ORACLE

What's in Oracle Database 23ai for Java Developers?

April, 2024, Version 1.0

Public

ORACLE

Purpose statement

This document provides an overview of features and enhancements included in release 23ai. It is intended solely to help you assess the business benefits of upgrading to 23ai and planning for the implementation and upgrade of the product features described.

Disclaimer

This document in any form, software, or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement, nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described in this document remains at the sole discretion of Oracle. Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

ORACLE

Table of contents

ORACLE

Purpose

This document provides an overview of features and enhancements included in release 23ai. It is intended solely to help you assess the business benefits of upgrading to 23ai and planning for the implementation and upgrade of the product features described.

Introduction

I am a Java developer or a Java architect, why should I read this document? There are several reasons for you to read this document. This technical brief gives you a summary of Java features in Oracle Database 23ai. The new features address the areas of Java support for Oracle AI Vector Search, ease of development, Multi-Cloud, Java apps performance and scalability, mission critical deployments, security, and availability.

AI Vector Search and Vector Datatype

A vector is an array of one or more numeric values i.e., integers (..., -2, -1, 0, 1, 2, ...) or fractional numbers (..., -2.2, -1.1, 0.0, 1.1, 1.1, ...). Vector embeddings are the mathematical representations of the meanings and relationships of objects). Vector embeddings are generated by models.

AI Vector Search allows question answering and similarity search i.e., semantic relationship between objects (words, phrases, images, and so on).

The VECTOR data type has been introduced for storing and indexing vector embeddings for fast retrieval and similarity search. The possible numeric types of a VECTOR are: INT8, FLOAT32, and FLOAT64.

```
CREATE TABLE my_vectors (id NUMBER, embedding VECTOR(768, INT8)) ;
```

In this example, each vector has 768 dimensions, and each dimension is an INT8.

JDBC Support for Vector Data Type

JDBC Support for the Vector datatype allows developers build robust, scalable, and high-performance Java applications with Artificial Intelligence focus.

Values of the INT8 type are 8-bit two's complement numbers, corresponding to a "byte" in Java. Values of the FLOAT32 type are 32-bit floating point numbers, corresponding to a "float" in Java. Values of the FLOAT64 type are 64-bit floating point numbers, corresponding to a "double" in Java.

The JDBC API

The Oracle JDBC driver implement the necessary components to support AI Vector Search for Java applications including SQLType, DatabaseMetaData, ResultSetMetaData and ParameterMetaData, VectorMetaData, Java to SQL Conversions with PreparedStatement and CallableStatement, SQL to Java Conversions with CallableStatement, SQL to Java Conversions with CallableStatment and ResultSet, and VECTOR Datum class.

Using these APIs, you may implement an interactive command line program that performs a similarity search using a vector embedding of text sentences.

```
PreparedStatement query = connection.prepareStatement(
        "SELECT info"
          + " FROM my_data"
          + " ORDER BY VECTOR_DISTANCE(v, ?, COSINE)"
          + " FETCH APPROX FIRST ? ROWS ONLY");
```

See the Oracle JDBC doc for more details.

Ease of Development, Multi-Cloud

This release brings new features for ease of development and multi-Cloud.

Ease of Development

Features for ease of development include Oracle JVM support for Java SE 11 and Java Modules, JDBC-Thin support for Native Boolean type, JDBC-Thin support for Relational-JSON Duality Views, Oracle JVM Web Services Callout Enhancements, JDBC-Thin support for SQL Annotations, and Multi-Cloud support.

JDBC Support for ARRAY ENQUEUE/DEQUEUE for JSON Payload

In this release JDBC fixes a bug regarding the correctness of the array object for enqueue and dequeue calls with JSON payload.

Oracle JVM Goes JDK 11 with Java Modules

In this DB 23.4.0.24.05 release, the database resident JVM a.k.a. Oracle JVM now supports JDK 11 and Java modules.
Project Jigsaw led to the Java module system introduced in Java 9. The design goals were:

- Make it easier for developers to construct and maintain libraries and large applications
- Improve the security and maintainability of Java SE Platform Implementations in general, and the JDK in particular
- Enable improved application performance
- Enable the Java SE Platform, and the JDK, to scale down for use in small computing devices and dense cloud deployments.

