework
* Use :-

1. Transaction Management :- It simplifies hibernate transaction management, synchronizes hibernate session with spring-managed transactions, binding sessions to threads and managing their lifecycles during commits and rollbacks
2. Declarative Transaction Management :- Enables usage of annotations to implement tx process. It also supports declarative transaction management via spring's "@Transactional" annotation which simplifies the code by automating transaction handling, enhancing readability and maintainability

* Jo session create hue hai unko yah bind krta hai , tx s related jo lifecycle hoti hai like kab hume commit krna hai & kab rollback yh sab aare part yah manage krta hai.
* HibernateTransactionManager iska bean id ki value hivernateTransactionManager hi rkhna yah fix hai kuch aur likha to error dega
* HibernateTemplate
* Database se interact krne k liye methods provide krta hai
* opening and closing of session is also managed by this
* It is a class that provides a simplified and consistent way to interact with hibernate ORM framework
* Use :-

1. Simplified Data Access :- It is an abstraction layer over hibernate "Session" which provides the methods for CRUD operations i.e. save(), update(), delete(), get() etc
2. Exception Translations :- It provides exception translation i.e. converting hibernate-specific exceptions into Spring's DataAccessException hierarchy

* It abstracts away the need to manually open and close hibernate sessions
* Note Points

1. aop ya fir tx wala schema copy paste krna xml config me -- chapter 16 which is before 40.2.7 [transaction search krna]
2. HibernateTeamplate and HibernateTransactionManager k ander hume session factory object pass krni padegi
3. Since hibernate bina transaction k kaam nahi krta hence DAO ki implementation class ko @Transactional annotate kr dena + ( xml config file m <tx:annotation-driven /> , java config file me @EnableTransactionManagement ) add kr dena to activate transaction mode warna read-only mode me hai bolke exception dega. Transaction annotation insert update and delete k liye hi use hota hai select k liye need nahi hai.
4. xml config + anno mappin valid combo. bean id name should be in camelCase ;
5. sessionfactory ka obj banana k liye datasource lgta hai , txManager & txTemplate ka object banana k liye sessionfactory ki need pdti hai.
6. BigDecimal :- MySQL -- marks DECIMAL(10, 2) , Hibernate Mapping -- <property name="marks"> <column name="marks" precision="10" scale="2"/></property> , Annotation -- @Column(precision = 10, scale = 2)
7. In Xml Config File Spring automatically converts the string value "765.89" to a BigDecimal using its built-in type conversion mechanism (via PropertyEditor or ConversionService).
8. Use Interface-Based Injection - promotes loose coupling ; Spring AOP by default creates proxy objects based on the interfaces implemented by your bean and work perfectly when your code and configuration expect the interface.
9. Always use .getObject() when you're manually using a FactoryBean like LocalSessionFactoryBean. Calling .getObject() gives you the actual SessionFactory from the bean. Think of LocalSessionFactoryBean like a tea bag and SessionFactory like the tea.
10. bean name if not explicitly defined will be same as class / method name in camel case
11. HibernateTemplate , HibernateTransactionManager and LocalSessionFactoryBean are OUTDATED.

**JPA**

**INTRODUCTION**

* Java Persistence API (JPA)
* JPA was released in May 2006 as the part of JavaEE-5 (Enterprise Edition)
* JPA is specification that simplifies the interaction between java "Objects" and "Relational Database"
* Hibernate java ke basis pr banaya gaya tha but hibernate java ka part nahi hai.
* JPA is only specification, its implementation is provided by ORM vendors i.e. JBoss (hibernate), Apache Software Foundation (Open JPA), Eclipse Foundation (EclipseLink) etc
* To use JPA, we have to use any one ORM tool for eg. hibernate, EclipseLink etc
* Advantages of JPA

1. Abstraction over database :- JPA provides a higher-level, object-oriented way to interact with databases, allowing developers to work with java objects rather than dealing with SQL queries and database-specific code
2. Improves Productivity :- By using JPA, developers can focus on business-logic rather than database interaction, leading to faster development and reduce the code complexity
3. ORM (Object-Relational Mapping) :- JPA enables ORM which means it maps Java Objects to Database and vice versa. This simplifies the storage and retrieval of java objects in database eliminating the need to write low-level SQL queries. JPA allows to developers to switch between different ORM vendors
4. Database Portability :- JPA abstracts away the database-specific details, making it easier to switch between different databases
5. Scalability :- JPA enables the development of scalable applications by managing database connections, pooling, optimizing performance etc

