What is jdbc Api?

It ha

-----s set of classes and interfaces provides the rules.

The JDBC API is comprised of two packages:

* [java.sql](https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html) : **Basic database operations** (what every app needs). part of**Java Standard Edition (Java SE)**

1. [javax.sql](https://docs.oracle.com/javase/8/docs/api/javax/sql/package-summary.html): **Advanced features** (for scalable apps, Java EE, etc.). part of**Java Enterprise Edition(Java EE)**

**JDBC Driver Types Explained Simply**

JDBC drivers are like "translators" that help Java programs talk to databases. There are **4 types**, each working differently:

The actual classes and interace is implementation is done by normal code

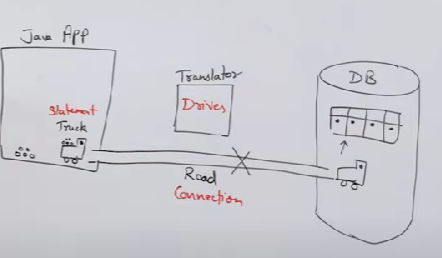
| **Driver Type** | **Example Drivers** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| **Type 1 (Bridge)** | sun.jdbc.odbc.JdbcOdbcDriver | Works with ODBC databases | Deprecated, Slow, Not Portable |
| **Type 2 (Native)** | Oracle OCI, IBM DB2 CLI | Faster for specific DBs | Needs native libs, OS-dependent |
| **Type 3 (Middleware)(firewall)** | Oracle RMI, DataDirect | Pure Java,db is behind firewall | Slower (extra middleware layer) |
| **Type 4 (Pure Java)** | MySQL Connector/J, PostgreSQL JDBC | Best for modern apps (recommended) | DB-specific |

Type 4 🡪 com.mysql.jdbc.Driver

Drivermanager and datasource 🡪 for db connections

* Visual repersentation of jdbc:

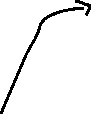
Part of jdbc api



Returns result(execute return result)



Execute part



* Rules for connecting database and java. (basic approach)

1. Register and load the Driver class (mysql-5)

//Basic syntax

Class.forname(“Driver name” ); 🡪 driver is dynamic based on the db we use

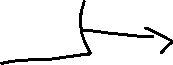
For mysql-5 :

* Java has it own class of Type Class.
* Class has forname() which is static
* **Forname()** -> used to load driver or any library of a class in our application. Driver class



* Class.forname(“com.mysql.jdbc.Driver” );

Both present in mysql jar(java archive) : i.e collection of all java class in zip



Pakage name for driver



1. Create the connection using drivermanager(use classes and interface of java mysql package)

*Connection con* = DriverManager.*getConnection*(“url,”username”,”password””)

➡️ This is like **calling your friend before talking**.

* DriverManager(class) -- > to create connection using getConnection().
* getConnection()🡪 static method of Driver class return Connection(interface)

Port by default of mysql



* url: jdbc:mysql://localhost:3306/DB name



Where we are using our db

Which dB



Server name (localhost) means my own device

1. To build query

Sql query to insert delete update etc

1. To create statement: various types here for Statement

* Connection(interface) has createStatement() : use to open Statement.
* createStatement() 🡪 returns statement interface.
* This **creates a Statement object** (stmt), which allows you to send SQL queries to the database.
* ➡️ This is like **preparing what you want to say in a message**.

1. Execute Statement

* Statement(interface) has execute method to execute query.
* Execute() 🡪 returns result
* ➡️ This is like **asking your friend for a list of employees**.

1. Close

Connection(I):

this interface extends autocloseable,wrapper:Its primary role is to manage the **communication session** between your Java application and the database. SQL statements are executed and results are returned within the context of a connection.