Java modules support with Oracle JVM works as follows

1. The Java SE modules that make the Oracle JVM system, are automatically included in the module root set
2. If the `main` class of a Java class is a member of a module, then it is added to the module root set
3. Other modules can be added using one of the following options
    a. `loadjava --add-modules` option when the main class is loaded already. This is like the `JDK's java --add-modules` command-line argument
    b. Specifying the `oracle.aurora.addmods` system property. See setting property, here.
4. The final set of modules is examined for consistency and completeness at Oracle JVM session start up.

JDBC-Thin Support for Relational-JSON Duality Views

The Relational-JSON Duality Views are a new Oracle Database 23ai feature which gives developers the best of the Relational (the efficiency of space management, the queryability, consistency, the powerful analytics and reporting capabilities of SQL) and JSON (self-describing, self-contained, and schema-less, ease of development i.e., access JSON data programmatically, hierarchical data, common interchange format, binary JSON, and ease of conversion to Java types) worlds.

See my blog post for more details.

JDBC Support for Native Boolean Data Type

The Oracle JDBC supports the ISO SQL standard-compliant BOOLEAN data type in `oracle.jdbc.OracleType` with the following APIs

```
INSERT
...
String query = "INSERT INTO BoolTable values (?) ";
PreparedStatement pstmt = con.prepareStatement(query);
pstmt.setBoolean(1, true);
pstmt.execute();
...
```

```
FETCH
...
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select * from BoolTable");
while(rs.next()) {
  System.out.print("Value: " + rs.getBoolean("booleanColumn"));
...
}
```

`ResultSetMetaData` should return `getColumnType = 16` and `getColumnTypeName = BOOLEAN`

My colleague @juarezjunior explores extensively, in his blog post, the BOOLEAN type and compatibility with table columns that use NUMBER and VARCHAR data types as logical Boolean values.

JDBC Connection property sendBooleanAsNativeBoolean

A new property `sendBooleanAsNativeBoolean`, ensures backward compatibility with the older JDBC driver release.

When set to false (the default is true), this property will restore the old behaviour of sending integer values (0/1) as Boolean data type. This feature brings compatibility to Java applications that rely on the old behaviour of the Boolean data type thereby simplifying upgrading to the latest JDBC driver without breaking existing Java applications.

Oracle JVM Web Services Callout Enhancements

The ability to invoke or callout external REST or SOAP Web Services from with the database is a popular use case. In this release, the Oracle JVM Web Services Callout Utility embeds the wadl2java tool; no need for a separate download of that tool and no need for specifying its location using the `-t <wadl2java tool location>` option of the `loadjava` utility.

See more details here.

JDBC-Thin Support for SQL Annotations

A database table, view or column can be associated with an application metadata or annotation thereby allowing central management of changes. JDBC-Thin furnishes a new API for retrieving annotations.

```
getAnnotations(java.lang.String objectName, java.lang.String domainName,
java.lang.String domainOwner) throws java.sql.SQLException
```

```
getAnnotations(java.lang.String objectName, java.lang.String columnName,
java.lang.String domainName, java.lang.String domainOwner) throws java.sql.SQLException
```

Cloud Computing and Multi-Cloud

This section covers new features to support Cloud computing using Oracle Cloud Infrastructure. On the multi-Cloud front, Oracle and Microsoft have been working together to ensure Java apps can run across Azure (Apps) and Oracle Cloud (Autonomous Database) seamlessly. The new features include centralized AppConfig and resource providers. Some of these features can be used on-premises and in private Clouds.

AppConfig, Resource and Trace Events Providers

Cloud computing fosters the need for storing applications configurations and resources separately from application code in Cloud storages or Vaults and retrieve these securely. Centralized AppConfigs allows configuration changes at runtime, without making changes to your Java apps.

In this release, the Oracle JDBC supports centralized App config and resource providers through a new extensibility or plugin mechanism based on the standard Service Provider Interface.

