1

Introducing JDBC

Java Database Connectivity (JDBC) is a Java standard that provides the interface for connecting from Java to relational databases. JDBC is based on the X/Open SQL Call Level Interface (CLI). JDBC 4.0 complies with the SQL 2003 standard.

The JDBC standard is defined and implemented through the standard `java.sql` interfaces. This enables individual providers to implement and extend the standard with their own JDBC drivers. This chapter provides an overview of the Oracle implementation of JDBC, covering the following topics:

1.1 Overview of Oracle JDBC Drivers

In addition to supporting the standard JDBC application programming interfaces (APIs), Oracle drivers have extensions to support Oracle-specific data types and to enhance performance.

Oracle provides the following JDBC drivers:

- Thin driver

  The JDBC Thin driver is a pure Java, Type IV driver that can be used in applications. It is platform-independent and does not require any additional Oracle software on the client-side. The JDBC Thin driver communicates with the server using Oracle Net Services to access Oracle Database.

  The JDBC Thin driver enables a direct connection to the database by providing an implementation of Oracle Net Services on top of Java sockets. The driver supports the TCP/IP protocol and requires a TNS listener on the TCP/IP sockets on the database server.

  > **Note:**
  >
  > Oracle recommends you to use the Thin driver unless you have a feature that is supported only by a specific driver.

- Oracle Call Interface (OCI) driver

  It is used on the client-side with an Oracle client installation. It can be used only with applications.

  The JDBC OCI driver is a Type II driver used with Java applications. It requires platform-specific OCI libraries. It supports all installed Oracle Net adapters, including interprocess

communication (IPC), named pipes, TCP/IP, and Internetwork Packet Exchange/ Sequenced Packet Exchange (IPX/SPX).

The JDBC OCI driver, written in a combination of Java and C, converts JDBC invocations to calls to OCI, using native methods to call C-entry points. These calls communicate with the database using Oracle Net Services.

The JDBC OCI driver uses the OCI libraries, C-entry points, Oracle Net, core libraries, and other necessary files on the client computer where it is installed.

OCI is an API that enables you to create applications that use the native procedures or function calls of a third-generation language to access Oracle Database and control all phases of the SQL statement processing.

- Server-side Thin driver

  It is functionally similar to the client-side Thin driver. However, it is used for code that runs on the database server and needs to access another session either on the same server or on a remote server on any tier.

  The JDBC server-side Thin driver offers the same functionality as the JDBC Thin driver that runs on the client-side. However, the JDBC server-side Thin driver runs inside Oracle Database and accesses a remote database or a different session on the same database for use with Java in the database.

  This driver is useful in the following scenarios:

  – Accessing a remote database server from an Oracle Database instance acting as a middle tier

  – Accessing an Oracle Database session from inside another, such as from a Java stored procedure

  The use of JDBC Thin driver from a client application or from inside a server does not affect the code.

- Server-side internal driver

  It is used for code that runs on the database server and accesses the same session. That is, the code runs and accesses data from a single Oracle session.

  The JDBC server-side internal driver supports any Java code that runs inside Oracle Database, such as in a Java stored procedure, and accesses the same database. It lets the Oracle Java Virtual Machine (Oracle JVM) to communicate directly with the SQL engine for use with Java in the database.

