JDBC

AND

ODBC

JDBC

JDBC, which stands for Java Database Connectivity, is an API (Application Programming Interface) that allows Java programs to interact with databases. It provides a set of classes and interfaces for connecting to a database, executing SQL queries, and processing the results. Here's an overview of JDBC and its key concepts:

1. **DriverManager**: JDBC programs typically start by loading and registering a JDBC driver using the **DriverManager** class. The driver is responsible for establishing a connection to the database.
2. **Connection**: Once a driver is registered, a connection to the database is established using the **Connection** interface. The connection represents a session with the database and is used to execute SQL statements and manage transactions.
3. **Statement**: SQL queries are executed using the **Statement** interface, which provides methods for executing SQL statements, such as **executeQuery()** for SELECT queries and **executeUpdate()** for INSERT, UPDATE, or DELETE queries.
4. **PreparedStatement**: **PreparedStatement** is a subinterface of **Statement** that represents a precompiled SQL statement. It allows parameterized queries, which help prevent SQL injection attacks and improve performance by reusing query execution plans.
5. **ResultSet**: When a SELECT query is executed, the result is returned as a **ResultSet** object, which represents a set of rows retrieved from the database. The **ResultSet** provides methods for navigating through the result set and retrieving data from individual rows.
6. **Transaction Management**: JDBC supports transaction management, allowing multiple SQL statements to be executed as part of a single transaction. The **Connection** interface provides methods for starting, committing, or rolling back transactions.
7. **Exception Handling**: JDBC methods can throw **SQLException** and its subclasses to indicate errors or exceptions encountered during database operations. Proper exception handling is essential to handle errors gracefully and ensure the reliability of JDBC programs.
8. **Connection Pooling**: In production environments, JDBC connection pooling is often used to improve performance and scalability by reusing and managing a pool of database connections. Connection pooling frameworks such as Apache Commons DBCP or HikariCP can be integrated with JDBC programs to manage database connections efficiently.

Overall, JDBC provides a powerful and flexible way to interact with databases from Java applications, making it possible to perform a wide range of database operations programmatically.

**Steps to Connect Java Application with Database**

**Below are the steps that explains how to connect to Database in Java:**

**Step 1** – Import the Packages  
**Step 2** – Load the drivers using the *forName() method*

Class.forName(“oracle.jdbc.driver.OracleDriver”);

DriverManager.registerDriver(neworacle.jdbc.driver.OracleDriver())

**Step 3** – Register the drivers *using DriverManager*

ConnectionDriverManager.getConnection(url,user,password)

**Step 4** – Establish a connection*using the Connection class object*

Statement st = con.createStatement();

**Step 5** – Create a statement  
**Step 6** – Execute the query  
**Step 7** – Close the connections

**ODFC**

ODBC stands for Open Database Connectivity. It is a standard API (Application Programming Interface) for accessing database management systems (DBMS). ODBC enables applications to interact with various database systems using a common set of function calls, regardless of the specific database's vendor or underlying architecture.

Here's an overview of ODBC and its key components:

1. **Driver Manager**: The Driver Manager is a component of the ODBC architecture responsible for managing ODBC drivers. It loads and unloads ODBC drivers, processes connection requests from applications, and routes function calls to the appropriate driver.
2. **ODBC Driver**: An ODBC driver is a software component that provides an interface between an application and a specific database management system (DBMS). Each database system typically has its own ODBC driver. The driver implements the ODBC API and communicates with the database server to execute SQL queries, retrieve data, and perform other database operations.
3. **Data Source Name (DSN)**: A Data Source Name is a user-friendly name that identifies a data source, such as a database or data file, to which an application wants to connect. DSNs are configured using the ODBC Administrator tool and can be either user DSNs (specific to a user) or system DSNs (available to all users on a system).
4. **ODBC API**: The ODBC API consists of a set of function calls that applications use to interact with ODBC drivers and data sources. These functions include connecting to a data source, executing SQL queries, retrieving result sets, fetching data, managing transactions, and error handling.
5. **Driver-Specific Settings**: ODBC drivers often support additional settings and capabilities specific to the underlying database system they connect to. These settings can be configured through connection strings or driver-specific options in the ODBC Administrator tool.

Overall, ODBC provides a standardized way for applications to access and manipulate data stored in various database systems. It abstracts the differences between different database systems, allowing developers to write database-agnostic applications that can easily switch between different data sources without needing extensive changes to the code.

DIFFERNCE BETWEEN JDBC AND ODBC

| **ODBC** | **JDBC** |
| --- | --- |
| 1. [ODBC](https://www.geeksforgeeks.org/connectivity-of-odbc/) Stands for Open Database Connectivity. | 1. [JDBC](https://www.geeksforgeeks.org/jdbc-tutorial/) Stands for Java database connectivity. |
| 2. Introduced by Microsoft in 1992. | 2. Introduced by SUN Micro Systems in 1997. |
| 3. We can use ODBC for any language like [C](https://www.geeksforgeeks.org/c-programming-language/), [C++](https://www.geeksforgeeks.org/c-plus-plus/), [Java](https://www.geeksforgeeks.org/java/) etc. | 3. We can use JDBC only for Java languages. |
| 4. We can choose ODBC only Windows platform. | 4. We can use JDBC on any platform. |
| 5. Mostly ODBC Driver is developed in native languages like C, and C++. | 5. JDBC Stands for Java database connectivity. |
| 6. For Java applications it is not recommended to use ODBC because performance will be down due to internal conversion and applications will become platform-dependent. | 6. For Java applications it is highly recommended to use JDBC because there are no performance & platform dependent problems. |
| 7. ODBC is procedural. | 7. JDBC is object-oriented. |