Java developers can now store and retrieve their entire app configs and/or discrete data (e.g., database username, database password, database access token, TLS/SSL configuration) from Oracle Cloud Infrastructure Object Storage, Azure App config, JSON https server or from on-premises (e.g., file system). Sensitive data such as database connection passwords database access tokens, are stored as secrets in Vaults.
The centralization allows seamless config update without Java app code change.

https://github.com/oracle-samples/ojdbc-extensions

Centralized Configuration Providers

If the JDBC URL starts with *jdbc:oracle:thin:@**config-<provider>*** (e.g., *jdbc:oracle:thin:@config-azure*) then the driver will attempt to load the ***config-<provider.jar>*** (e.g., config-azure.jar) from the list of *registered service providers.*

The list of currently supported config providers includes the built-in providers, Azure App Config, OCI Object Storage, OCI Database Tools Connections and user-defined custom providers. See the JDBC providers Github repository.

Built-in providers: file system and https servers:

```
jdbc:oracle:thin:@config-file:config.json
```

```
jdbc:oracle:thin:@config-https://server/config/myappconfig?key=dev
```

1) Azure App Config connect_descriptor with optional user; password and wallet_location are optional as well but specified as secrets in Azure Key Vault.
   *URL= jdbc:oracle:thin:@config**-azure**:{appconfig-name}[?key=prefix&label=value&option1=value1…]*
   Example: `jdbc:oracle:thin:@config-azure:myappconfig?key=sales_app1&label=dev`
   As illustrated in figured 1, the combination of the app-config name (e.g., *myappconfig)*, the prefix (e.g., */sales_app1*) and the label (*e.g., dev*) are used for retrieving key-value pairs specified in the app config.
   Specific Oracle JDBC properties may also be specified and retrieved using <prefix>/jdbc.

Key	Value	Label
/sales_app1/user	scott	dev
/sales_app1/password	{"uri":"https://mykeyvault.vault.azure.net/secrets/passwordsalescrm"}	dev
/sales_app1/wallet_location	{"uri":"https://mykeyvault.vault.azure.net/secrets/walletcrm"}	dev
/sales_app1/connect_descriptor	(description=(retry_count=20)(retry_delay=3)(address=(protocol=tcps)(port=1521)(host=adb.us-phoenix-1.oraclecloud.com))(connect_data=(service_name=gebqqvpozhjbqbs_dbtest_medium.adb.oraclecloud.com))(security=(ssl_server_dn_match=yes)(ssl_server_cert_dn="CN=adwc.uscom-east-1.oraclecloud.com, OU=Oracle BMCS US, O=Oracle Corporation, L=Redwood City, ST=California, C=US")))	dev
/sales_app1/jdbc/autoCommit	false	dev

Figure 1 - App config

2) OCI Object Storage connect_descriptor with optional user; password as well as wallet_location are optional or specified as secrets in Azure Key Vault.
`URL = jdbc:oracle:thin:@config-ociobject:{object-url}[?key=name&option1=value1...]`
The connect_descriptor is stored under Object Storage / Buckets / Object → Object Details.
Example: `jdbc:oracle:thin:@config-ociobject:https://objectstorage.us-phoenix-1.oraclecloud.com/n/oracleonpremjava/b/bucket1/o/payload_ojdbc_objectstorage.json`

```
{"connect_descriptor": "(description=(retry_count=20)(retry_delay=3)(address=(protocol=tcps)(port=1521)(host=adb.us-phoenix-1.oraclecloud.com))(connect_data=(service_name=gebqqvpozhjbqbs_dbtest_medium.adb.oraclecloud.com))(security=(ssl_server_dn_match=yes)))",
  "user": "scott",
  "password": {
    "type": "vault-oci",
    "value": "ocid1.vaultsecret.oc1.phx.amaaaaaxxxx",
    "authentication": {
      "method": "OCI_INSTANCE_PRINCIPAL"
    }
  }
```

3) OCI Database Tools Connections with optional reference to OCI Vault for secrets.
Each configuration has an OCID that will be used to identify which connection is used. It contains a connectionString, userName, userPassword, keyStores and advancedProperties.