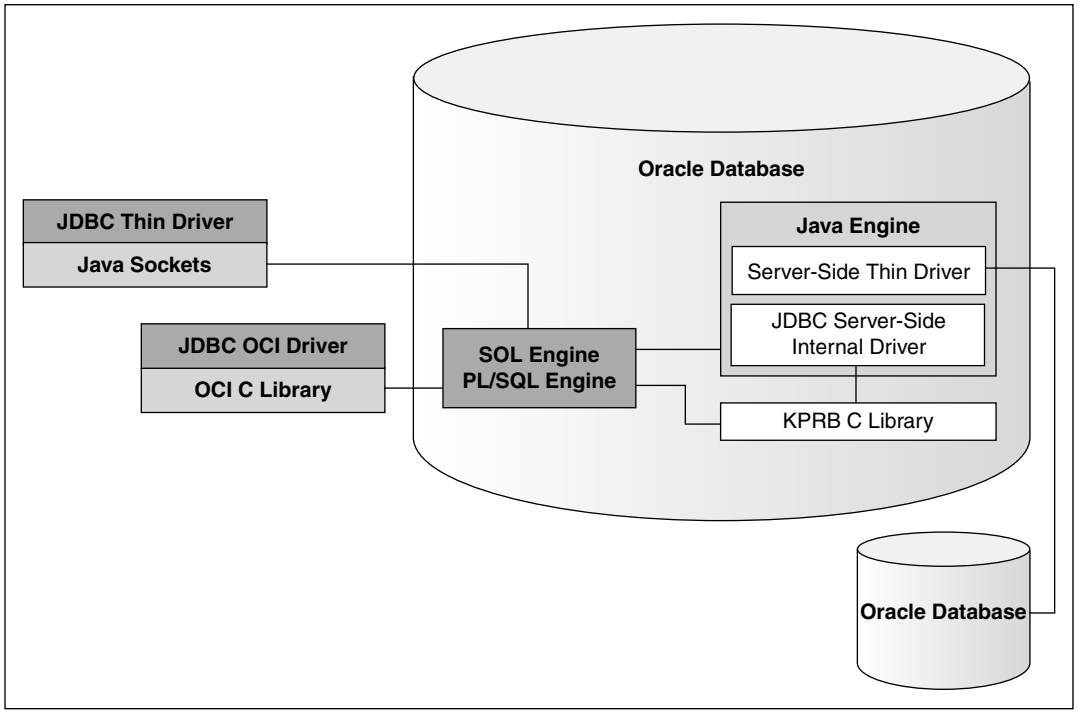
  The JDBC server-side internal driver, the Oracle JVM, the database, and the SQL engine all run within the same address space, and therefore, the issue of network round-trips is irrelevant. The programs access the SQL engine by using function calls.

  > **Note:**
  >
  > The server-side internal driver does not support the `cancel` and `setQueryTimeout` methods of the `Statement` class.

  The JDBC server-side internal driver is fully consistent with the client-side drivers and supports the same features and extensions.

The following figure illustrates the architecture of Oracle JDBC drivers and Oracle Database.

Figure 1-1 Architecture of Oracle JDBC Drivers and Oracle Database



Related Topics

- Features Specific to JDBC Thin
- Features Specific to JDBC OCI Driver
- Server-Side Internal Driver

1.2 Choosing the Appropriate Driver

Consider the following when choosing a JDBC driver for your application or applet:

- In general, unless you need OCI-specific features, such as support for non-TCP/IP networks, use the JDBC Thin driver.
- If you want maximum portability and performance, then use the JDBC Thin driver. You can connect to Oracle Database from an application using the JDBC Thin driver.
- If you want to use Lightweight Directory Access Protocol (LDAP) over Transport Layer Security (TLS), then use the JDBC Thin driver.
- If you are writing a client application for an Oracle client environment and need OCI-driver-specific features, such as support for non-TCP/IP networks, then use the JDBC OCI driver.
- For code that runs in the database server and needs to access a remote database or another session within the same database instance, use the JDBC server-side Thin driver.
- If your code runs inside the database server and needs to access data locally within the session, then use the JDBC server-side internal driver to access that server.

1.3 Use Cases of Oracle JDBC Drivers

This section describes the use cases for Oracle JDBC drivers.