* JDBC (Java Database Connectivity)
* JDBC is an API which is used to store the data in database
* JDBC is only specification, its implementation is provided by database vendors i.e. MySQL, Oracle, PostgreSQL etc
* ORM (Object Relational Mapping)
* ORM is programming approach or functionality used to provide the relationship between "Objects" and "Relational Database" by using xml mapping file or annotations
* Hibernate
* Hibernate is an ORM tool which is used to store the objects in database
* It follows ORM approach
* JPA (Java Persistence API)
* JPA is an API which is used to store the data in database (but it follows ORM approach)
* JPA is only specification, its implementation is provided by ORM vendors i.e. JBoss (hibernate), Apache Software Foundation (Open JPA), Eclipse Foundation (EclipseLink) etc
* Specification
* Specification means guidelines or blueprint or interfaces for how a particular technology or API should work
* Specification means formal document that defines a set of rules, behaviour and interfaces that software application implements
* Persisting an Entity
* Persistence means "store permanently"
* Entity means "objects"; Bean = Object = Entity
* Persisting an entity means "storing the object in database permanently"

**JPA PROGRAM PROCEDURE**

* JPA
* JPA is an API which contains interfaces and classes which are as follows :-
* Following are Some of the Classes and Interfaces

1. Persistence <class> :- It provides a static method i.e. createEntityManagerFactory() for obtaining EntityManagerFactory instances.
2. EntityManagerFactory <interface> :- It is used to create EntityManager instance for database operations
3. EntityManager <interface> :- It is used to perform crud operation. It manages the lifecycle of entities and provides methods for CRUD operations.
4. EntityTransaction <interface> :- It is used for transaction management including starting, committing and rolling back the transactions
5. Query <interface>
6. TypedQuery <interface>
7. CriteriaBuilder <interface>
8. CriteriaQuery <interface>

* All above classes and interfaces are present in "javax.persistence" package
* The "Java Persistence API (JPA)" name has been changed or rebranded to "Jarkarta Persistence API (JPA)" in 2019
* Procedure

1. Download and Install any one IDE (eclipse)
2. Create "Simple Java Project" or "Maven Project"
3. Add "jars" (for simple java project) or "provide dependencies" (for maven project)
4. Create POJO (Plain Old Java Object) class
5. Create mapping file (or provide annotations in POJO class)
6. Create Persistence Unit Configuration (persistence.xml) in META-INF folder (yahi location per create krte hai yah standard hai)
7. Create main class and execute the application
8. Commit the transaction
9. Close the resources

* Steps in Main Class

1. Create EntityManagerFactory object

EntityManagerFactory emf = Persistence.createEntityManagerFactory("my-persistence-unit");

1. Create EntityManager object

EntityManager em = emf.createEntityManager();

1. Create EntityTransaction object and begin the transaction

EntityTransaction et = em.getTransaction(); // crt entity tx

et.begin(); // starts tx

1. Perform database operations using EntityManager instance

Insert operation -- persist()

Retrieval operation -- find()

Update operation -- merge()

Delete operation -- remove()

* Note Points

1. All the above CRUD operation methods are used to work with individual record or to perform single record manipulation only
2. If we want to work with multiple records (perform manipulations with multiple records) like select, insert, update, delete multiple records then we have to use

* JPQL (Java Persistence Query Language)
* Native SQL
* JPA Criteria API (less used)

1. mapping file name -- student.xml (tableName.xml confirm it) dont use hibernate like name.hbm.xml
2. create a folder named src/main/resources in that provide the META-INF folder. Yah Convention hai and other reason yah h ki container ess location ko automatically scan kr deta hai.
3. Search "persistence .xml file schema" then go on appropriate website and choose the .xsd file named something like persistence , Schema paste krne k baad <persistence-unit> yh tag add karega to error chala jayega ess tag me name kuch bhi provide kr skta hai.
4. properties me jo db config details provide kr raha hai uske name attribute me javax.persistence.jdbc. prefix kr dena. properties me jo hibernate prop provide kr raha hai uske name attribute me hibernate. prefix kr dena
5. Java based mapping provide kiya to persistence.xml me jo mapping tag use kiya tha uski need nahi hai. Select operation me transaction ki need nahi hai.
6. Update procedure -- pahle row get kiya then set kiya then merge. But yaha set krne pr hi wah automatically upadate kr dega hence merge becomes option directly commit
7. Catalog -- Internal ; Artifact -- maven-archetype-quickstart ;Use JDK 17 , javax & not jakarta