URL = jdbc:oracle:thin:@config-ocidbtools:ocid1.databasetoolsconnection.oc1.phx.ama ...

4) Custom providers: Java developers may build their own providers by implementing the *`oracle.jdbc.spi.OracleConfigurationProvider`* interface. That interface is in the JDBC Driver's jar; the built-in providers implement it. These providers must define their names and return *`java.util.Properties`*.

The Java app must

- Include the provider JAR file in the classpath or the provider reference in the POM file
- Set the required values in the connection URL.

Resource Providers

A Resource Provider furnishes the Oracle JDBC with a single resource such as database connection string, database username, database password, database access token, TLS/SSL configuration and Trace event listener. Resource providers are configured by connection properties that identify the name of the resource providers including: `oracle.jdbc.provider.connectionString, oracle.jdbc.provider.username, oracle.jdbc.provider.password, oracle.jdbc.provider.accessToken, oracle.jdbc.provider.tlsConfiguration, and oracle.jdbc.provider.traceEventListener.`

In the following code snippet, a connection property is used for configuring a password provider:

```
oracle.jdbc.provider.password=password-provider

oracle.jdbc.provider.password.vaultId=9999-8888-7777
```

The "`oracle.jdbcprovider.password`" property configures the name of a password provider while the "`oracle.jdbc.provider.password.vaultId`" property configures a vaultId which is recognized by the password provider.

Trace Event Listener Provider - OpenTelemetry

The Oracle JDBC driver may generate events such as database roundtrips during query execution, IP address retries while establishing a connection, the beginning of Application Continuity (AC) recovery from a database outage, a successful AC recovery.

The JDBC driver defines a listener that receives application and system tracing events from the Oracle Database JDBC drivers. See the `oracle.jdbc.TraceEventListener` javadoc for more details.

The JDBC driver also defines an `oracle.jdbc.spi.TraceEventListenerProvider` Service Provider interface that can be used to register a listener for publishing those events to OpenTelemetry or registering a custom `TraceEventListener`.

1. You can Implement programmatically using OracleConnectionBuilder.traceEventListener(TraceEventListener)
2. Or use an OracleResourceProvider implementation: either your own one or the Oracle's open-source provider for OpenTelemetry. The new TraceEventListener OpenTelemetry Provider is an implementation of the TraceEventListenerProvider interface; it has been published @ https://github.com/oracle-samples/ojdbc-extensions/tree/main/ojdbc-provider-opentelemetry.
3. Identify the Trace Event Listener Provider with oracle.jdbc.provider.traceEventListener connection property

```
oracle.jdbc.provider.traceEventListener=example-provider
oracle.jdbc.provider.traceEventListener.traceLevel=INFO
```

Here is the SpringBoot example

```
spring.datasource.url=jdbc:oracle:thin:@tcps://adb.us-phoenix-1.oraclecloud.com:1521/xyz.adb.oraclecloud.com?oracle.jdbc.provider.traceEventListener=open-telemetry-trace-event-listener-provider
```

The other JDBC providers are @ https://github.com/oracle-samples/ojdbc-extensions.

Expect more built-in and open-source providers in future releases.

Java Performance and Scalability

The new features for performance and scalability include: the TrueCache data source, JDBC support for pipelined database operations, a new DataLoad mode for the Reactive Streams Ingestion library (RSI), multi-pool support in DRCP, UCP support for Sharding Split Partition Set , UCP support for XA Transactions with Sharding, UCP support for Connection Creation Consumer and Callback, UCP support for Maximum Connection Reuse Time, Reactive

Extension to UCP, JDBC-Thin support for Bequeath protocol, performance enhancement for executeBatch()/andexecuteLargeBatch(), and enhanced UCP connection borrow.

TrueCache DataSource

A True Cache instance is an in-memory, mostly diskless, fully functional, read-only replica of the primary Oracle database. It resides in the middle-tier, collocated with the application. It is enabled at JDBC level by setting the new `oracle.jdbc.useTrueCacheDriverConnection` property to `true`. Once enabled, the True Cache datasource creates a logical connection which can be used either against the True Cache database instance or against the primary database.