JDBC Thin Driver or Type 4 Client Driver

You must use the JDBC Thin driver in your client-side Java applications for accessing Oracle Database over the TCP/IP protocol, for both `tcp` and `tcps`. This is the most widely used driver that Oracle recommends to use because it offers a range of features as mentioned in the following non-exhaustive list:

- Row count per iteration
- Support for promoting a local transaction to a global transaction
- Transaction Guard
- Transparent Application Continuity and Application Continuity
- Support for the Reactive Streams Ingestion library
- JDBC Reactive Extensions

> **See Also:**
>
> Feature List

JDBC OCI Driver or Type 2 Client Driver

You must use the JDBC OCI driver in your client-side Java applications only if your applications use any of the following features that are dependent on the platform-specific OCI libraries:

- Bequeath protocol
  This procol lets you use the local connections without going through the listener, which is typically used by the Database Administrators to perform various administrative operations; however, other non-administrative users too can use this protocol.

  > **See Also:**
  >
  > Using Bequeath Connection and SYS Logon

- OS Authentication
  The JDBC OCI driver supports OS Authentication on Linux when the client and the server are on the same computer. On Window domains, it supports OS Authentication even across multiple computers.
- Transparent Application Failover (TAF) that supports failover of read transactions.

  > **See Also:**
  >
  > Application Continuity for Java for information about complete high availability features

Server-Side Internal Driver or Type 2 Server-Side Driver

You must use this driver for Java code that runs on the JVM embedded in the Database server and accesses the same Database session. It means that the code runs and accesses data from a single Database session. This driver is also known as the JDBC KPRB Driver.

This driver is built in the JVM embedded in the Database, also known as Oracle JVM, and it is not a fully-featured driver.

Server-Side Thin Driver or Type 4 Server-Side Driver

You must use this driver in the following scenarios:

- For accessing a remote database server from an Oracle Database instance acting as a middle tier
- For accessing an Oracle Database session from within another session, such as from a Java stored procedure

This driver is built in the Oracle JVM, and it is not a fully-featured driver.

1.4 Feature Differences Between JDBC OCI and Thin Drivers

The following table lists the features that are specific either to the JDBC OCI or the JDBC Thin driver in Oracle Database Release 23ai.

Table 1-1 Feature Differences Between the JDBC OCI Driver and the JDBC Thin Driver

JDBC OCI Driver	JDBC Thin Driver
OCI connection pooling	NA
Transparent Application Failover (TAF)	NA
NA	Support for row count per iteration for array DML
NA	SHA-2 Support in Oracle Advanced Security
`oraaccess.xml` configuration file settings	NA
NA	Oracle Advanced Queuing
NA	Support for the `O7L_MR` client ability
NA	Support for promoting a local transaction to a global transaction
NA	Java Data Source for Sharded Databases Access
NA	Java Library for Reactive Streams Ingestion
NA	JDBC Reactive Extensions
NA	Native JSON Type Support

Note:

- The OCI optimized fetch feature is internal to the JDBC OCI driver and not applicable to the JDBC Thin driver.
- Some JDBC OCI driver features, inherited from the OCI library, are not available in the Thin JDBC driver.

1.5 Environments and Support

This section provides a brief description of the tools and environments that you need to run a JDBC application.

- Supported JDK and JDBC Versions
- JNI and Java Environments
- JDBC and IDEs
- Availability on Maven Central

1.5.1 Supported JDK and JDBC Versions

In Oracle Database 23ai, all the JDBC drivers are compatible with JDK 8, JDK 11, and JDK 17, and the `ojdbc8.jar`, `ojdbc11.jar`, and `ojdbc17.jar` files provide the support to these JDK versions.

When to Use ojdbc8.jar File

Use the `ojdbc8.jar` file when you want JDBC 4.2 features and need to compile your code with JDK 8 and JDK 11.

When to Use ojdbc11.jar File

Use the `ojdbc11.jar` file when you want JDBC 4.3 features and need to compile your code with JDK 11.

> **See Also:**
>
> JDBC FAQ Page

When to Use ojdbc17.jar File

Use the `ojdbc17.jar` file when you want JDBC 4.3 features and need to compile your code with JDK 17 and above. This JAR file is compatible with the Jakarta APIs.

Related Topics

- RDBMS and JDK Version Compatibility for Oracle JDBC Drivers
  Oracle Database Release 23ai JDBC drivers are certified with all the supported Oracle Database releases (23ai, 21c, and 19c).