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

[SQLException](https://docs.oracle.com/javase/8/docs/api/java/sql/SQLException.html) - if a database access error occurs or the url is null

[SQLTimeoutException](https://docs.oracle.com/javase/8/docs/api/java/sql/SQLTimeoutException.html) - when the driver has determined that the timeout value specified by the setLoginTimeout method has been exceeded and has at least tried to cancel the current database connection attempt.

Statement(I) :

The object used for executing a static SQL statement and returning the results it produces.

Results are on the bases of resultset

public interface **Statement** extends [Wrapper](https://docs.oracle.com/javase/8/docs/api/java/sql/Wrapper.html), [AutoCloseable](https://docs.oracle.com/javase/8/docs/api/java/lang/AutoCloseable.html)

createStatement()

Creates a Statement object for sending SQL statements to the database.

CallableStatement(I): for stored procedures.

Preparecall().

PreparedStatement : for dynamci values.

* Different methods use execute query part

1. execute()

* **Description:** The method used for all types of SQL statements, and that is, returns a Boolean value of TRUE or FALSE.
* **Return type:** This method return a Boolean value. TRUE indicates that query returned a Result Set(Collection of data having rows and column) object(for select satement) and FALSE indicate returned an int value(DML/DDL) or returned nothing.
* **Usage:** This method is use to execute  Both select and non select queries.
* **Example**: All SQL statements.

1. execute Query()

* **Description:**  this method execute statements that returns a result set by fetching some data from the database.
* **Usage:** This method is use to execute select query.
* **Return type:** This method returns a Result Set object which contains the result returned by query.
* the query returns no rows. In such cases, the Result Set object will still be created, but it will be empty. (for all)
* One of it’s example that is widely common: ‘SELECT’

1. execute Update()

* **Description:** This method is used for execution of DML statement(INSERT, UPDATE and DELETE) which return int value, count of the affected rows.
* **Usage:**This method is use to execute non select query
* **Return type:** An integer value which represent number of rows affected by the query. This will be 0 for statement which are returning nothing.
* **Example**

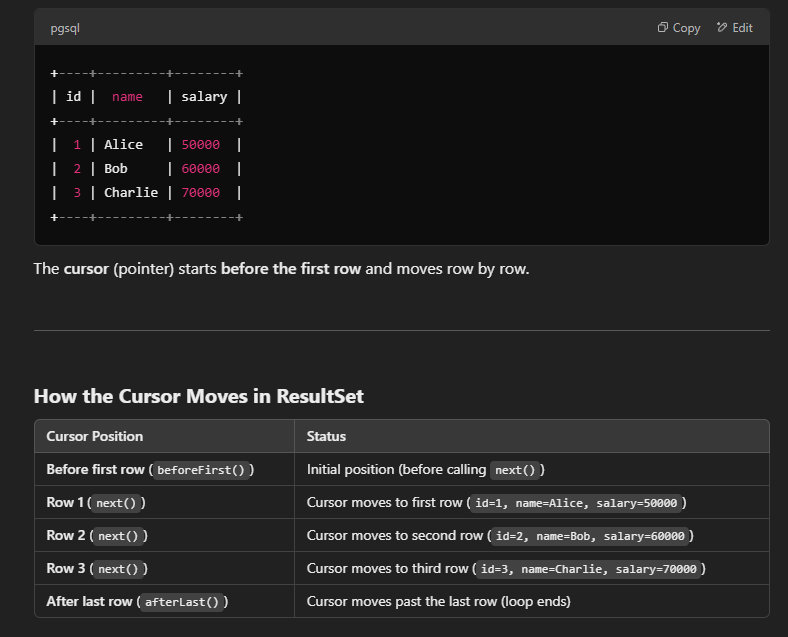
**DDL (Data Definition Language)** statements like CREATE TABLE, ALTER TABLE, DROP TABLE do **not** affect rows—they modify the database structure. Hence, they return 0.

* The ResultSet object has a cursor/pointer which points to the current row. Initially this cursor is positioned before first row.