For marking a connection as Read-Only, the True Cache data source uses the standard `java.sql.Connection.setReadOnly(boolean)` and `java.sql.Connection.isReadOnly()` methods. By default, the read-only mode for a connection is set to false.

The following example illustrates using TrueCache

https://gist.github.com/Kuassim/350e775c6dcf7b4448418c920b4f9425

Pipelined Database Operations

A database pipeline consists of a sequence of multiple database requests submitted without waiting for a response between operations. The database sends a response when the results of each query are ready. Pipelined database operations foster an asynchronous programming model, in which a user thread returns immediately upon submitting a SQL statement for execution without waiting for its execution and the ResultSets.

Java developers would leverage database pipelining through the Oracle JDBC Reactive Extensions, the Reactive Streams libraries (R2DBC, Reactor, RxJava, Akka Streams, Vert/x, etc), Java Virtual Threads.
See my blog post for code samples illustrating each of these use cases and the transparent pipelined operations support for the standard JDBC batching.

New DataLoad Mode for the Reactive Streams Ingestion

The Reactive Streams Ingestion library (RSI) allows fast data ingest into the Oracle database, using the direct path load and Reactive Streams mechanisms. It leverages the Java connection pool (UCP) as well as table partitions, Oracle RAC connection affinity and the Oracle Globally Distributed Database (formerly Database Sharding).

In this release, a new DataLoad mode has been added to the default Streaming mode. In the Streaming mode, the worker threads share a pool of JDBC connections, and the ingested data is committed on a frequent basis whereas in the DataLoad mode, the number of connections could be large and the ingested data are committed only when the RSI instance is closed.

```
ReactiveStreamsIngestion.Builder rsiBuilder = ReactiveStreamsIngestion.builder()
    .useDataLoadMode()
    .username("<user_name>")
    .password("<password>")
    .url("jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)
(HOST=myhost.com)(PORT=5521))(CONNECT_DATA=(SERVICE_NAME=myservice.com)))")
    .table("customers")
    .columns (new String[] { "id", "name", "region" });
// Use try-with-resource statement to ensure that RSI instance is closed at the
// end of the statement.
try (ReactiveStreamsIngestion rsi = rsiBuilder.build()){
   // Publish Records.
}
```

See more details and code samples in the Oracle Database 23ai JDBC Dev guide.

UCP Support for Split Partition Set

There is no public API for this new feature; it operates under the covers. During the partition-set split, the client-side connection pools (e.g., UCP) receive ONS events about data in a chunk being split and moved across partition sets; the connection pools update their Sharding topology appropriately.

UCP Support for XA Transactions with Database Sharding

Java applications that use the UCP native data source in WebLogic Server to connect to Sharded Oracle databases can participate in JTA/XA transactions managed by the WebLogic Transaction Manager (TM).

UCP Support for Connection Creation Consumer and Callback

This feature allows a Java application to register "*a connection creation consumer*" for a specific `PoolDataSource` object. That consumer will be notified i.e., called back, upon explicit (i.e., by the Java application) or implicit (i.e., by UCP to adjust sizing settings) connection creation.

See chapter 4 of the UCP Dev guide for more details and code samples for registering a connection creation consumer, unregistering it, checking its status, and so on.

UCP Support for Maximum Connection Reuse Time

The typical use case for this feature is when the middle-tier and the database tier are separated by a firewall; in that case, some connections may be blocked by the firewall, and remain idle in the pool for a long time.
Setting the maximum connection reuse time (in seconds) to a smaller value than the firewall timeout will avoid such situation

```
pds.setMaxConnectionReuseTime(300);
```

Setting the new system property `oracle.ucp.timersAffectAllConnections` to `TRUE`, allows the periodic poll, to check all available connection for the maximum connection reuse time.

See more details in section 5.4.1.1 of the UCP Dev guide.

Multi-Pool Support in DRCP

The Database Resident Connection Pool (DRCP) is an RDBMS-side pool either at a Pluggable Database (PDB) level or at the managing infrastructure also known as Container Database (CDB) level. Java applications can refer to DRCP, using `(SERVER=POOLED)` in the JDBC connect string.