1.5.2 JNI and Java Environments

The JDBC OCI driver uses the standard Java Native Interface (JNI) to call OCI C libraries. You can use the JDBC OCI driver with Java Virtual Machines (JVMs), in particular, with Microsoft and IBM JVMs.

1.5.3 JDBC and IDEs

The Oracle JDeveloper Suite provides developers with a single, integrated set of products to build, debug, and deploy component-based database applications for the Internet. The Oracle

JDeveloper environment contains integrated support for JDBC, including the JDBC Thin driver and the native OCI driver. The database component of Oracle JDeveloper uses the JDBC drivers to manage the connection between the application running on the client and the server.

1.5.4 Availability on Maven Central

All supported releases of the Oracle JDBC drivers, including 21.1.0.0, 19.9.0.0, 19.8.0.0, 19.6.0.0, 19.3.0.0, and 18.3.0.0, are available on Maven Central. So, you can consider Maven Central as a distribution center for the Oracle JDBC drivers and companion JAR files.

Group IDs for JDBC Drivers and Companion JAR Files on Maven Central

All Oracle Database artifacts on Maven Central reside under the same umbrella `com.oracle.database` as shown in the following image:

You can find the Oracle Database Artifacts under their specific focus area. For example, JDBC, XML, security, high-availability (HA), NLS, observability, and so on. The following table lists the group IDs of the JDBC drivers and the companion JAR files:

Group ID	Corresponding JAR Files
`com.oracle.database.jdbc`	`ojdbc11.jar`, `ojdbc10.jar`, `ojdbc8.jar`, `ojdbc6.jar`, `ojdbc5.jar`, `ucp.jar`, `ojdbc10dms.jar`, `ojdbc8dms.jar`, `ojdbc6dms.jar`, `ojdbcd5dms.jar`
`com.oracle.database.jdbc.debug`	`ojdbc11_g.jar`, `ojdbc10_g.jar`, `ojdbc8_g.jar`, `ojdbc6_g.jar`, `ojdbc5_g.jar`, `ojdbc11dms_g.jar`, `ojdbc10dms_g.jar`, `ojdbc8dms_g.jar`, `ojdbc6dms_g.jar`, `ojdbc5dms_g.jar`
`com.oracle.database.security`	`oraclepki.jar`
`com.oracle.database.ha`	`ons.jar`, `simplefan.jar`
`com.oracle.database.nls`	`orai18n.jar`

Group ID	Corresponding JAR Files
com.oracle.database.xml	xdb.jar, xdb6.jar, xmlparserv2.jar
com.oracle.database.observability	dms.jar

Note:

- The ojdbc8dms.jar and ojdbc11dms.jar files provide complete support for the Dynamic Monitoring System (DMS) and limited support for the java.util.logging package.
- xdb6.jar is a legacy name. The new name is xdb.jar.

Managing Dependencies on Maven Central with GAVs

You can manage the JDBC and UCP dependencies in the pom.xml file of your project by using the corresponding group ID, artifact ID, and the version (GAV), as defined in this section. For example, the following GAV pulls the ojdbc10.jar, ucp.jar, oraclepki.jar, ons.jar, and simplefan.jar from the 19.3 release:

```
<dependency>
      <groupId>com.oracle.database.jdbc</groupId>
      <artifactId>ojdbc10</artifactId>
      <version>19.3.0.0</version>
    </dependency>
```

Similarly, the following GAV pulls the orai18n.jar file from the 19.3.0.0 release:

```
<dependency>
      <groupId>com.oracle.database.nls</groupId>
      <artifactId>orai18n</artifactId>
      <version>19.3.0.0</version>
    </dependency>
```

See Also:

JDBC FAQ Page

1.6 Feature List

This section lists the supported features and the corresponding versions in which they were first supported in the JDBC OCI driver and the JDBC Thin driver.