DML->INSERT , UPDATE and DELETE

DDL-> CREATE, ALTER

* the ResultSet object holds the tabular data returned by the executeQuery() that execute the statements



The next() method of the ResultSet interface moves the pointer of the current (ResultSet) object to the next row, from the current position.

* i.e., on calling the next() method for the first time the result set pointer/cursor will be moved to the 1st row (from default position).

ResultSet object 🡪 cannot be serilizable(data transfer ke time issue)

Q : Can we use execute() for insert delete

* Yes we can but we wont get count.
* JDBC interfaces or types of statement
* Statement(I)
* The object used for executing a static SQL statement and returning the results it produces.
* Results are on the bases of resultset
* public interface **Statement** extends [Wrapper](https://docs.oracle.com/javase/8/docs/api/java/sql/Wrapper.html), [AutoCloseable](https://docs.oracle.com/javase/8/docs/api/java/lang/AutoCloseable.html)
* createStatement()
* For hard-coded values
* Mostly used for DQL commands
* Fire command repeatdly
* It is an parent interface for all other types of statements
* HAS CREATESTATMENT

1. Prepared Statement(I)

[PreparedStatement](https://docs.oracle.com/javase/8/docs/api/java/sql/PreparedStatement.html) prepareStatement([String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) sql)throws [SQLException](https://docs.oracle.com/javase/8/docs/api/java/sql/SQLException.html)

* Dynamic values
* DML coammnds
* Fire command repeatdly
* Child interface of Stament
* HAS PREPARECALL
* setInt(index,value),setNext(index,value)

example query:insert into orders values(?,?)

1. Callable statement

[CallableStatement](https://docs.oracle.com/javase/8/docs/api/java/sql/CallableStatement.html) prepareCall([String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) sql)throws [SQLException](https://docs.oracle.com/javase/8/docs/api/java/sql/SQLException.html)

* For stored procedure i.e for pre-compiled query
* It is more efficient
* Child interface of Statement

* Single-ton desing pattern
* Class having only one instance(object) and provides global point to access it.
* IMPLEMENTATION
* PRIVATE CONSTRUCTOR()-> Other classes cannot make instance of that class using new keyword like eg: new MyConnection().
* CREATE METHOD -> make this method static public so that it can be globally accesed.
* STATIC PRIVATE VARIABLE -> make global var as static to use it in static method

* JDBC Drawbacks /disadvantages
* Everyline throws checked(compile time)exception
* Boiler plate code
* Query dependent language like oracle DB toh uske hi query chalega
* Database dependent lang
* Hardcode/manual query
* Performance low
* Code readiability low

How drivers are loaded .

**Legacy (**Class.forName()**)**

* **Pre-JDBC 4.0 (Before Java 6)**
* **Manual driver loading**:

Class.forName("com.mysql.jdbc.Driver"); // Explicit registration

Connection conn = DriverManager.getConnection(url, user, pass);

* **Drawbacks**:
  + Requires knowing driver class name
  + Verbose, tightly coupled
  + Risk of ClassNotFoundException

**Modern (JDBC 4.0+ ServiceLoader)**

* **Java 6+ (Automatic Registration)**
* **Just add driver JAR to classpath**:

Connection conn = DriverManager.getConnection(url, user, pass); // No Class.forName()

* **How it works**:
  + Uses META-INF/services/java.sql.Driver in driver JARs
  + ServiceLoader auto-discovers and registers drivers
* **Benefits**:
  + Cleaner code
  + No hardcoded driver names
  + Secure (prevents SQL injection via proper parameterization)

| **Feature** | **Statement** | **PreparedStatement** | **CallableStatement** |
| --- | --- | --- | --- |
| Purpose | Static SQL | Parameterized SQL | Stored procedures |
| Performance | Low | High | High |
| SQL Injection | Vulnerable | Protected | Protected |
| Parameter Types | None | IN only | IN and OUT |
| Reusability | Low | High | High |
| Syntax Example | "SELECT..." | "SELECT...WHERE x=?" | "{call proc(?)}" |

Why Statement is prone to SQL injection?

**🚨 Using**Statement**(UNSAFE)**

* **Input:** ' OR '1'='1 (prop), or ' OR 'a'='a
* **Query Becomes:**

SELECT \* FROM users WHERE username = '' OR '1'='1';

* **Result:** Returns **ALL RECORDS** (because '1'='1' is always true).
* **Risk:** Hackers can **steal data, delete tables, or bypass logins**.

**✅ Using**PreparedStatement**(SAFE)**

* **Input:** ' OR '1'='1
* **Query Stays Protected:**

SELECT \* FROM users WHERE username = ?; → (Parameterized)

* **Result:** Searches for **literal text**' OR '1'='1 (no injection).

**Key Takeaway**

🔹 Statement**= Dangerous** (concatenates input directly into SQL → **hackable!**).  
🔹 PreparedStatement**= Safe** (treats input as data, not code → **blocks attacks!**).

**Always use**PreparedStatement**for security!** 🔒

**What is a**DataSource**?**

* It’s a **better way to get database connections** than DriverManager.
* **Why?**
  + No hardcoding usernames/passwords in code.
  + Supports **connection pooling** (reuse connections = faster).
  + Supports **distributed transactions** (multiple databases in one transaction).

DataSources extends CommonDataSource,Wrapper

* System admin does this type of operation using apache tool,or oracle weblogic server
* The actual EE level application internally uses datasource for db connection.
* connection pooling and distributed transactions.

How many ways we can implement dataSources

* A basic DataSource implementation produces Connection objects that are not pooled or used in a distributed transaction.
* A DataSource implementation that produces Connection objects that participate in connection pooling, that is, connections that can be recycled.
* A DataSource implementation that supports distributed transactions produces Connection objects that can be used in a distributed transaction, that is, a transaction that accesses two or more DBMS servers.

**Extended Comparison: DriverManager vs DataSource**

| **Feature** | **DriverManager** | **DataSource (with Connection Pool)** |
| --- | --- | --- |
| **Connection Creation** | Creates new physical connection each time | Borrows from pool of pre-established connections |
| **Connection Lifetime** | Closed when close() is called | Returned to pool when close() is called |
| **Performance** | Slower (new TCP handshake per request) | Faster (reuses existing connections) |
| **Thread Safety** | Not thread-safe if shared | Thread-safe (pool manages connections) |
| **Resource Management** | Manual management required | Automatic connection recycling |
| **Configuration** | Hardcoded in application | External configuration possible(properties files) |
| **JNDI Support** | No | Yes |
| **Typical Usage** | Simple applications, testing | Production systems, web applications |

HiKariCp Connection pool 🡪 developed by brett wooldridge in 2012.

HikariCp 🡪 uses logger to print message by default uses slf4j

Datasource is only sipported by java 11+

Datasource prints logs. 🡪 hikariDatasource

JARs required for datasource

Needs slf4j jar

Slf4j

Hikaricp60

Mysqlconnector

1. **slf4j-api-2.0.13.jar** (the interface)
   * This is the logging facade that HikariCP directly depends on
   * HikariCP calls SLF4J API methods for all its logging needs
   * It doesn't know or care about the actual logging implementation
2. **slf4j-simple-2.0.13.jar** (the implementation)
   * This is a simple logging implementation that outputs to the console
   * It "binds" to the SLF4J API at runtime
   * Provides the actual logging functionality

next() 🡪true if the new current row is valid; false if there are no more rows

Moves the cursor forward one row from its current position. A ResultSet cursor is initially positioned before the first row; the first call to the method next makes the first row the current row; the second call makes the second row the current row, and so on.

Driver -🡪 jdbc calls 🡪 mysql calls , convert datatypes

DriverManager 🡪 contains list of all driver based on the loaded driver it routes the connection request to appropriate driver.

Connection 🡪 acts as a session

Statement 🡪 executing the query and handling result.