The new multi-pool feature allows sub-partitioning the DRCP between several applications by naming the sub-partitions using `(POOL_NAME= <pool_name>)` in the connect string.

Here is an example of connect string with DRCP and multi-pool.

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=)(PORT=))
             (CONNECT_DATA=(SERVER=POOLED)(POOL_NAME=)))
```

You can add or remove a pool to/from the multi-pool using the `add_pool ()` and `remove_pool()` procedures of the `dbms_connection_pool` package.

```
exec dbms_connection_pool.add_pool('mypool')

exec dbms_connection_pool.remove_pool('mypool')
```

Reactive Extension to UCP

The previous Oracle Database 21c release introduced the reactive extensions to Oracle JDBC, an implementation of the Java util concurrent Flow interface. In this release the Java connection pool a.k.a. UCP has been extended with Reactive extension allowing to issue asynchronous connection borrowing requests.
It works as follows:

1. Instantiate either a `UCPConnectionBuilder` or an `UCPXAConnectionBuilder`
2. Request a vanilla connection asynchronously with the `UCPConnectionBuilder` using either a `CompletableFuture<Connection>` or a `Publisher<Connection>`. Alternatively, you may request an XA connection asynchronously with the `UCPXAConnectionBuilder` using either a `CompletableFuture<XAConnection>` or a `Publisher<XAConnection>`.
3. The asynchronous borrow operation runs with the default `ForkJoinPool` executor if you have not implemented the `java.util.concurrent.Executor` interface in your Java code.
4. The CompletableFuture or the Publisher are notified when the borrow operation is complete.
5. You can then perform operations on the connections object.

See code samples in chapter 11 of the UCP Developer's Guide.

JDBC-Thin Support for Bequeath Protocol

This protocol allows the database client (e.g., Java/JDBC application) and the database server process residing on the same Linux host, to communicate directly without the network layer and the network listener. Using that protocol to connect to an Oracle database requires setting the values of ORACLE_HOME and ORACLE_SID variables either in the connection URL (as shown hereafter) or as application environment variable.

```
jdbc:oracle:thin:@(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq))(ENVS=ORACLE_HOME=/
var/lib/oracle/dbhome,ORACLE_SID=oraclesid))
```

Enhancement to executeBatch() executeLargeBatch()

In this release, the response time of PreparedStatement.executeBatch() and PreparedStatement.executeLargeBatch() has been significantly improved through a single round trip for batches larger than 2MB. In the previous releases, the JDBC driver was making multiple database calls for each batch DML operation.

Enhanced UCP Connection Borrow

Connection borrow may take longer than the *connectionWaitTimeout* specified value, if UCP must create a new connection to fulfill the request. In this release, if a connection has been released by another thread in the meantime, UCP allocates the just-released connection rather than waiting for the one being created.

Mission Critical Deployment, Security and Availability

This section covers new features to support mission critical deployment of Java apps, security and availability (zero-downtown).

Mission Critical Deployment

This section gives a summary of the new App Stack for Java, and the new Observability features.

IaC - App Stack for Java

When deploying Java apps in the Cloud developers face several pain points including configuring the virtual network, provisioning of the deployment platform, as well as the database, configuring the JDBC datasource, as well as the load balancer and DNS, generating the build and deployment of CI/CD pipelines. An infrastructure as Code (IaC) framework (the App Stack for Java) allows automating most tasks involved with provisioning Cloud infrastructure resources thereby eliminating those pains.

The App Stack for Java is not technically part of the Oracle DB23ai release; I covered it in this blog post; please check it out.

Observability

Observability refers to monitoring, capturing, and dynamically analyzing the logs, metrics and traces of your applications and diagnosing issues in real-time. It is a crucial requirement for modern, service-based applications development, deployment, DevOps, and so on. This release brings enhanced logging, new debugging (diagnose on first failure), and new tracing capabilities.

Single Jar for all use cases

Let me start with the good news "*No more switching between the production jar and the debug jar to investigate an issue*"!

A single ojdbc jar (e.g., *ojdbc8.jar, ojdbc11.jar*) for all use cases (production, debug, metrics). In other words, no more *ojdbc8_g.jar or ojdbc11_g.jar* for debugging, no more *ojdbc8dms.jar or ojdbc11dms.jar* for the Oracle Dynamic Monitoring Service (DMS) metrics, and no more *ojdbc8dms_g.jar or ojdbc11dms_g.jar* for DMS debugging.

Diagnose on First Failure (Self-Driven Diagnosability)

This feature diagnoses the first occurrence of a failure in your Java app. It records the critical execution state in memory, then dumps the recording on error. It is always ON (by default) but may be disabled via `-Doracle.jdbc.diagnostic.enableDiagnoseFirstFailure=false` or using the DiagnosticMBeans interface. You must configure `java.util.logging` to get diagnostic output on diagnose-on-first-failure.

See more details, specifically the handling of sensitive data, in my blog post.

Logging

The core JDBC Jars (i.e., *ojdbc8.jar* or *ojdbc11.jar*) include the logging capabilities which need to be turned on using the following properties:

```
-Doracle.jdbc.diagnostic.enableLogging=true.
-Djava.util.logging.config.file=./logging.config
```

The handler as well as the granularity of the logging must also be specified.

Distributed Tracing - OpenTelemetry

See **Trace Event Listener Provider – OpenTelemetry** on page 9.

DMS Metrics

The Oracle Dynamic Monitoring Service (DMS) metrics are recorded if the *dms.jar* is in the classpath.

Security

The new security features for Java include:

Thumbprint based certificate selection, EZConnect support for LDAP/LDAPS, JDBC support for TLS with DRCP, OJVM support for FIPS; support for longer passwords; token-based authentication for Oracle Cloud Infrastructure (OCI) Identity and Access Management (IAM) using OCI IAM token, and Azure Active Directory (AD) using OAuth 2.0 access tokens; RADIUS Challenge-Response Authentication (a.k.a. 2FA); enhancements for Kerberos support; and Oracle JVM Support for HTTP and TCP.

Thumbprint-based Certificate Selection

JDBC Thin driver performs certificate selection based on alias when multiple certificates are present in the KeyStore. The selected certificate and its thumbprint are used for client authentication through a TLS handshake.

EZConnect+ support for LDAP/LDAPS

The syntax follows a combination of the LDAP URL syntax and the Easy Connect Plus syntax:

ldap[s]://host[:port]/name[,context]?[parameter=value{&parameter=value}]

Example:

sqlplus
"<user_name>/<password>@ldaps://<host_name>/test?DIRECTORY_SERVER_TYPE=oid&WALLET_LOCATION=/oracle/network/admin&AUTHENTICATE_BIND=true&AUTHENTICATE_BIND_METHOD=LDAPS_SIMPLE_AUTH"

See the JDBC doc for more details.

OJVM Support for FIPS

The database resident JVM a.k.a. OJVM now allows installing FIPS 140-2 Java classes thereby making JsafeJCE the default cryptography provider. The steps for installing the Java classes and enabling FIPS are described in the OJVM doc FIPS section.

Support for Longer Passwords

JDBC now supports long passwords up to 1024 bytes, transparently, i.e., no API change.

Token-Based Authentication for OCI IAM and Azure AD

The Oracle JDBC support for Cloud Directory services including Oracle Cloud Infrastructure (OCI) Identity and Access Management (IAM) and Azure Active Directory (AD).

Token-Based Authentication for OCI IAM

The JDBC-Thin driver furnishes enhanced support for Oracle Cloud Infrastructure (OCI) Identity and Access Management (IAM).
The steps are as follows

1) The Java app supplies a database token using any of the methods described hereafter
2) The database verifies the token with a public key obtained from the authentication service and completes the user authorization.
3) The Java app sends a header proving it possesses a private key that is paired with a public key embedded in the token.
4) If both the token and the signature are valid, and there exists a mapping between the IAM user and a database user, then access to the database is granted to the JDBC application.

This authentication method can be used: for database tokens on file-system; using the `oracle.jdbc.accessToken` Connection Property to pass the token value; using the `OracleConnectionBuilder.accesToken method` to obtain the token from the authentication service; and using the supplier function of the `OracleDataSource` class `OracleCommonDataSource.setTokenSupplier(AccessToken accessToken.`

See the Client-side security chapter of the Oracle JDBC doc for more details.

Token-based Authentication for Azure AD

The JDBC-Thin driver supports Azure Active Directory OAuth2 access tokens.

The steps are as follows

1. The Java app supplies a database token using any of the methods described hereafter
2. The database verifies the token with a public key obtained from the authentication service and completes the user authorization.
