38 Sessionless Transactions

Sessionless Transactions are the transaction that provides you with the flexibility to suspend and resume the transaction during its life cycle.

When you use Sessionless Transactions, you do not need to use a transaction manager when communicating with the Oracle Database (single or multi-instance). The database does all the work of coordinating the transaction for you. Moreover, the database internally coordinates the two-phase commit (2PC) protocol, without the need for any application-side logic.

Note:

This feature is supported only from Oracle Database Release 23ai, version 23.6.

38.1 About Sessionless Transactions

A sessionless transaction breaks the coupling between the transaction and the session. Once you start a transaction, it does not need to be tied to a session or connection.

You can free the session or connection, allowing it to be used by another client. Additionally, you do not need a transaction manager to coordinate the XA protocol because the 2-phase commit process and the error recovery process are handled automatically by the database. Therefore, there is no risk of in-doubt transactions and no need for any recovery mechanism. For more information about Sessionless Transactions, its use cases, and benefits, see Developing Applications with Sessionless Transactions in the *Database Development Guide*.

Note:

Sessionless Transactions are applicable only to a single database. Do not use Sessionless Transactions to manage transactions that span across multiple resource managers.

38.2 Manage Sessionless Transactions using JDBC APIs

Use the JDBC APIs described in this section to start a new transaction, resume an existing transaction, suspend a transaction, and either commit or roll back the changes to end a transaction.

These APIs are available in the `OracleConnection` interface of the `oracle.jdbc` package. See Oracle® Database JDBC Java API Reference.

The section covers the following topics:

38.2.1 Start Sessionless Transactions

Use the `startTransaction()` method from the `OracleConnection` interface of the `oracle.jdbc` package to start a sessionless transaction. You may specify a unique Global Transaction ID (GTRID) and timeout for the transaction.

This is an asynchronous call as `startTransaction()` does not perform a round-trip to the database. Instead, Oracle JDBC Thin driver sends a request to start a sessionless transaction when it makes the next round trip to the Oracle Database. The transaction starts only when the server successfully returns the call.

To start a sessionless transaction, complete the following tasks:

1. (Optional) Provide a unique identifier called global transaction ID (GTRID) for every sessionless transaction. Specify the unique value as a byte array which has the size between 1 to 64 bytes.

   If you do not specify a value, the Oracle JDBC Thin driver generates a unique value as the GTRID. You will use the GTRID to manage a transaction from start to end.

2. (Optional) Decide the `timeout`, which specifies the maximum duration for which a sessionless transaction can stay in the `suspended` state. If a suspended sessionless transaction is not resumed within the specified duration, Oracle Database cancels the transaction.

   Consider the following points while specifying the timeout.

   - If you specify a very high value, the transaction may hold database resources, such as locked rows, for the specified duration.
   - If you specify a very low value, for example 0, the transaction is canceled as soon as a sessionless transaction is suspended and released.
   - The default value is 60 seconds.

3. Call the `startTransaction()` method of the `OracleConnection` interface in one of the following ways.

   - This API starts a sessionless transaction with the default timeout of 60 seconds. The driver creates a unique GTRID for the transaction.

     ```
     public byte[] startTransaction() throws SQLException;
     ```

   - This API starts a sessionless transaction with the specified timeout. The driver creates a unique GTRID for the transaction.

     ```
     public byte[] startTransaction(int timeout) throws SQLException;
     ```

   - This API starts a sessionless transaction with the GTRID that you specify and the default timeout of 60 seconds.

     ```
     public void startTransaction(byte[] GTRID) throws SQLException;
     ```

   - This API starts a sessionless transaction with the GTRID and timeout that you specify.

     ```
     public void startTransaction(byte[] GTRID, int timeout) throws
     SQLException;
     ```

When a new sessionless transaction starts, it is in the active state. The GTRID for a transaction must be unique. If an active sessionless transaction already exists when you make a request to start a new sessionless transaction, the Oracle Database suspends the existing transaction and starts a new sessionless transaction based on your request.

38.2.2 Retrieve GTRID

Retrieves the GTRID of a sessionless transaction that is currently in the active state.

Sessionless Transactions are identified by a unique identifier called global transaction ID (GTRID). GTRID is used to manage a transaction from start to end.

- The following API retrieves the GTRID of a sessionless transaction that is currently in the active state.

  ```
  byte[] getTransactionId() throwsSQLException;
  ```

  It returns null, if there is no sessionless transaction in the active state.

Example

The retrieved GTRID reflects the state of the sessionless transaction locally and not its state in the server. Let's understand this with the following scenario, where `startTransaction()` is an asynchronous call and it does not perform a round-trip to the database. Instead, Oracle JDBC Thin driver sends the request to start a sessionless transaction when it makes the next round trip to the Oracle Database. The transaction starts only when the server successfully returns the call.

```
...
conn.startTransaction();
byte[] gtrid = getTransactionId();
...
```

Even though the transaction has not actually started on the server until the next round-trip, `getTransactionId()` returns the value of the GTRID generated by the Oracle JDBC Thin drivers.

38.2.3 Suspend Sessionless Transactions

Suspends an active sessionless transaction.

- Call a method to suspend a transaction in one of the following ways.
  - Use the following API to synchronously suspend an active sessionless transaction. This makes a round trip to the Oracle Database and immediately suspends a sessionless transaction.

    ```
    void suspendTransactionImmediately() throws SQLException;
    ```

  - Use the following API to asynchronously suspend an active sessionless transaction. This is an asynchronous call as `suspendTransaction()` does not perform a round-trip to the database. Instead, Oracle JDBC Thin driver sends a request to suspend a sessionless transaction when it makes the next round trip to the Oracle Database. The transaction is suspended only when the server successfully returns the call.

```
void suspendTransaction() throws SQLException;
```

- Use the following API to suspend an active sessionless transaction when you know about the last operation that is performed on the database before suspending the transaction. This is useful because it allows you to reduce the number of roundtrips to the server which provides better performance. However, note that if either `executeUpdate()` or `suspend()` operations throw an exception, both operations fail.

```
void executeUpdateAndSuspend() throwsSQLException;
```

If there are no sessionless transactions in the active state, then no operation is performed as there are no active transactions to suspend. This request is ignored, and no error message is returned.

38.2.4 Resume Sessionless Transactions

Resumes a suspended sessionless transaction.

If a sessionless transaction is in the suspended state when you send the request to resume a transaction, it resumes the transaction. The sessionless transaction is resumed on the instance where you have requested the transaction to be resumed. After a sessionless transaction is resumed, it is in the active state.

When you send a request to resume a transaction in a session, if a different sessionless transaction is already active in that session, Oracle Database first attempts to suspend that transaction and then proceeds to perform the resume operation.

This is an asynchronous call as `resumeTransaction()` does not perform a round-trip to the database. Instead, Oracle JDBC Thin driver sends a request to resume a sessionless transaction when it makes the next round trip to the Oracle Database. The transaction resumes only when the server successfully returns the call.

To resume a suspended sessionless transaction:

1. Identify the GTRID of the sessionless transaction that you want to resume.
2. (Optional) Decide the `timeout`, which specifies the maximum duration for which a sessionless transaction can stay in the `suspended` state. If a suspended sessionless transaction is not resumed within the specified duration, Oracle Database cancels the transaction.

   Consider the following points while specifying the timeout.

   - If you specify a very high value, the transaction may hold database resources, such as locked rows, for the specified duration.
   - If you specify a very low value, for example 0, the transaction is canceled as soon as a sessionless transaction is suspended and released.
   - The default value is 60 seconds.
3. Call the `resumeTransaction()` method, which is available in the `OracleConnection` interface, in one of the following ways. You must specify the GTRID of the transaction that you want to resume.

- Use the following API to resume a sessionless transaction while using the default timeout of 60 seconds.

  ```
  void resumeTransaction(byte[] GTRID) throws SQLException;
  ```

- Use the following API to resume a sessionless transaction while using the specified timeout.

  ```
  void resumeTransaction(byte[] GTRID, int timeout) throws SQLException;
  ```

38.2.5 Example

This section provides an example to modify an existing application code to use the sessionless transaction feature.

Let's consider that an application wants to avoid locking sessions while working on a transaction that starts, makes some updates, and then makes multiple calls to other services. After it gets the results from other services, it continues working on the transaction. Example code is provided below for your reference to compare the changes required when you use sessionless transaction.

Existing Sample Code Which Does Not Use Sessionless Transactions

The following code provides a sample implementation of such an application that does not use Sessionless Transactions.

```
PoolDataSource ds;

Connection conn = ds.getConnection();
conn.setAutoCommit(false);
// update database

// Fetch some data from other services(takes a while)

// do more updates on the database

conn.commit();
conn.close();
```

Optimized Sample Code to Use Sessionless Transactions

The following sample code provides the optimized sample code to use Sessionless Transactions. To manage Sessionless Transactions using JDBC APIs, which are available in the `OracleConnection` interface of the `oracle.jdbc` package, you must type cast the connection object to `OracleConnection` as shown in the following sample code.

```
import oracle.jdbc.*;

PoolDataSource ds;
// GTRID is the unique identifier for a session.
byte[] gtrid;

// Acquire a session on instance 1.
Connection conn = ds.getConnection();
```

```
conn.setAutoCommit(false);
// Start a sessionless transaction with a unique identifier on instance 1
gtrid = (OracleConnection(conn)).startTransaction();
// Update database
(OracleConnection(conn)).conn.suspend();
conn.close();

// Fetch some data from other services, which takes a while

Connection conn2 = ds.getConnection();
conn2.setAutoCommit(false);
// Resume the sessionless transaction using the unique identifier,
// that you had defined earlier, on instance 2.
(OracleConnection(conn2)).resumeTransaction(gtrid);
// do more updates on the database
conn2.commit();
conn2.close();
```

38.3 Manage Sessionless Transactions with Existing XA APIs

This section provides information about how you can manage Sessionless Transactions using the existing `XAResource` interface.

Sessionless Transactions are compatible with the APIs used for XA transactions. Set the `oracle.jdbc.XAThroughSessionlessTransactions` system property to `true` when you want the Oracle JDBC Thin driver to process sessionless transactions using XA APIs. The driver converts the XID provided by the transaction manager to GTRID, a unique identifier for a sessionless transaction. By default, `oracle.jdbc.XAThroughSessionlessTransactions` is set to `false`.

Limitations:

- You can use sessionless transaction only when your application connects to a single Oracle Database running on one server or in a RAC.
- You can't have other resources managers, for example a different database, involved in a sessionless transaction.

Before running your application, set the following system property value using the command line when you want to use XA APIs to manage sessionless transactions.

- Use the `-D` option of the java command to set the `oracle.jdbc.XAThroughSessionlessTransactions` system property to `true` using the command line.

  ```
  java -Doracle.jdbc.XAThroughSessionlessTransactions=true
  ```

So, if your existing application code uses XA transactions and implements the `XAResource` interface, then you can use the same code manage a sessionless transaction. When you set the `oracle.jdbc.XAThroughSessionlessTransactions` property value, behavior of the methods in the `XAResource` interface change as described in the following table and the transaction is treated as a sessionless transaction. In this way, you can benefit the from the performance improvements of sessionless transaction without having to change a lot in the code of an existing application.

Method	Description
`xaResource.`**`start`**`(xid, XAResource.`**`TMNOFLAGS`**`);`	Starts a sessionless transaction.
`xaResource.`**`start`**`(xid, XAResource.`**`TMRESUME`**`);`	Resumes a sessionless transaction.
`(OracleXAResource).`**`suspend`**`(xid);`	Suspends a sessionless transaction.
`xaResource.`**`end`**`(xid1, XAResource.`**`TMSUSPEND`**`);`	Suspends a sessionless transaction.
`xaResource.`**`end`**`(xid2, XAResource.`**`TMSUCCESS`**`);`	Suspends a sessionless transaction.
`xaResource.`**`setTransactionTimeout`**`(seconds);`	Sets the transaction timeout value in seconds.
`xaResource.`**`commit`**`(xid, true);`	Commits a sessionless transaction. It first resumes the transaction, and then commits the transaction.
`xaResource.`**`rollback`**`(xid);`	Rolls back a sessionless transaction. It first resumes the transaction, and then rolls back the transaction.
`xaResource.`**`join`**`(xid);`	No operation is performed in case of sessionless transaction.
`xaResource.`**`prepare`**`(xid);`	No operation is performed in case of sessionless transaction.
`xaResource.`**`recover`**`(XAResource.TMSTARTRSCAN);`	XA recover is **not supported** with sessionless transactions. This throws an exception.
`xaResource.`**`forget`**`(xid);`	No operation is performed in case of sessionless transaction.
`xaResource1.`**`isSameRM`**`(xaResource2);`	No operation is performed in case of sessionless transaction.

For more information about the `XAResource` interface, see https://docs.oracle.com/javase/8/docs/api/javax/transaction/xa/XAResource.html.

You can also manage Sessionless Transactions in a client application using JDBC APIs defined in `OracleConnection` interface. See Manage Sessionless Transactions using JDBC APIs.