Table 1-2 Feature List

Feature	JDBC OCI	JDBC Thin
TimeZone Patching	11.2	11.2

Table 1-2 (Cont.) Feature List

Feature	JDBC OCI	JDBC Thin
Secure LOB Support	11.2	11.2
LOB prefetch Support	11.2	11.2
Network Connection Pool	NA	11.2
Column Security Support	NA	11.2
XMLType Queue Support (AQ)	NA	11.2
Notification Grouping (AQ and DCN)	NA	11.2
SimpleFAN	11.2	11.2
Application Continuity	12.2	12.1
Transaction Guard	12.2	12.1
SQL Statement Translation	NA	12.1
Database Resident Connection Pooling	12.1	12.1
SHA-2 Support in Oracle Advanced Security	NA	12.1
Invisible Columns Support	12.1	12.1
Support for PL/SQL Package Types as Parameters	12.1	12.1
Support for Monitoring of Database Operations	12.1	12.1
Support for Increased Length Limit for Various Data Types	12.1	12.1
Implicit Results Support	12.1	12.1
Support for row count per iteration for array DML	NA	12.1
`oraaccess.xml` configuration file settings	12.1	NA
Transparent Application Continuity	NA	18c
Support for verifying JSON Data	18c	18c
Support for Lightweight Connection Validation	NA	18c
Support for REF CURSOR as IN bind variables	18c	18c
Support for Key Store Service	NA	18c
Easy Connect Plus (Easy Connect Naming Syntax Improvements)	NA	19c
Token-Based Authentication for IAM	NA	19c
Token-Based Authentication for Azure AD	NA	19c
Java Library for Reactive Streams Ingestion	NA	21c
Java Data Source for Sharded Databases Access	NA	21c
JDBC Support for Native JSON Data Type	NA	21c
JDBC Reactive Extensions	NA	21c
Java Virtual Threads	NA	21c
JDBC Pipelining Support	23ai	23ai
JDBC Service Provider Extensions	NA	23ai
Self-Driven Diagnosability	NA	23ai
Kerberos Constrained Delegation	NA	23ai
RADIUS Challenge-Response Authentication	NA	23ai
True Cache Data Source	23ai	23ai
Data Load Mode in RSI	NA	23ai
Bequeath Protocol	NA	23ai
Resumable Cursors	NA	23ai

Table 1-2 (Cont.) Feature List

Feature	JDBC OCI	JDBC Thin
Multiple Pool in DRCP	23ai	23ai

Note:

- The following features of JDBC drivers were introduced in releases earlier than release 11.2:
  - NLS Support
  - New Statement Caching API
  - Row Prefetch
  - Java Native Interface
  - Native LOB
  - Associative Arrays/index-by-table
  - Implicit Statement Caching
  - Explicit Statement Caching
  - Temporary LOBs
  - Object Type Inheritance
  - Multilevel Collections
  - `oracle.jdbc` Interfaces
  - Native XA
  - OCI Connection Pooling
  - Transparent Application Failover
  - Implicit Connection Cache
  - Fast Connection Failover
  - Connection Wrapping
  - DMS
  - Service Names in URLs
  - Set Statement Parameters by Name
  - End-to-End Tracing
  - Web `RowSet`
  - Proxy Authentication
  - Run-time Connection Load Balancing
  - Extended `setXXX` and `getXXX` methods for LOBs
  - XA Connection Cache
  - DML Returning
  - JSR 114 `RowSets`
  - SSL/TLS Encryption
  - SSL/TLS Authentication
  - AES Encryption
  - SHA1 Hash

  - Radius Authentication
  - Kerberos Authentication
  - `ANYDATA` and `ANYTYPE` types
  - Native AQ
  - Query Change Notification
  - Database start up and shut down
  - Factory methods for data types
  - Buffer Cache
  - Secure File LOBs
  - Diagnosability
  - Server Result Cache
  - Universal Connection Pool
- The `ConnectionCacheImpl` connection cache feature is deprecated since Oracle Database 10*g*.
- The Implicit Connection Cache feature is desupported now.