**JPQL**

* JPQL
* JPQL is a query language used for querying and manipulating java objects with relational database
* It allows the developers to write the queries in an object-oriented way, treating database records as objects
* SQL table per kaam krti hai but JPQL objects per kaam krti hai jo tables se link hote hai
* There is no INSERT query in JPQL. If we have to insert new entity in database then we have to use EntityManager [persist()] or native SQL query
* There are many other queries i.e. COUNT, JOIN, GROUP ID, ORDER BY, DISTINCT etc
* Each JPQL query should be specific to a single entity. (1 query per class only)
* Some JPQL queries are as follows :-

Here en -- alias name ; EntityName -- Entity(class name) ka naam (not table name) [ @Entity class Student {---} ]

1. Select query :-

SELECT en FROM EntityName en

SELECT en FROM EntityName en WHERE en.attribute = :value

1. Update query :-

UPDATE EntityName en SET en.attribute = :newValue WHERE en.id = :entityId

1. Delete query :-

DELETE FROM EntityName en WHERE en.id = :entityId;

* Advantages

1. Advanced Querying :- If we want to use complex queries i.e. joins, aggregations etc then we can use JPQL (as standard CRUD operations are not suitable)
2. Improve Performance :- If we are fetching large datasets or executing complex queries then JPQL is used to improve the performance (as standard CRUD operations reduces the performance)
3. Aggregations :- JPQL can perform aggregation functions eg SUM, COUNT etc for reporting and analytics
4. Type Safety :- JPQL provides type-safety which is helpful for catching the errors at compile time
5. Other benefits :- JPQL provides more flexibility, enabling custom queries and advanced filtering for a wider range of use case

* Query Interface
* It is the fundamental JPA interface that represents a database query
* It is used to execute dynamic queries including both JPQL and native SQL queries
* It allows parametrization, result retrieval and is used for various querying operations
* TypedQuery Interface
* It is a specilized subtype of the Query interface of JPA
* It enhances type safety by demanding us to specify the expected result type when creating a query. This ensures that query results are automatically cast to the specified type, reducing the risk of type-related errors.
* It is useful when working with entity types in JPA as it eliminates the need for explicitly casting when retriving the results
* Note Points

1. jpql query execute krne ke liye use createQuery(String sql) method of EntityManager
2. Then use getResultList() method from Query to get the result set of the query executed.
3. [Hum jadatr Query ki jagah TypedQuery use krte hai when getting data from single table]
4. Like Parameterized query using map we provide the param :p1 , :p2 and then set them with value (key name should be same)
5. Ager ek hi object/row get hone wali hai then you can use getSingleResult() instead of getResultList() and replace list with appropriate class.
6. If only selected columns hi get kr rhe ho out of all then use Object[] as a reference variable and sql command me sirf alias(\*) ki jagah alias.prop\_name (col\_name)
7. JPA wale standard methods se hum log advance sql operation perform nahi\* kr skte
8. NO SEMICOLON AT END LIKE SQL CMD
9. Update/delete queries cannot be typed
10. JPQL DMLn (insert, update, delete) queries do not return entities or result sets. They return the number of affected rows. Therefore, Hibernate prohibits using methods like .list() or .getSingleResult() on them [use them only in case of select statement]
11. Execute konse line per ho raha hai -- executeUpdate() for update and delete and createQuery() for select operation.

**NAMED QUERIES**

* Named Queries in JPA
* Named queries in JPA are queries to which we assign a particular name, making them easier to reference and reuse throughout our application
* We can declare named queries in an entity class or in xml file
* Syntax :-

1. In an entity class using annotations :-

@NamedQuery(name="queryName", query="----JPQL-----")

public class Student { ----- }

1. In xml file (orm.xml) using tags :-

<entity-mappings ... >

<named-query name="queryName">

<query>

----JPQL-----

</query> </named-query> </entity-mappings>

* Advantages

1. Code Reusability :- Named Queries can be reused in different parts of our application, reducing the code duplication. This enhances the consistency and readability of our code
2. Code Maintainability :- Named Queries centralize query definitions, making it easier to maintain and update queries across our application. When a query needs modification, we can update it in one place and all the references to that query will automatically updated
3. Type Safety :- Named Queries are checked for correctness at compile time, which helps to catch the errors early in the development process

* Note Points

1. Named queries in JPA are primarily used for selecting data (SELECT query). These are not used for inserting, updating or deleting data (INSERT, UPDATE or DELETE queries)
2. Use createNamedQuery("name", type) for execution; select operation ke case me transaction ki need nahi hoti
3. Yah named queries bhi jpql ka hi part hai
4. Xml file case

* create file in META-INF folder and paste the schema from the file named orm\_version.xsd it is available on the same website as of persistence schema file.
* Schema paste krne k baad you will see error but yah error saare above mentioned tags (named-query and query) use krne pr error chala jayega.
* Yah file ko link kr dena using mapping tag in persistence.xml file

**NATIVE SQL**

* Native SQL Queries in JPA
* "Native SQL Queries" refers to the SQL query that is specific to a particular database like MySQL, Oracle etc
* Native SQL queries are specific to the database which make them database-dependent (But JPQL are database-independent query language)
* Native SQL queries are very flexible which allows us to write complex queries that may not easily expressible in JPQL
* In case of native SQL queries, we use "Query" interface rather than "TypedQuery" interface because native SQL queries don't return managed JPA entities and therefore there is no entity type to specify as the result type
* Note Points

1. Complex sql queries likha using JPA thoda jada difficult hota hai hence instead of using jpa in such we use native sql queries jaha hum native queries likhte hai (like MySQL cmds)
2. Usage of semicolon in JPQL query will raised an error but in Native JPA query the semicolon usage is optional (will not raise error if used or skipped)

**SPRING JPA**

* Spring with JPA
* Spring with JPA combines the power of the Spring Framework with the JPA to build robust and maintainable java application
* It simplifies configurations, enhances transaction management and reduces the boilerplate code for managing database interaction
* Key components for Spring with JPA
* LocalContainerEntityManagerFactoryBean

1. It is a class which is responsible for creating and configuring the EntityManagerFactory, which is the central interface for working with JPA
2. It allows for easy setup of JPA providers like hibernate, eclipselink etc

* JpaTransactionManager

1. It is a class which is used for managing transactions in spring-managed JPA environment
2. It coordinates transactions across one or more EntityManager instances
3. iski id name fix hai transactionManager
4. isko entity manager factory ki need hoti hai

* @PersistenceContext

1. It is a JPA-specific annotation used in the JPA to inject EntityManager into spring-managed bean or components
2. iska use krne se pahle isko invoke krwana pdta hai [xml-cntxt:anno-config ; java-automatically ho jayega]

* Steps to create Spring with JPA program

1. Create maven project and provide dependencies
2. Create an entity class in "in.yoy.entity" package
3. Create Spring configuration file (xml) in "in.yoy.pro\_name.resources" package

* DataSource DriverManagerDataSource
* LocalContainerEntityManagerFactoryBean
* dataSource
* package to scan the entity class [Takes value in String... hence provide it in List]
* Specify the jpa vendor (adapter)
* provide the jpa vendor properties (required values should be in type Properties which is in key-value pair)

1. Create DAO interface in "in.sp.dao" package
2. Create DAO implemented class in "in.sp.dao" package
3. Create main class and execute the application

* Dependencies
* spr core -- Any 5th version -- 5.3.20
* spr ctxt -- Any 5th version -- 5.3.20
* spr orm -- Any 5th version -- 5.3.20
* javax.persistence-api -- 2.2 version
* hivernate core reloc -- 5.6.15 version
* MySQL Connector J -- 8.1.0 version
* JDK – 17
* Note Points

1. Xml Schema 40.2.8 (for ctx) + 40.2.6 (for tx)
2. select ko try block ke ander rakha because ager usko object nahi mila to wah exception raise krta hai.
3. Tx (@Transactional) ka use krne se pahle isko invoke krna pdta hai [xml-tx:anno-drvn ; java-@EnableTransactionManagement]
4. @Bean methods ke liye ager kuch parameter rakh raha hai then wah automatically provide kr dega spring
5. me purane wale codes me update() dao me obj ku le raha tha ? -- Logic yah tha ki dene wala object modify krke dega and hum bus uss modified object ko DB me update krdnge
6. <bean class="in.yoy.SprJpaXml.dao.DeptDaoImplement" id="dao" /> <!-- Yaha implementated class hi rkhna interface nahi kuki yaha mentioned class ka hi object bnta hai
7. As noted in JPA documentation, one must check and reattach entities before deleting : use em.contains(entity) or re-fetch via find() or merge(). If not, the JPA provider will reject the remove() ; Hence use this approach if (!em.contains(dept)) { dept = em.merge(dept); } em.remove(dept);
8. In JPA, remove(entity) only works on an entity that’s still managed in the current transaction. A call to persist() or merge() makes an instance managed only until that transaction ends—after commit it becomes detached. If you pass a detached object to remove(), you’ll get an IllegalArgumentException.
9. DriverManagerDataSource ka hi ref rkhna as DataSource me required methods nahi hai
10. @Bean public DataSource dMDataSource() { DriverManagerDataSource dataSource = new DriverManagerDataSource() ; ... }
11. Object as a property value aese pass kare Xml Config me -- <property name="prop-name"> <bean class="Fully-Qualified-Class-Name" /> </property>

SPRING DATA JPA