13

Shared Pool for Sharded Databases

Sharding is a data tier architecture in which data is horizontally partitioned across independent databases.

This chapter describes UCP Shared Pool for sharded databases in the following sections:

13.1 Overview of UCP Shared Pool for Database Sharding

Starting from Oracle Database 12*c* Release 2 (12.2.0.1), Universal Connection Pool (UCP) supports database sharding. UCP recognizes the sharding keys specified and connects to the specific shard. Sharding uses Global Data Services (GDS), where GDS routes a client request to an appropriate database, based on various parameters such as availability, load, network latency, and replication lag.

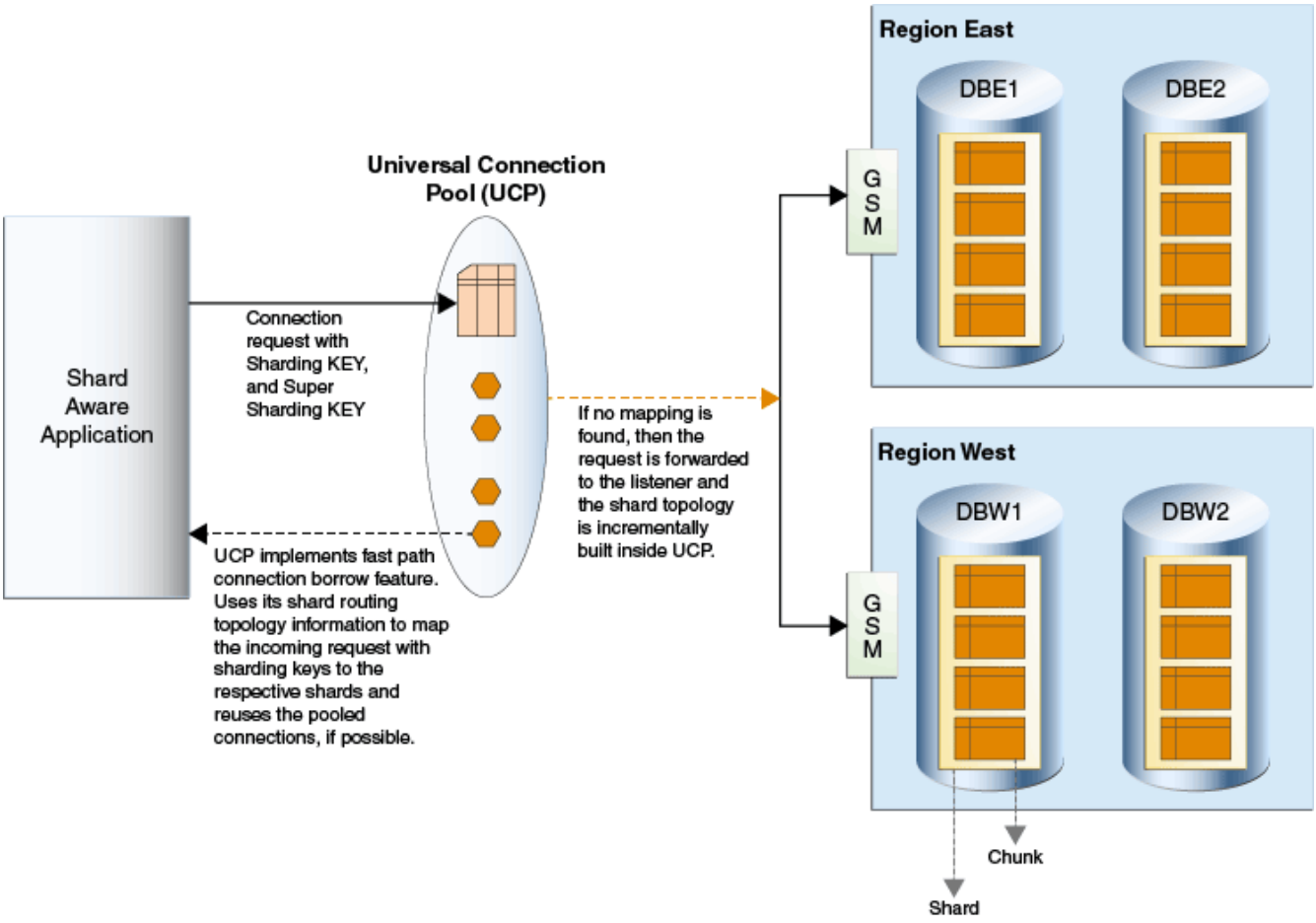
> **See Also:**
>
> - *Oracle Database JDBC Developer's Guide*
> - *Oracle Database Administrator's Guide*

Use Case of UCP Shared Pool for Database Sharding

This section describes a use case of UCP Shared Pool for database sharding. In the use case, the applications connecting to sharded database use UCP to store connections to different shards and chunks of the sharded GDS database within the same Shared Pool. The applications must provide the sharding key to UCP during the connection request. Based on the sharding key, the pool routes the connection request to the correct shard. The data distribution across the shards and chunks in the database is transparent to the user. UCP transparently handles resharding and chunk movements, minimizing the impact on the end users.

The following diagram illustrates this use case:

Figure 13-1 Universal Connection Pool (UCP) Using Sharded Database Architecture



Related Topics

- Global Data Services

13.2 About Handling Connection Requests for a Sharded Database

This section describes how connection requests are made on a pool for sharded databases.

- How to Checkout Connections from a Pool with a Known Sharding key
- About Configuring the Number of Connections Per Shard
- About Connecting to the Shard Catalog or Co-ordinator for Multishard Queries

13.2.1 How to Checkout Connections from a Pool with a Sharding Key

When a connection is borrowed from UCP, then the shard aware application can provide the sharding key and the super sharding key using the new connection builder present in the `PoolDataSource` class.

If sharding keys do not exist or do not map to the data types specified by the database metadata, then an `IllegalArgumentException` is thrown. The following code snippet shows how to checkout a connection with sharding keys:

Example 13-1 Checking Out a Connection with Sharding Keys

```
import java.sql.Connection;
import java.sql.JDBCType;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.ShardingKey;

import oracle.ucp.jdbc.PoolDataSource;
import oracle.ucp.jdbc.PoolDataSourceFactory;

public class UCPShardingExample {

  public static void main(String[] args) throws SQLException {
      String url = "jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=myhost)
(PORT=<gsm_port>)(PROTOCOL=tcp))(CONNECT_DATA=(SERVICE_NAME=myGSMservice)))";
      String user="db_user_name";
      String pwd = "db_password";

      PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
      pds.setURL(url);
      pds.setUser(user);
      pds.setPassword(pwd);
      pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setInitialPoolSize(5);
      pds.setMinPoolSize(5);
      pds.setMaxPoolSize(20);

      int empId = 1234;
      // Employee ID is the sharding key column in sharded table

      ShardingKey shardingKey = pds.createShardingKeyBuilder()
                                   .subkey(empId, JDBCType.INTEGER)
                                   .build();

      // Borrow a connection to direct shard using sharding key
      try(Connection connection = pds.createConnectionBuilder()
              .shardingKey(shardingKey)
              .build()) {

                           PreparedStatement pst =
connection.prepareStatement("select * from employee where emp_id=?");
                           pst.setInt(1, 1234);
                           ResultSet rs = pst.executeQuery();
```

```
                        // retrieve the employee details using resultset
                        rs.close();
                        pst.close();
                    }
            }

    }
```

> **Note:**
>
> You must specify a sharding key during the connection checkout. Otherwise, an error or exception is thrown back to the application.

13.2.2 About Configuring the Number of Connections Per Shard

When UCP is used to pool connections for a sharded database, the pool contains connections to different shards. So, when connections are pulled, to ensure a fair usage of the pool capacity across all shards connected, UCP uses the `MaxConnectionsPerShard` parameter. This is a global parameter, which applies to every shard in the sharded database, and is used to limit the total number of connections to any shard below the specified limit.

The following table describes the APIs for setting and retrieving this parameter:

Method	Description
`poolDatasource.setMaxConnectionsPerShard(<max_connections_per_shard_limit>)`	Sets the maximum number of connections per shard.
`poolDatasource.getMaxConnectionsPerShard()`	Retrieves the value that was set using the `setMaxConnectionsPerShard(<max_connections_per_shard_limit>)` method.

> **Note:**
>
> You cannot use the `MaxConnectionsPerShard` parameter in a sharded database with Oracle Golden Gate configuration.

13.2.3 About Connecting to the Shard Catalog or Co-ordinator for Multishard Queries

When connecting to the Shard Catalog or Co-ordinator for running multishard queries, it is recommended that a separate pool be created using a new `PoolDataSource` instance. You can run multishard queries on connections retrieved from a data source that is created on the coordinator service. The connection request for the coordinator should not have sharding keys in the connection builder API.

13.3 Sharding Data Source for Transparent Access to Sharded Databases

Oracle Database Release 21c introduced a new JDBC data source that enables Java connectivity to a sharded database without the need for an application to furnish a sharding key.

If you use the sharding data source, then you do not have to identify and build the sharding key and the super sharding key to establish a connection to the sharded database, as discussed in the earlier How to Checkout Connections from a Pool with a Sharding Key section. The sharding data source also eliminates the need to maintain a separate data source for multishard queries.

This data source scales out to sharded databases transparently if you set the connection property `oracle.jdbc.useShardingDriverConnection` to `true`.

> **See Also:**
>
> *Oracle Database JDBC Developer's Guide*

The following code snippet shows how to use the sharded data source:

Example 13-2 Using the Sharded Data Source

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Properties;

import javax.sql.DataSource;

import oracle.jdbc.internal.OracleConnection;
import oracle.ucp.jdbc.PoolDataSource;
import oracle.ucp.jdbc.PoolDataSourceFactory;

public class ShardingDataSourceUCP {
    public static void main(String[] args) throws SQLException {
        ShardingDataSourceUCP sample = new ShardingDataSourceUCP();
        DataSource ucpDataSource = sample.getDataSource();
        // Get the details of following customers
        int[] customerIds = new int[] {
            100,
            101,
            102,
            103,
            104,
            105
        };

        for (int id: customerIds) {
            try (Connection conn = ucpDataSource.getConnection()) {
```

```
                sample.displayCustomerDetails(conn, id);
                System.out.println(((OracleConnection)
conn).getPercentageQueryExecutionOnDirectShard());
            }
        }
    }

    private void displayCustomerDetails(Connection conn, int id) throws
SQLException {
        try (PreparedStatement pstmt = conn.prepareStatement("SELECT * FROM
CUSTOMER where ID = ?")) {
            pstmt.setInt(1, id);
            try (ResultSet rs = pstmt.executeQuery()) {
                while (rs.next()) {
                    // Print the customer details
                }
            }
        }
    }

    private DataSource getDataSource() throws SQLException {
        PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
        pds.setURL("<gsmURL>");
        pds.setUser("<userName>");
        pds.setPassword("<password>");

pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
        Properties prop = new Properties();
        // Connection property to enable sharding datasource feature, when
this property
        // is set you don't need to pass sharding key to UCP pool while
borrowing the connection

prop.setProperty(OracleConnection.CONNECTION_PROPERTY_USE_SHARDING_DRIVER_CONN
ECTION, "true");
        pds.setConnectionProperties(prop);
        return pds;
    }
}
```

See Also:

- The UCPConnectionBuilder Interface
- The PoolDataSource Interface
- The PoolXADataSource Interface

13.3.1 Support for Single Shard Transactions

The sharding data source enables you to limit your transactions to one single shard.

To enable single shard transaction support, you must set `CONNECTION_PROPERTY_ALLOW_SINGLE_SHARD_TRANSACTION_SUPPORT`. If this property is not set, then by default, all the transactions are started on the Shard Catalog. If you set the value of this property to `true`, then you must ensure that all the transactions span over a single shard only.

> **See Also:**
>
> - Oracle Database JDBC Java API Reference
> - *Oracle Database JDBC Developer's Guide*

Example 13-3 Enabling Single Shard Transactions

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Properties;

import javax.sql.DataSource;

import oracle.jdbc.internal.OracleConnection;
import oracle.jdbc.pool.OracleDataSource;

public class SingleShardTransactionUCP {
    public static void main(String[] args) throws SQLException {
        SingleShardTransactionUCP sample = new SingleShardTransactionUCP();
        DataSource ucpDS = sample.getDataSource();

        // Insert and update the details of following customers in a single
transaction
        int[] customerIds = new int[] {
            100,
            101,
            102,
            103,
            104,
            105
        };

        for (int id: customerIds) {
            try (Connection conn = ucpDS.getConnection()) {
                conn.setAutoCommit(false);
                sample.insertCustomerDetails(conn, id);
                sample.displayCustomerDetails(conn, id);
                sample.updateCustomerDetails(conn, id);
                sample.displayCustomerDetails(conn, id);
```

```
                conn.commit();
                System.out.println(((OracleConnection)
conn).getPercentageQueryExecutionOnDirectShard());
            }

        }
    }

    private void insertCustomerDetails(Connection conn, int id) throws
SQLException {
        String sql = "insert into CUSTOMER values(?, ?, ?, ?)";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, id);
            ps.setString(2, name);
            ps.setString(3, email);
            ps.setString(4, phoneNumber);
            ps.executeUpdate();
        }
    }

    private void updateCustomerDetails(Connection conn, int id) throws
SQLException {
        String sql = "UPDATE CUSTOMER SET name = ?, email = ?, phoneNumber
= ? WHERE customerId = ?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setString(1, name);
            ps.setString(2, email);
            ps.setString(3, phoneNumber);
            ps.setInt(4, id);
            ps.executeUpdate();
        }
    }

    private void displayCustomerDetails(Connection conn, int id) throws
SQLException {
        try (PreparedStatement pstmt = conn.prepareStatement("SELECT * FROM
CUSTOMER where ID = ?")) {
            pstmt.setInt(1, id);
            try (ResultSet rs = pstmt.executeQuery()) {
                while (rs.next()) {
                    //Print the customer details
                }
            }
        }
    }

    private DataSource getDataSource() throws SQLException {
        OracleDataSource ds = new OracleDataSource();
        ds.setURL( < gsmURL > );
        ds.setUser( < userName > );
        ds.setPassword( < password > );
        Properties prop = new Properties();
        //Connection property to enable sharding datasource feature

prop.setProperty(OracleConnection.CONNECTION_PROPERTY_USE_SHARDING_DRIVER_CONN
ECTION, "true");
```

```
        //Connection property to enable single shard transaction support. If
this property is not set,
        // by default all the transactions are started on catalog DB. When
setting this property value
        // to "true", applications must ensure that all the transactions span
over a single shard only.

prop.setProperty(oracle.jdbc.OracleConnection.CONNECTION_PROPERTY_ALLOW_SINGLE
_SHARD_TRANSACTION_SUPPORT, "true");
        ds.setConnectionProperties(prop);
        return ds;
    }
}
```

13.4 Middle-Tier Routing Using UCP

Since Oracle Database Release 18c, Oracle Universal Connection Pool (UCP) supports the Middle-Tier Routing feature. This feature helps the Oracle customers, who use the Sharding feature, to have a dedicated middle tier from the client applications to the sharded database.

Typically, the middle-tier connection pools route database requests to specific shards. During such a routing, each middle-tier connection pool establishes connections to each shard, creating too many connections to the database. The Middle-Tier Routing feature solves this problem by having a dedicated middle tier (Web Server or Application Server) for each Data Center or Cloud, and routing client requests directly to the relevant middle tier, where the shard containing the client data (corresponding to the client sharding key) resides.

13.4.1 Middle-Tier Routing with UCP Example

The following example explains the usage of the middle-tier routing API of UCP.

Example 13-4 Example of Middle-Tier Routing Using UCP

```
import java.sql.SQLException;
import java.util.Properties;
import java.util.Random;
import java.util.Set;

import oracle.jdbc.OracleShardingKey;
import oracle.jdbc.OracleType;
import oracle.ucp.UniversalConnectionPoolException;
import oracle.ucp.routing.ShardInfo;
import oracle.ucp.routing.oracle.OracleShardRoutingCache;

/**
 * The code example illustrates the usage of the middle-tier routing feature
of UCP.
 * The API accepts sharding key as input and returns the set of ShardInfo
 * instances mapped to the sharding key. The ShardInfo instance encapsulates
 * unique shard name and priority. The unique shard name then can be mapped
 * to a middle-tier server that connects to a specific shard.
 *
 */
public class MidtierShardingExample {
```

```
  private static String user = "testuser1";
  private static String password = "testuser1";

  // catalog DB URL
  private static String url = "jdbc:oracle:thin:@//hostName:1521/
catalogServiceName";
  private static String region = "regionName";

  public static void main(String args[]) throws Exception {
    testMidTierRouting();
  }

  static void testMidTierRouting() throws UniversalConnectionPoolException,
      SQLException {

    Properties dbConnectProperties = new Properties();
    dbConnectProperties.setProperty(OracleShardRoutingCache.USER, user);
    dbConnectProperties.setProperty(OracleShardRoutingCache.PASSWORD,
password);
    // Mid-tier routing API accepts catalog DB URL
    dbConnectProperties.setProperty(OracleShardRoutingCache.URL, url);

    // Region name is required to get the ONS config string
    dbConnectProperties.setProperty(OracleShardRoutingCache.REGION, region);

    OracleShardRoutingCache routingCache = new OracleShardRoutingCache(
        dbConnectProperties);

    final int COUNT = 10;
    Random random = new Random();

    for (int i = 0; i < COUNT; i++) {
      int key = random.nextInt();
      OracleShardingKey shardKey = routingCache.getShardingKeyBuilder()
          .subkey(key, OracleType.NUMBER).build();
      OracleShardingKey superShardKey = null;

      Set<ShardInfo> shardInfoSet = routingCache.getShardInfoForKey(shardKey,
          superShardKey);

      for (ShardInfo shardInfo : shardInfoSet) {
        System.out.println("Sharding Key=" + key + " Shard Name="
            + shardInfo.getName() + " Priority=" + shardInfo.getPriority());
      }
    }

  }
}
```

13.5 Sharding with JTA/XA Transaction in WebLogic Server

Starting from Oracle Database 23ai Release, you can use sharding with JTA/XA transactions, when you configure UCP as a native data source in WebLogic Server. This feature broadens

UCP middle-tier coverage to include more applications, like the ones that require container-managed JTA/XA transactions or sharding.

Java Transaction API (JTA) specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications. Now, Java Enterprise applications that use UCP native data source sharding APIs, to obtain connections to Oracle sharded databases, can participate in JTA/XA transactions managed by WebLogic Transaction Manager (TM).

For any XA transaction with all supplied sharding keys parameter in the connection requests, which lead to the same Oracle sharded database instance, the XA transaction goes through successfully and commits the changes. For an XA transaction, leading to different Oracle sharded database instances, UCP native data source raises an exception to the WebLogic Transaction Manager and the transaction is rolled back.

See Also:

Oracle WebLogic Server documentation for more information