3. This authentication method can be used: for database tokens on file-system; using the `oracle.jdbc.accessToken` Connection Property to pass the token value; using the

`OracleConnectionBuilder.accesToken method` to obtain the token from the authentication service; and using the supplier function of the `OracleDataSource` class `OracleCommonDataSource.setTokenSupplier(AccessToken accessToken.`

See the Client-side security chapter of the Oracle JDBC doc for more details.

RADIUS Challenge-Response Authentication (a.k.a. 2FA)

1) The Java app performs the first level of authentication using the user name and password
2) The RADIUS server sends a challenge to the Java app.
3) The Java app responds to the challenge using a handler.

The handler is configured using either the `oracle.net.radius_challenge_response_handler` connection property or the `ConnectionBuilder.radiusChallengeResponseHandler` method.

See more details including code samples in section 9.9.4 of the Oracle JDBC doc.

Kerberos Enhancements

Kerberos authentication has been simplified in this release by removing the requirement for a Ticket Grant Ticket in the `CredentialCache` or instantiating the `KerberosLoginModule`.

Configuring the Kerberos Principal and Password Properties

For connecting to Oracle Database using Kerberos Principal

Set the `PASSWORD_AUTH` parameter to `KERBEROS5` in the connection string. You can also set `PASSWORD_AUTH` to `KERBEROS5` using the `oracle.jdbc.passwordAuthentication` connection property. The JDBC Thin driver initializes the `KerberosLoginModule` for your application thereby simplifying Kerberos Authentication.

See the following code example

https://gist.github.com/Kuassim/3a628317a4501a0004b5e21fde829696

Kerberos Authentication Using the JAAS Configuration

By default, the JDBC-Thin driver uses the default Kerberos login module bundled with the Oracle JDK (`com.sun.security.auth.module.Krb5LoginModule`) however, you can choose to use the JAAS configuration instead.

See the following code example.

https://gist.github.com/Kuassim/fe6774588556f1f443334de57b408a99

Oracle JVM Support for HTTP and TCP

The database embedded JVM (a.k.a. OJVM) now supports enabling or disabling HTTP and TCP operations while disabling other OS call.

See more details in Database Security in a Multitenant Environment.

Availability

With Transparent Application Continuity (TAC), we are making high-availability the most transparent possible for Java Applications.

Transparent Application Continuity Enhancement

Application Continuity (AC) and Transparent Application Continuity (TAC) are high availability, and zero-downtime features of the Oracle database that hide database instance or network failures from Java applications. These features straddle the RDBMS server, the JDBC driver and UCP. Through releases, AC & TAC push most of the settings from the database clients and applications to the RDBMS server.
In this release, the JDBC drivers support *session stable cursors*; these are long-running cursors of a session, that stay open beyond transactions. With session stable cursors, TAC establishes *application request* boundaries, implicitly and more often, thereby ensuring broader TAC coverage.

Please read the entire chapter on Application Continuity for Java to get the whole picture and more details.

Conclusion

You got it. Through this technical brief, you got a complete summary of the new Java features in the Oracle Database 23ai (23.4.0.24.05) in the areas of ease of development, Cloud computing, multi-Cloud, performance and scalability, mission critical deployment, security, and availability. These features will undoubtedly help you design and deploy modern Java applications.

```
@kmensah, #javaOracleDB, http://oracle.com/jdbc
```

ORACLE

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle