Connection Management Strategies for Java Applications using JDBC and UCP

Oracle Database 12c

ORACLE WHITE PAPER JUNE 2016

Introduction 2

Quick Introduction of JDBC & UCP 3

Connection Management Strategy for **Performance** 4

1.1 Single Database/Single Instance 4

1.2 Real Application Clusters (RAC) 7

1.3 Multitenant Database – Multiple Databases/Single Instance 9

1.4 Data Guard or Active Data Guard - Multiple Datacenters/Single Instance 9

1.5 DG or ADG with RAC - Multiple Datacenters/Multiple Instances 10

1.6 Global Data Services (GDS) – Multiple Geographies 11

Connection Management Strategy for **Scalability & Load Balancing** 11

2.1 Single Database/Single Instance 11

2.2 Real Application Clusters (RAC) 14

2.3 Multitenant Database 15

2.4 Data Guard or Active Data Guard 15

2.5 DG or ADG with RAC 15

2.6 Global Data Services (GDS) 16

Connection Management Strategies for **High Availability (HA)** 16

3.1 Single Instance Database 16

3.2 RAC ONE Node 17

Hiding planned maintenance 17

Hiding unplanned downtime 18

3.3 Real Application Clusters (RAC) 19

3.4 Data Guard or Active Data Guard 19

3.5 DG or ADG with RAC 20

3.6 Global Data Services (GDS) 20

Connection Management Strategy for **Security** 21

Connection Management Strategy for **Manageability** 25

4.1 Single Instance Database 25

4.2 Real Application Clusters (RAC) 26

4.3 DG or ADG 27

4.4 DG or ADG with RAC 27

4.5 Global Data Services (GDS) 27

Conclusion 28

Introduction

Architects, application developers, and DBAs strive to achieve the best system throughput by tuning the RDBMS, the operating system, and Java applications. Database connections play a crucial role in application performance, scalability, availability, security, and manageability. In many cases, applications perform sub-optimally because of short term design decisions, incorrect database configurations, poor tuning and poor understanding of database capabilities.

The Oracle database along with the Oracle JDBC drivers and the Oracle Universal Connection Pool (UCP) offer many connection management strategies to improve the quality of services in terms of performance, scalability, availability, security, and manageability. The strategies consist in setting the appropriate connection descriptors and properties, choosing the appropriate connection pool, tuning the pool to meet the application behavior, using connection affinity, using connection labeling, leveraging high availability features such as Application Continuity(AC) or Transaction Guard(TG) for achieving smooth planned maintenance and hiding unplanned outages; and using the security and manageability features etc., This whitepaper makes recommendations based on your database configuration such as Single Instance Database, Oracle Real Application Clusters (RAC), Oracle Multitenant, Oracle Data Guard (DG), Oracle Active Data Guard (ADG), and Oracle Global Data Services (GDS).

The paper is structured around performance, scalability, availability, security, and manageability. Every section presents the recommendations related to each Oracle database 12c configuration.

The companion connection management code samples[1] are available for download from Oracle Technology Network (OTN) and also from GITHUB.

[1] OTN @ http://www.oracle.com/technetwork/database/features/jdbc/default-2345085.htm or GITHUB @ http://github.com/oracle/jdbc-ucp

Quick Introduction of JDBC & UCP

The Oracle JDBC drivers implement and comply with the latest JDBC specifications. Java applications need to have **`ojdbc7.jar`** (for JDK 7 and JDK8) or **`ojdbc6.jar`** (for JDK 6) in their `classpath`. Refer to “JDBC Developer’s guide“[2] for more details. The following code sample shows how to obtain a JDBC connection.

```java
//Use connection descriptors
final static String DB_URL=
"jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(HOST=myhost)(PORT=1521)(PROTOCOL=tcp
))(CONNECT_DATA=(SERVICE_NAME=myorcldbservicename)))";
final static String DB_USER = "hr";
final static String DB_PASSWORD = "hr";
//Set connection level properties
Properties connProps = new Properties();
connProps.put(OracleConnection.CONNECTION_PROPERTY_USER_NAME,DB_USER);
connProps.put(OracleConnection.CONNECTION_PROPERTY_PASSWORD,DB_PASSWORD);

OracleDataSource ods = new OracleDataSource();
ods.setURL(DB_URL);
ods.setConnectionProperties(connProps);
OracleConnection connection = (OracleConnection) ods.getConnection();
```

Oracle Universal Connection Pool (UCP) is a feature rich Java connection pool tightly integrated with all Oracle database configurations, providing high availability, scalability and work load balancing; In addition, UCP may be used with non-Oracle JDBC drivers against non-Oracle databases. To use UCP, Java applications or containers must have **`ucp.jar`**[3] in their class path along with **`ojdbc7.jar`** (for JDK 7 and JDK8) or **`ojdbc6.jar`** (for JDK 6). Many third party connection pools provide basic functionalities. UCP stands out from other connection pools because of its tighter integration with Oracle Real Application Clusters (RAC), Active Data Guard (ADG), and Global Data Services (GDS). Refer to “UCP Developer’s guide”[4] for more details.

The following code fragment shows how to retrieve a connection with UCP.

```java
// Get the PoolDataSource for UCP
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
//Set the connection factory first before all other properties
pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
pds.setURL(DB_URL);
pds.setUser(DB_USER);
pds.setPassword(DB_PASSWORD);
//Set the pool level properties
pds.setConnectionPoolName("JDBC_UCP_POOL");
pds.setInitialPoolSize(5);
pds.setMinPoolSize(5);
pds.setMaxPoolSize(20);
Connection conn = pds.getConnection();
```

[2] JDBC Developer’s Guide @ https://docs.oracle.com/database/121/JJDBC/toc.htm

[3] Download 12.1.0.2 UCP.jar from OTN @ http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html

[4] UCP Developer’s Guide @ https://docs.oracle.com/database/121/JJUCP/toc.htm

Connection Management Strategy for **Performance**

Many factors affect performance of Java applications with Oracle database. This section focuses on the appropriate strategy for each database configuration.

1.1 Single Database/Single Instance

A single instance database has a *one-to-one relationship between the Oracle database and the instance.*

Recommendation 1: **Use Connection URL for a Single Instance Database**

The recommended format of a connection URL is a long format with descriptors as they allow passing parameters such as CONNECT_TIMEOUT and other that can improve performance.

```
jdbc:oracle:thin:@(DESCRIPTION=(CONNECT_TIMEOUT=15) (RETRY_COUNT=20)
(RETRY_DELAY=3)  (ADDRESS=(PROTOCOL=tcp)(HOST=myhost-vip)(PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=myorcldbservicename))
```

CONNECT_TIMEOUT: When enabled, this parameter instructs Oracle Net services to wait for the specified number of seconds (15 seconds in the example) for the completion of the connection establishment. This is equivalent to SQLNET.OUTBOUND_CONNECT_TIMEOUT which specifies the time for the client to establish a connection to the oracle database instance. CONNECT_TIMEOUT overrides SQLNET.OUTBOUND_CONNECT_TIMEOUT.
RETRY_COUNT: It specifies the number of network connect retry attempts before returning a failure message to the client. In the example above, Oracle Net retries 3 times before returning an error message to the client. This helps in increasing the possibility of getting a connection and thus improves the performance.
RETRY_DELAY: This parameter specifies the wait time in seconds between reconnection attempts. It works in conjunction with RETRY_COUNT. So, it is advised to use RETRY_DELAY and RETRY_COUNT together to avoid unnecessary CPU cycles.

Recommendation 2: **Choose the performance related connection properties.**

Note: Refer to `OracleConnection`[5] for a complete list of connection level properties.

Session Data Unit (SDU): It sets the size of the network buffer that Oracle Net uses for transmitting data back to the Java application. SDU size can be tuned to optimize the throughput of data packets being sent across the network thereby improving performance, network utilization and memory consumption. SDU size for the client can be configured in `sqlnet.ora`, `tnsnames.ora` or in the connection URL. You may change the DEFAULT_SDU_SIZE in `sqlnet.ora` for all connections, or in tnsnames.ora for specific services or in the connection URL for a specific application. With Oracle database 12c the default SDU size is 8K which can be increased to 2MB, if necessary. SDU should be increased with caution as it applies to each connection.

```
jdbc:oracle:thin:@(DESCRIPTION=(SDU=11280)
(ADDRESS=(PROTOCOL=tcp)(HOST=myhost-vip)(PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=myorcldbservicename)))
```

[5] OracleConnection in JDBC Javadoc @ http://docs.oracle.com/database/121/JAJDB/toc.htm

Increase SDU in the following scenarios:

- While using a wide area network (WAN) that has long delays.
- When larger amounts of data are returned.
- When transferring XML& JSON documents

DO NOT modify the default SDU value in the following scenarios:

- Using a high speed network where the effect of the data transmission is negligible
- When the requests return small amounts of data from the server

CONNECTION_PROPERTY_THIN_READ_TIMEOUT: Enable read timeout on sockets to avoid connections from being severed due to firewall timeout and thereby causing applications to hang waiting for a response from the RDBMS. The timeout value is in milliseconds.

Recommendation 3: **Use Java in the Database**

"Java in the database" provides the ability to group SQL operations with Java data logic and load in the database for in-place data processing. Java in the database comes with a "server side internal driver" for direct access to data and PL/SQL subprograms through local calls. Java in the database is recommended for applications that are data-intensive as it does not incur network calls experienced by client applications. Applications will achieve higher performance and faster execution. The connection string for getting a connection through a server side driver is as shown below.

```
 OracleDataSource ods = new OracleDataSource();
ods.setURL("jdbc:oracle:kprb");
// alternatively any of the following  URLs
// ods.setURL("jdbc:default:connection");
Connection connection = ods.getConnection();
```

Refer to **InternalT2Driver.java, InternalT2Driver .sql, InternalT4Driver.sql,** and **InternalT4Driver.java** in connection management code samples.

Recommendation 4: **Use Universal Connection Pool (UCP)**

Universal Connection Pool (UCP)
Database connection creation and deletion is an expensive operation and repeated creation/deletion lead to performance/scalability issue. A connection pool promotes the reuse of connection objects and reduce the number of times the connection objects are created or deleted. Also, an application waits less time to get a connection as the connections are already created. The recommendation is to use Universal Connection Pool (UCP)[6] a Java connection pool for managing database connections to improve performance and to better utilize system resources of Java applications.

[6] Refer to UCP Developers Guide for more details http://docs.oracle.com/database/121/JJUAR/toc.htm

UCP Performance Properties

UCP properties are used to control the connection pool size, handle stale connections, and balance quick response times. The optimal settings for the pool properties depend on the application and hardware resources. In many cases, it is necessary to try different settings to find an optimal balance. Some of the performance properties are mentioned here.

MaxPoolSize specifies the maximum number of connections that a pool maintains to ensure it does not exhaust system resources. The recommendation is to set it to a value based on the number of connections expected from the application and tune it as required. The value can also be set as suggested by the following formula.

MaxPoolSize = *(rdbms-cores * n) / sum (pools-per-mid-tier)*
where n is an integer with a typical recommended value of 9 or 10
Example:
Consider a single node database server with 4 cores per node and 5 mid-tiers each running a single JVM with one pool each.
MaxPoolSize = (4 * 10) / (5 * 1) = (4 * 10) / 5 = 40 / 5 = 8
so MaxPoolSize should be 8 for each mid-tier, as a first approximation.
Applications that require an even number of connections per mid-tier can use the above formula. However, in scenarios where connections required per mid-tier varies, we suggest to compute the optimal connection workload that the database can sustain then split these connections among mid-tiers. In the example above, 40 is the total number of connections that can be split among 5 mid-tiers based on the connection requirement.

MinPoolSize specifies the minimum number of available connections that a pool maintains. Set this value to the minimum number of connections required by your application at any given time and tune it as required. MinPoolSize should be less than or equal to MaxPoolSize. The default value is 0.

InitialPoolSize specifies the number of connections that are created when the pool is created or re-initialized. It should be closer to MinPoolSize, which will let the connection pool start faster. The default value is 0 which means that no connections are pre-created.

In applications where the number of concurrent active connections are known and memory is not a concern then set *MaxPoolSize*, *InitialPoolSize*, and *MinPoolSize* to the same value.

MaxStatements specifies the size of the SQL statement for each connection. Statement caching lets cursors be re-executed without reparsing the statement, eliminating repetitive statement parsing thereby improving the performance and scalability. By default, statement caching is disabled.
MaxStatements should be set to the number of frequently used SQL statements by the application. So, use a number that is neither too high nor too small.

TimeToLiveConnectionTimeout allows a borrowed connection to remain borrowed only for a pre-determined period of time. When this period expires, the connection is reclaimed back into the pool. It is enforced even if the connection is in use. This timeout helps maximize connection reuse and helps conserve system resources. It should be set sufficiently high, based on the application profile.

AbandonedConnetionTimeout: a borrowed connection is considered abandoned when there is no activity for an extended period of time; then it is reclaimed back into the pool. This timeout helps maximize connection reuse and helps conserve system resources. It should be set to twice or three times the longest duration that connection may be checked out, including failover time.

InactiveConnectionTimeout specifies how long an available connection can remain idle before it is closed and removed from the pool. It is only applicable to available connections, not borrowed ones. This timeout helps conserve resources. The default value is 0. A non-zero value depends on application requirements.

ConnectionWaitTimeout specifies how long an application may wait before obtaining a connection. Upon the expiration of this timeout, an exception is thrown. This timeout improves overall application usability by minimizing the amount of time an application is blocked and provides the ability to implement a graceful recovery. The default value is 3 seconds. Choose the value based on the application.

Refer to **UCPSample.java** and **UCPWithTimeoutProperties.java** code samples in connection management code samples.

Recommendation 5: **Use Connection Labelling**

Universal Connection Pool (UCP) enables an application to associate custom labels i.e., non-transactional states (e.g., NLS, Transaction Isolation or custom state) to a connection then later search for the same connections. Connection labeling avoids the time and cost of connection re-initialization. For more details on “Connection Labeling”, refer to *UCP Developer’s guide*[7].

Refer to **UCPConnectionLabelingSample.java** in connection management code samples.

1.2 Real Application Clusters (RAC)

The Oracle Real Application Clusters[8] (Oracle RAC) configuration has *one-to-many relationship between the Oracle database and one or many instances.* Oracle RAC satisfies high performance, increased throughput, high availability, and expanded scalability requirements. Oracle RAC can be deployed with Oracle Multitenant and Oracle Active Data Guard (ADG).

All connection management strategies discussed under “Single Instance/Single Database” are applicable to Real Application Clusters (RAC). In addition, the following recommendations apply.

Recommendation 6: **Connection URL for a RAC database**

Performance related connection descriptors are explained in this section.

```
jdbc:oracle:thin:@
(DESCRIPTION=
  (CONNECT_TIMEOUT=15)(RETRY_COUNT=20) (RETRY_DELAY=3)
       (ADDRESS_LIST =
           (LOAD_BALANCE=ON)
               (ADDRESS=(PROTOCOL=tcp)(HOST=primaryscan)(PORT=1521)))
  (CONNECT_DATA=(SERVICE_NAME=myorcldbservicename)))
```

CONNECT_TIMEOUT, RETRY_COUNT, RETRY_DELAY, and SERVICE_NAME: Refer to

[7] UCP Developers Guide @ https://docs.oracle.com/database/121/JJUCP/toc.htm

[8] Refer to RAC datasheet for more details on benefits www.oracle.com/technetwork/database/options/clustering/rac-ds-12c-1898881.pdf

“Recommendation 1: Connection URL for a Single Instance Database” for detailed explanation of these descriptors.
HOST: Always use a SCAN name as it provides better manageability. Refer to “Recommendation 33: Use Single Client Access Name (SCAN)” for more details.
SERVICE_NAME: Always use application service name. Services provide location transparency, facilitate load balancing, enable recovery of work, and provide performance attributes. Default database service corresponds to db_name, db_unique name or PDB name. Avoid using default database services which are strictly used for database administration usage and reserved for Oracle Enterprise Manager and DBAs.

Recommendation 7: **Use Universal Connection Pool (UCP)**

Universal Connection Pool is tightly integrated with Oracle RAC and is a proven solution to leverage all benefits and features offered by the RAC database. Refer to “Single Instance/Single Database” section for more details about UCP and its performance properties. In a RAC environment, set *MaxPoolSize* as suggested by the following formula.

*MaxPoolSize = (sum (rdbms-cores-per-instance) * n) / sum (pools-per-mid-tier)*
where n is an integer with a typical recommended value of 9 or 10.
Example:
Consider a 3 nodes Oracle RAC cluster with 24 cores per node and 10 mid-tiers each running 2 JVMs with one pool each and n=10.
MaxPoolSize = (sum (24, 24, 24) * 10) / (10 * 2)
= (72 * 10) / 20 = 720 / 20 = 36
so *MaxPoolSize* should be 36 as a first approximation.
Applications that require an even number of connections per mid-tier can use the above formula. However, in scenarios where connections required per mid-tier varies then compute the optimal connection workload then split these connections among mid-tiers. In the example above, 720 is the total number of connections that can be split among 10 mid-tiers based on the connection requirement.

Recommendation 8: **Use Connection Affinity**

Connection Affinity is a RAC performance feature that enables a pool to select connections that are directed at a specific RAC instance. The pool uses run-time connection load balancing (if configured) to select an instance specific connection; then subsequent connections request from the same application are allocated with an affinity to the same instance. UCP supports two connection affinity types (a) Web Session Affinity and (b) Transaction based Affinity[9] (a.k.a. XA Affinity).

UCP Web Session Affinity:
With web session affinity, UCP attempts to allocate a connection to the same database instance, for consecutive connection requests from the same web session. The application must implement an affinity callback.
A code fragment of web session affinity callback is shown here. Refer to **UCPWebSessionAffinitySample.java** in connection management code samples.

[9] Transaction Affinity aka XA Affinity is not covered in this whitepaper

```
MyConnectionAffinityCallback callback = new MyConnectionAffinityCallback();
poolDataSource.registerConnectionAffinityCallback(callback);
// Web Session Affinity Callback implementation
public class MyConnectionAffinityCallback implements
ConnectionAffinityCallback {
    // Affinity policy used by callback
    ConnectionAffinityCallback.AffinityPolicy affinityPolicy =
        ConnectionAffinityCallback.AffinityPolicy.WEBSESSION_BASED_AFFINITY;

    public MyConnectionAffinityCallback() {
    }
    // Other Affinity callback methods go here
    ...
  }
```

1.3 Multitenant Database – Multiple Databases/Single Instance

Oracle Multitenant is an option with Oracle Database 12*c* that simplifies consolidation, provisioning, upgrades, and more. It allows a container database called CDB to manage many pluggable databases abbreviated as PDB. PDBs are full-fledged databases containing customers and applications data and metadata. Oracle Multitenant fully complements other options, including Oracle Real Application Clusters and Oracle Active Data Guard. Connecting to PDBs through Oracle Net is similar to connecting to a non-CDB database.

All connection management strategies discussed under “Single Instance/Single Database” are applicable to a Multitenant database. In addition, the following recommendations apply.

Recommendation 9: **Connection URL for a Multitenant database**

```
jdbc:oracle:thin:@(DESCRIPTION=
(ADDRESS=(PROTOCOL=tcp)(HOST=myhost-vip)(PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=mypdbservice)))
```

SERVICE_NAME: The service name must be a valid service name associated with a given PDB and unique within the CDB. DO NOT use service name that correlates to db_unique_name (database or PDB name).

Recommendation 10: **Use a Shared UCP Pool**

“A shared UCP pool” enables sharing a single pool of connections across multiple PDBs (or tenants) to improve the performance and better utilize the database resources. The solution available in Oracle database 12.1 requires few configuration and programmatic steps for reusing a connection across PDBs. A shared UCP pool requires a combination of UCP Connection labeling, Global database user (with access privileges to any PDB), and the new SET CONTAINER statement within a call back function. Refer to “UCP Developer’s guide”[10] for step by step instructions for creating a shared pool across multiple PDBs and also refer to **UCPMultitenantSample.java** and **UCPMultitenantSample.sql** in connection management code samples.

1.4 Data Guard or Active Data Guard - Multiple Datacenters/Single Instance

[10] UCP Developer’s Guide @ https://docs.oracle.com/database/121/JJUCP/toc.htm

Oracle Data Guard ensures high availability, data protection, and disaster recovery for enterprise data. Oracle Data Guard maintains standby databases as copies of the production data center. Then, if the production data center becomes unavailable, Oracle Data Guard switches the standby database to the production role, minimizing the downtime associated with the outage. Oracle Data Guard is included with Oracle Enterprise Edition with no additional license fees.

Active Data Guard (ADG) [11] is a superset of Data Guard (DG) and furnishes the following additional features: Real-Time Query (performance and ROI), Far Sync (Zero Data Loss Protection at any Distance), Automatic Block Repair (HA) and Database Rolling Upgrades. Active Data Guard requires additional license with Oracle Database Enterprise Edition.

All connection management strategies discussed under “Single Instance/Single Database” are applicable to a Data Guard or Active Data Guard database configuration and make sure to use a connection URL with relevant connection descriptors as shown below.

Recommendation 11: **Connection URL for DG or ADG**

```
jdbc:oracle:thin:@(DESCRIPTION = (FAILOVER=on) (LOAD_BALANCE=off)
     (CONNECT_TIMEOUT= 15) (RETRY_COUNT=20) (RETRY_DELAY=3)
       (ADDRESS_LIST =
           (ADDRESS = (PROTOCOL = TCP) (HOST=primary-vip) (PORT=1521)))
      (ADDRESS_LIST =
             (ADDRESS = (PROTOCOL = TCP) (HOST=secondary-vip) (PORT=1521)))
      (CONNECT_DATA= (SERVICE_NAME = gold-cloud-service-name)))
```

CONNECT_TIMEOUT, RETRY_COUNT, RETRY_DELAY, and SERVICE_NAME: Refer to “Recommendation 1: Connection URL for a Single Instance database” for detailed explanation of these descriptors.

1.5 DG or ADG with RAC - Multiple Datacenters/Multiple Instances

Oracle RAC and Data Guard architecture combines the benefits of both RAC and DG and is the recommended architecture for Maximum Availability Architecture (MAA).

All connection management strategies discussed under “Single Instance/Single Database” are applicable to DG or ADG with RAC. An important strategy is to use a connection URL with connection descriptors relevant to ADG/DG with RAC.

Recommendation 12: **Connection URL for DG or ADG with RAC**

```
jdbc:oracle:thin:@(DESCRIPTION = (FAILOVER=on) (LOAD_BALANCE=off)
     (CONNECT_TIMEOUT= 15) (RETRY_COUNT=20) (RETRY_DELAY=3)
       (ADDRESS_LIST =
           (LOAD_BALANCE=on)
           (ADDRESS = (PROTOCOL = TCP) (HOST=primary-scan) (PORT=1521)))
      (ADDRESS_LIST =
             (LOAD_BALANCE=on)
             (ADDRESS = (PROTOCOL = TCP) (HOST=secondary-scan) (PORT=1521)))
      (CONNECT_DATA= (SERVICE_NAME = gold-cloud-service-name)))
```

[11] Refer to ADG datasheet @ www.oracle.com/technetwork/database/availability/active-data-guard-wp-12c-1896127.pdf

CONNECT_TIMEOUT, RETRY_COUNT, RETRY_DELAY, and SERVICE_NAME: Refer to "Recommendation1: Connection URL for Single Instance database" for detailed explanation of these descriptors.

1.6 Global Data Services (GDS) – Multiple Geographies

Oracle Global Data Services (GDS) [12] caters to applications that efficiently serve geographically distributed customers from the instances that are closely situated to the user. GDS is integrated with Oracle Data Guard broker and is included with Oracle Active Data Guard.

All connection management strategies discussed under "Single Instance/Single Database" are applicable to a GDS database configuration and along with that, make sure to use a connection URL with descriptors relevant to GDS.

Recommendation 13: **Connection URL for Global Data Services (GDS)**

```
jdbc:oracle:thin:@(DESCRIPTION= (FAILOVER=on) (LOAD_BALANCE=off)
     (CONNECT_TIMEOUT=15) (RETRY_COUNT=20) (RETRY_DELAY=3)
     (ADDRESS_LIST=
        (LOAD_BALANCE=ON)
        (ADDRESS= (PROTOCOL= TCP) (HOST=myorclgsm1) (PORT=1571)))
      (ADDRESS_LIST=
         (LOAD_BALANCE=ON)
         (ADDRESS= (PROTOCOL= TCP) (HOST=myorclgsm2) (PORT=1572)))
     (CONNECT_DATA= (SERVICE_NAME=gold-cloud-service-name) (REGION=region1))
```

CONNECT_TIMEOUT, RETRY_COUNT, RETRY_DELAY: Refer to "Recommendation 1: Connection URL for a RAC database" for detailed explanation of these descriptors.
SERVICE_NAME: A global service is a database service provided by multiple databases synchronized through data replication. Recommendation is to use a service name that is a **global service** name in case of GDS.

Connection Management Strategy for **Scalability & Load Balancing**

This section focuses on improving scalability and addressing load balancing by choosing an appropriate strategy for each database configuration.

2.1 Single Database/Single Instance

Recommendation 14: **Use Database Services**

A Database Service represents a single database and divides workloads into mutually disjoint groups. Use database service for a workload with common attributes, service-level thresholds, and priorities. Database services enable you to configure workload for a single database, administer it, enable and disable it, and measure the workload as a single entity. Using database services requires no changes in the application code. For example, the Oracle E-Business Suite defines a database service for each responsibility, such as general ledger, accounts receivable, order entry,

[12] Refer to GDS Whitepaper @ www.oracle.com/technetwork/database/availability/global-data-services-12c-wp-1964780.pdf

and so on.
jdbc:oracle:thin:@(DESCRIPTION=(CONNECT_TIMEOUT=15)
(ADDRESS=(PROTOCOL=tcp)(HOST=myhost-vip)(PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=myorcldbservicename))
SERVICE_NAME: Use a database service name.

Recommendation 15: **Use UCP and enable run time load balancing.**

Enable run time load balancing on the service side for RAC and GDS. Refer to the section **Run Time Load Balancing** in the whitepaper[13] for more details.
Set 'Runtime Load Balancing Goal' to SERVICE_TIME or THROUGHPUT and
Set 'Connection Load Balancing Goal' to SHORT on the server side for both RAC and GDS.

```
$srvctl modify service -db <db_name> -service <service_name> -clbgoal
SHORT
$srvctl modify service -db <db_name> -service <service_name> -rlbgoal
SERVICE_TIME
```

```
$gdsctl modify service -db <db_name> -service <service_name> -rlbgoal
SERVICE_TIME
$gdsctl modify service -db <db_name> -service <service_name> -clbgoal
SHORT
```

Recommendation 16: **Use Shared Server**

Oracle database server process can either be dedicated (default), shared server (a.k.a. MTS) or pooled server (a.k.a. DRCP).
In a shared server architecture, a dispatcher directs multiple incoming network session requests to a pool of shared server processes, eliminating the need for a dedicated server process for each connection thus reducing system resources when supporting an increased number of users.

Shared Server would be recommended for any database configurations to achieve scalability for applications with a medium number of connections. Dedicated sessions are recommended for applications with a small number of connections.

The downside of shared server is an extra hop through the dispatcher, a restriction with some database features support, and increased complexity for setup and tuning.

As a general guideline, use shared server only when there are more concurrent connections to the database than the operating system can handle. SHARED_SERVERS and MAX_SHARED_SERVERS initialization parameters can be used to control the number of shared servers created. Use the following formula for computing the optimal number of connections for a database server.

*MaxConnections = (sum (rdbms-cores-per-instance) * n)*
where n is an integer with a typical recommended value of 9 or 10.

[13] Oracle 12c HA Concepts @ www.oracle.com/technetwork/database/application-development/12c-ha-concepts-2408080.pdf

Example: Consider a single node database server with 4 cores per node with n=10
MaxConnections = (4 * 10) = 40

Server Side configurations:

Shared Server must be configured and enabled by setting a few initialization parameters. Refer to Oracle Database Administrator's Guide[14] for detailed steps on enabling shared server on the database side.

Client Side configurations:

```
jdbc:oracle:thin:@
  (DESCRIPTION=
    (ADDRESS_LIST=
      (ADDRESS=(PROTOCOL=tcp)(HOST=proddbcluster)(PORT=5521)))
    (CONNECT_DATA=(SERVICE_NAME=proddb)(SERVER=SHARED)))
```

SERVER: Use this parameter to specify a particular service handler type. The connect descriptor uses (SERVER=SHARED) to request a dispatcher when connecting to a database indicating a shared server usage. (SERVER=DEDICATED) indicates a dedicated server which is the default behavior.

Recommendation 17: **Use Oracle Database Resident Connection Pool (DRCP)**

Database Resident Connection Pool (DRCP) is the *server side connection pool* consists of a set of "dedicated servers" managed by connection broker(s). A connection broker manages the "pooled servers" and shares these among multiple mid-tiers based on requests.

DRCP should be used in any database configurations to achieve better scalability in environments requiring a large number mid-tiers (or web servers) accessing the same database server and the number of active connections is fairly less than the number of open connections. For more details on DRCP refer to *JDBC Developer's guide*[15]*.*

The downside of DRCP is: it requires an additional hop to the connection broker for authentication and getting a connection from the pool however, once the connection is established it works exactly the same way as dedicated servers. DRCP also incurs more server side CPU and memory consumption. Use it only when database connections are to be shared across multiple middle-tier processes.

Server Side configurations:

`DBMS_CONNECTION_POOL` package is used for configuring and starting DRCP pool on the server side. Use `DBMS_CONNECTION_POOL.START_POOL()` and `DBMS_CONNECTION_POOL.STOP_POOL()` for starting and stopping a pool. Also, invoke `DBMS_CONNECTION_POOL.CONFIGURE_POOL(session_cached_cursors=>50)` to enable statement caching on the server side.

Client Side Configurations:

```
jdbc:oracle:thin:@
(DESCRIPTION=
  (ADDRESS_LIST=
```

[14] Oracle Database Administrator's Guide - http://docs.oracle.com/database/121/ADMIN/manproc.htm

[15] JDBC Developer's Guide https://docs.oracle.com/database/121/JJDBC/toc.htm

```
    (ADDRESS=(PROTOCOL=tcp)(HOST=proddbcluster)(PORT=5521)))
  (CONNECT_DATA=(SERVICE_NAME=proddb-service)(SERVER=POOLED)))
```

SERVER: Connection descriptor (SERVER=POOLED) is used to enable DRCP. Permitted values for this parameter are DEDICATED, SHARED, and POOLED.

Another code change required is to set a non-null or non-empty string value to the `oracle.jdbc.DRCPConnectionClass` connection property while getting a connection.

DRCP with UCP as client side connection pool

UCP helps in maintaining a liaison or attachment to connection broker. The client-side connection pools must attach and detach connections to the connection broker through `attachServerConnection()` and `detachServerConnection()`. The benefit of using UCP over third party client pool is that, UCP transparently takes care of attaching and detaching server connections. Refer to **UCPWithDRCPSample.java** in connection management code samples.

DRCP without UCP

DRCP can be used with any third party client-side connection pools. Third party client side connection pools must attach and detach connections explicitly to the connection broker through `attachServerConnection()` and `detachServerConnection()` methods. Connection class should also be set as shown in the sample. Refer to **DRCPSample.java** in connection management code samples.

```
OracleDataSource ods = new OracleDataSource();
// DRCP Property: Connection Class
// Set the connection-class to share connections across pool
Properties connproperty = new Properties();
connproperty.setProperty("oracle.jdbc.DRCPConnectionClass","DRCP_conn_class");
ods.setConnectionProperties(connproperty);
try (OracleConnection connection = (OracleConnection) (ods.getConnection())) {
  System.out.println("DRCP enabled: " + connection.isDRCPEnabled());
  connection.attachServerConnection();
  doSQLWork(connection);
  connection.detachServerConnection((String)null);
}
```

2.2 Real Application Clusters (RAC)

All connection management strategies discussed under “Single Instance/Single Database” are applicable to Real Application Clusters (RAC). In addition to that make sure to have scalability/load balancing descriptors in the connection URL.

Recommendation 18: **Use connection URL for RAC**

```
jdbc:oracle:thin:@
(DESCRIPTION=
  (CONNECT_TIMEOUT=15)(RETRY_COUNT=20) (RETRY_DELAY=3)
    (ADDRESS_LIST =
      (LOAD_BALANCE=ON)
        (ADDRESS=(PROTOCOL=tcp)(HOST=primaryscan)(PORT=1521)))
        (CONNECT_DATA=(SERVICE_NAME=myorcldbservicename)))
```

ADDRESS_LIST: If there is only one address then ADDRESS_LIST is not necessary. In case of RAC, ADDRESS_LIST provides a way to specify LOAD_BALANCE and FAILOVER connection descriptors for each ADDRESS. It enables connection time load balancing across the list of sites specified in SCAN or a list of host names spelled out within an ADDRESS.
LOAD_BALANCE: This parameter enables connection time load balancing thus enabling listeners to make routing decisions based on the load on the nodes. When this parameter is set to OFF, Oracle Net tries the protocol addresses sequentially until one succeeds.

When this parameter is set to ON for a SCAN based address, new connection requests will be randomly assigned to one of the 3 SCAN-based IP addresses resolved by DNS. This randomization enables all listeners to share the job of servicing incoming connect requests. For clients without SCAN, Oracle Net services randomly selects an address in the address list and connects to the listener on that node.

2.3 Multitenant Database

All connection management strategies discussed under “Single Instance/Single Database” are applicable to Multitenant Database.

2.4 Data Guard or Active Data Guard

All connection management strategies discussed under “Single Instance/Single Database” are applicable to Data Guard or Active Data Guard. In addition to that make sure to have scalability/load balancing descriptors in the connection URL.

Recommendation 19: **Connection URL for DG or ADG**

```
jdbc:oracle:thin:@(DESCRIPTION = (FAILOVER=on) (LOAD_BALANCE=off)
     (CONNECT_TIMEOUT= 15) (RETRY_COUNT=20) (RETRY_DELAY=3)
       (ADDRESS_LIST =
           (ADDRESS = (PROTOCOL = TCP) (HOST=primary-vip) (PORT=1521)))
      (ADDRESS_LIST =
             (ADDRESS = (PROTOCOL = TCP) (HOST=secondary-vip) (PORT=1521)))
      (CONNECT_DATA= (SERVICE_NAME = gold-cloud-service-name)))
```

LOAD_BALANCE (Connect time Load balancing): When set to ON, this parameter instructs Oracle Net to progress through the list of protocol addresses in a random sequence, balancing the load on the various host name addresses. When set to OFF, it instructs Oracle Net to try the address sequentially until one succeeds. It is turned OFF for single instances.

2.5 DG or ADG with RAC

All connection management strategies discussed under “Single Instance/Single Database” are applicable to DG or ADG with RAC. In addition to that make sure to have scalability/load balancing descriptors in the connection URL.

Recommendation 20: **Connection URL for DG or ADG with RAC**

```
jdbc:oracle:thin:@(DESCRIPTION = (FAILOVER=on)
     (CONNECT_TIMEOUT= 15) (RETRY_COUNT=20) (RETRY_DELAY=3)
       (ADDRESS_LIST =
           (LOAD_BALANCE=on)
```

```
          (ADDRESS = (PROTOCOL = TCP) (HOST=primary-scan) (PORT=1521)))
     (ADDRESS_LIST =
            (LOAD_BALANCE=on)
            (ADDRESS = (PROTOCOL = TCP) (HOST=secondary-scan) (PORT=1521)))
     (CONNECT_DATA= (SERVICE_NAME = gold-cloud-service-name)))
```

HOST: Always use SCAN name
LOAD_BALANCE (Connect time Load balancing): Refer to "Recommendation 19: Connection URL for DG or ADG" for more details.

2.6 Global Data Services (GDS)

All connection management strategies discussed under "Single Instance/Single Database" are applicable to GDS. In addition to that make sure to have scalability/load balancing descriptors in the connection URL.

Recommendation 21: **Connection URL for GDS**

```
jdbc:oracle:thin:@(DESCRIPTION= (FAILOVER=on)
     (CONNECT_TIMEOUT=15) (RETRY_COUNT=20) (RETRY_DELAY=3)
     (ADDRESS_LIST=
        (LOAD_BALANCE=ON)
        (ADDRESS= (PROTOCOL= TCP) (HOST=myorclgsm1) (PORT=1571)))
      (ADDRESS_LIST=
         (LOAD_BALANCE=ON)
         (ADDRESS= (PROTOCOL= TCP) (HOST=myorclgsm2) (PORT=1572)))
     (CONNECT_DATA= (SERVICE_NAME=gold-cloud-service-name) (REGION=region1))
```

HOST: Always use a Global Service Manager (GSM) listener which is analogous to a SCAN listener for a RAC database. GSMs are "Global Listeners" that are instrumental in performing inter-database service failovers and load balancing of GDS.
REGION: This is another important connection descriptor for GDS that specifies the region. The region name is required because run time load balancing advisory is customized for particular region in GDS. Examples of a GDS Regions are Asia region or Europe region etc.

Connection Management Strategies for **High Availability (HA)**

Applications should be designed to mask downtime (time that a resource is unavailable) from end users. Downtime can be planned or unplanned. The primary characteristics of high availability are reliability, recoverability, timely error detection, and continuous operations. Oracle Database 12c provides integrated HA technologies to reduce or avoid unplanned downtime, enable rapid recovery from failures, and minimize planned downtimes. Oracle Real Application Cluster (Oracle RAC), Oracle Data Guard (DG), Oracle Active Data Guard (ADG), and Application Continuity (AC) are the key components that enable high availability. This section focuses on strategies for achieving high availability for each database configuration.

3.1 Single Instance Database

Single Instance Database do not provide any high availability capabilities. Therefore, our recommendation is to convert a single instance database to either Oracle RAC or Oracle RAC ONE Node database to take advantage of the high availability capabilities. Refer to "*3.2 RAC ONE*

Node" and "*3.3 Real Application Clusters (RAC)*" for more details.

3.2 RAC ONE Node

Oracle Real Application Clusters One Node (Oracle RAC One Node) is a single instance of an Oracle Real Application Clusters (Oracle RAC) database that runs on one node in a cluster. RAC ONE benefits from the same infrastructure used for Oracle RAC.

The online database relocation feature ensures service availability by relocating an Oracle RAC One Node to another node during maintenance. The second instance of an Oracle RAC One Node database is created only during a planned online database relocation. On the start of planned maintenance, when the primary instance is down, move all services to the second instance through service relocation (`srvctl relocate service`), wait for all the connections to migrate to the relocated instance, and shut down the primary database instance forcing any remaining connections to move to the relocated instance.

Customers who need sub-minute recovery should deploy their databases on multi-node Oracle Real Application Clusters (RAC). Oracle RAC provides the best possible availability and fastest recovery from failures.

Hiding planned maintenance

Planned maintenance is required for regular maintenance of the technology infrastructure, software upgrades/patching, or for hardware maintenance. It is important to design a system to minimize planned interruptions.
The connection management strategy recommended to hide planned downtime is to drain the workload away from the server schedule for maintenance. Enabling `FastConnectionFailover (FCF)` a UCP property handles the draining.

Recommendation 22: Enable FAN events and Draining

Fast Application Notification (FAN) events such as DOWN, Planned DOWN, UP, RLB% etc., are notified to subscribers (database drivers, containers) in a fast and reliable manner using the Oracle Notification System (ONS). `FastConnectionFailover (FCF)` is a mechanism through which UCP subscribes to FAN events and helps in rapid failover of connections or rebalance of workload based on the event. The "UCP configuration in Apache Tomcat" is as shown below. An application using UCP with FCF, handles the planned maintenance gracefully; DBAs and developers should follow the steps mentioned in the whitepaper[16] for achieving a smooth and transparent planned maintenance.

[16] UCP with Tomcat whitepaper @ www.oracle.com/technetwork/database/application-development/planned-unplanned-rlb-ucp-tomcat-2265175.pdf

```
<Context docBase="UCPTomcat" path="/UCPTomcat“
   reloadable="true" source="org.eclipse.jst.jee.server:UCPTomcat">
<Resource name="tomcat/UCPPool" auth="Container“
   factory="oracle.ucp.jdbc.PoolDataSourceImpl"
   type="oracle.ucp.jdbc.PoolDataSource“
   description="UCP Pool in Tomcat"
   connectionFactoryClassName="oracle.jdbc.pool.OracleDataSource“
   minPoolSize="5"    maxPoolSize="50"    initialPoolSize="15“   autoCommit="true“
   user="scott"    password="tiger"
   fastConnectionFailoverEnabled="true“
   url
="jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=tcp)(HOST=lo
calhost)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=myorcldbservicename)))"
/>
```

Hiding unplanned downtime

Oracle database provides high availability solutions that prevent, tolerate, and reduce downtime for all types of unplanned failures. Unplanned downtime can be caused by site failures, computer failures, storage failures, data corruption, or human errors.

The connection management strategy recommended to hide unplanned outages for any database configurations is to use Application Continuity (AC). Application Continuity (AC) which internally invokes Transaction Guard (TG) attempts to mask unplanned outages from end users.

Recommendation 23: **Use Transaction Guard (TG)**

Transaction Guard (TG) provides an API for ensuring at-most-once COMMIT execution. TG furnishes a reliable commit outcome that indicates whether a transaction is committed and completed or not. Every transaction is tagged with a Logical Transaction Identifier (LTXID), which can be used by an application after a COMMIT failure to verify whether the in-flight transaction had committed before the failure or not. Transaction Guard prevents applications from submitting duplicate transactions. Refer to the whitepaper[17] for application changes and server side changes to enable TG.

Recommendation 24: **Enable Application Continuity (AC)**

Application Continuity (AC) improves the end user experience by masking many unplanned outages. It attempts to hide downtimes by replaying in-flight application requests during planned and unplanned outages. A request is a unit of work from the application that corresponds to DML statements and other database calls of a single web request.

Application Continuity[18] needs to be enabled on the server side and also on the client side.

Java Applications require the following configurations for Application Continuity:

- Oracle Database 12c Release 1 (12.1)

[17] Oracle Database 12c HA Building Blocks @ www.oracle.com/technetwork/database/application-development/12c-ha-concepts-2408080.pdf

[18] AC Whitepaper @ http://www.oracle.com/technetwork/database/database-cloud/private/application-continuity-wp-12c-1966213.pdf

- Java applications to use Oracle Universal Connection Pool with the Oracle replay data source (**`oracle.jdbc.replay.OracleDataSourceImpl)`** and also, subscribe to FAN events through `setFastConnectionFailover(true)` in UCP.
- Oracle Real Application Clusters (Oracle RAC) or Oracle Data Guard should ensure that FAN is configured with Oracle Notification System (ONS). Note that ONS is configured by default in RAC and DG.
- Use an application service; do not use the default database service that is reserved for Oracle Enterprise Manager and DBAs.
- Set the required properties on the service for replay and load balancing as shown below.
  ```
  $srvctl modify service -db <db_name> -service <service_name>  -
  failovertype TRANSACTION  -replay_init_time 600 -failoverretry 30 -
  failoverdelay 10 -commit_outcome TRUE
  ```
- **`ojdbc7.jar, ucp.jar and ons.jar`** should be in the class path. Make sure that all these jars are from the same database version i.e., 12.1.0.2
- Replace Oracle JDBC concrete classes[19] with standard JDBC interfaces: AC does not support concrete classes and these must be replaced before using AC.
- Disable replay to avoid any side effects. Replaying transactions related to mailing, printing checks etc., may cause side effects hence, you must disable replay for such critical sections using the `disableReplay()` method of the `ReplayableConnection` interface.

3.3 Real Application Clusters (RAC)

All connection management strategies discussed under "3.2: RAC ONE Node" are applicable to Real Application Clusters (RAC). In addition, make sure to have high availability descriptors in the connection URL.

Recommendation 25: **Connection URL for RAC**

```
jdbc:oracle:thin:@
(DESCRIPTION=
  (CONNECT_TIMEOUT=15)(RETRY_COUNT=20) (RETRY_DELAY=3)
       (ADDRESS_LIST =
           (LOAD_BALANCE=ON)
               (ADDRESS=(PROTOCOL=tcp)(HOST=primaryscan)(PORT=1521)))
  (CONNECT_DATA=(SERVICE_NAME=myorcldbservicename)))
```

Connection URL: Some points to remember.
(a) Never use Easy Connection URL aka EZConnect (Example: jdbc:oracle:thin:@//localhost:1521/myorcldbservicename) for connecting to a RAC. It does not provide a way to specify connection descriptors and thus cannot take advantage of high availability (HA) and load balancing, et al.
(b) DO NOT use syntax from old versions. Always use the latest version connection URL syntax as shown above.
HOST: The recommendation is to always use SCAN and not use a database host name as it will not offer high availability capabilities.

3.4 Data Guard or Active Data Guard

[19] Refer to MOS Note "New Jdbc Interfaces for Oracle types (Doc ID 1364193.1)"

All connection management strategies discussed under “3.2: RAC ONE Node” are applicable to DG or ADG. In addition, make sure to have high availability descriptors in the connection URL.

Recommendation 26: **Connection URL for DG or ADG**

```
jdbc:oracle:thin:@(DESCRIPTION = (FAILOVER=on) (LOAD_BALANCE=off)
     (CONNECT_TIMEOUT= 15) (RETRY_COUNT=20) (RETRY_DELAY=3)
       (ADDRESS_LIST =
           (ADDRESS = (PROTOCOL = TCP) (HOST=primary-vip) (PORT=1521)))
      (ADDRESS_LIST =
             (ADDRESS = (PROTOCOL = TCP) (HOST=secondary-vip) (PORT=1521)))
      (CONNECT_DATA= (SERVICE_NAME = gold-cloud-service-name)))
```

FAILOVER: It enables connect time failover. When turned ON, it instructs Oracle Net to fail over to a different listener if the first listener fails. The number of addresses in the list determines how many addresses are tried. When set to OFF, it instructs Oracle Net to try only one address. Always set FAILVOER=ON for ADG or DG database configuration.

3.5 DG or ADG with RAC

All connection management strategies discussed under “3.2: RAC ONE Node” are applicable to DG or ADG with RAC. In addition to that make sure to have high availability descriptors in the connection URL.

Recommendation 27: **Connection URL for DG or ADG with RAC**

```
jdbc:oracle:thin:@(DESCRIPTION = (FAILOVER=on)
     (CONNECT_TIMEOUT= 15) (RETRY_COUNT=20) (RETRY_DELAY=3)
       (ADDRESS_LIST =
           (LOAD_BALANCE=on)
           (ADDRESS = (PROTOCOL = TCP) (HOST=primary-scan) (PORT=1521)))
      (ADDRESS_LIST =
             (LOAD_BALANCE=on)
             (ADDRESS = (PROTOCOL = TCP) (HOST=secondary-scan) (PORT=1521)))
      (CONNECT_DATA= (SERVICE_NAME = gold-cloud-service-name)))
```

FAILOVER: Refer to FAILOVER description under “Recommendation 26: Connection URL for DG or ADG”.

3.6 Global Data Services (GDS)

All connection management strategies discussed under “3.2: RAC ONE Node“ are applicable to GDS. In addition to that make sure to have high availability descriptors in the connection URL.

Recommendation 28: **Connection URL for GDS**

```
jdbc:oracle:thin:@(DESCRIPTION= (FAILOVER=on)
     (CONNECT_TIMEOUT=15) (RETRY_COUNT=20) (RETRY_DELAY=3)
     (ADDRESS_LIST=
        (LOAD_BALANCE=ON)
        (ADDRESS= (PROTOCOL= TCP) (HOST=myorclgsm1) (PORT=1571)))
      (ADDRESS_LIST=
         (LOAD_BALANCE=ON)
         (ADDRESS= (PROTOCOL= TCP) (HOST=myorclgsm2) (PORT=1572)))
     (CONNECT_DATA= (SERVICE_NAME=gold-cloud-service-name) (REGION=region1))
```

FAILOVER: Refer to FAILOVER description under "Recommendation 26: Connection URL for DG or ADG".

Connection Management Strategy for **Security**

Oracle database provides a rich set of security features that can be used to manage user accounts, authentication, privileges and roles, application security, encryption, network traffic, and auditing.

The connection management strategy is to choose any one of the following security features based on the security requirements/restrictions of the business. These security features are applicable to any database configurations such as Single Instance Database, Oracle Real Application Clusters (RAC), Oracle Multitenant, Oracle Data Guard (DG), Oracle Active Data Guard (ADG), and Oracle Global Data Services (GDS).

- **Advanced Security**: Includes Data Encryption and Data Integrity
- **Strong Authentication** methods such as SSL, Kerberos and RADIUS
- **Proxy Authentication**

Recommendation 29: **Consider Advanced Security**

The Oracle JDBC drivers embed the classes that implement advance security APIs. The security parameters for encryption and integrity are usually set in the `sqlnet.ora` file; these can also be set as connection properties or system properties.

Data Encryption and Data Integrity

Sensitive information communicated over enterprise networks and internet can be protected by using encryption algorithms, which transform information into a form that can be deciphered only with a decryption key. Some of the supported encryption algorithms are RC4, DES, 3DES, and AES. Oracle Advanced Security uses the following hashing algorithms to generate the secure message digest and includes it with each message sent across a network. Refer to the connection level properties related to Data Encryption and Data Integrity in this section.

```
Properties connProps = new Properties();
// For Data Integrity Check
connProps.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES, "( MD5, SHA1, SHA256, SHA384 or SHA512 )");
connProps.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL, "REQUIRED");

// For Data Encryption
connProps.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL, REQUIRED");
connProps.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES,"(DES40C)");
// OracleDataSource – Oracle JDBC Connection
OracleDataSource ods = new OracleDataSource();
```

```
ods.setConnectionProperties(connProps);
// PoolDataSource – While Using UCP as connection pool
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
pds.setConnectionProperties(connProps);
```

Recommendation 30: **Use Strong Authentication**

For customers looking for a stronger authentication than the basic username/password, Oracle Advanced Security enables users to authenticate externally with RADIUS, Kerberos, Certificate-Based Authentication, Token Cards, and Smart Cards. Oracle JDBC driver supports the following strong authentications:

- SSL (certificate-based authentication)
- Kerberos
- RADIUS

SSL Authentication

Secure Sockets Layer (SSL) provides authentication, data encryption, and data integrity. SSL uses digital certificates that comply with the X.509v3 standard for authentication and a public and private key pair for encryption. SSL also uses secret key cryptography and digital signatures to ensure privacy and integrity of data. When a network connection over SSL is initiated, the client and server perform a SSL handshake. Refer to the whitepaper *“SSL with Oracle JDBC Thin Driver”*[20] for code samples and more details. Some of the connection level properties used in SSL authentication setup are as shown below. SSL properties can also be set as system level properties using the name included in parenthesis.

Name of the connection property	Detailed Description
CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES	To specify the authentication service to be used. Use SSL as the value.
CONNECTION_PROPERTY_THIN_SSL_CIPHER_SUITES (oracle.net.ssl_cipher_suites)	To enable a subset of the cipher suites available. Prioritize cipher suites starting with the strongest and moving to the weakest to ensure the highest level of security possible.
CONNECTION_PROPERTY_THIN_SSL_SERVER_DN_MATCH (oracle.net.ssl_server_dn_match)	To force the driver to verify if the server’s DN matches. Permitted values are ON, OFF, TRUE or FALSE. Sample connection URL looks like: jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcps)(HOST=servername)(PORT=2484))(CONNECT_DATA=(SERVICE_NAME=servicename))**(SECURITY=(SSL_SERVER_CERT_DN=\"CN=server_test, C=US\")**))

[20] SSL with Oracle JDBC Thin driver @ www.oracle.com/technetwork/database/enterprise-edition/wp-oracle-jdbc-thin-ssl-130128.pdf

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORETYPE (javax.net.ssl.keyStoreType)	To denote a valid type of the key store supported by SSL. Example. JKS, PKCS12, SSO etc.,
CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE (javax.net.ssl.keyStore)	To specify the location of a key store.
CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTOREPASSWORD (javax.net.ssl.keyStorePassword)	To specify a password for a key store, used to check the integrity of the data before accessing it.
CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORETYPE (javax.net.ssl.trustStoreType)	To denote a valid type of the trust store supported by SSL. Example. JKS, PKCS12, SSO etc.,
CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE (javax.net.ssl.trustStore)	To specify the location of a trust store.
CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTOREPASSWORD(javax.net.ssl.trustStorePassword)	To specify a password for a trust store used to check the integrity of the data before accessing it.
CONNECTION_PROPERTY_THIN_SSL_VERSION (oracle.net.ssl_version)	To specify the SSL version. e.g., 1.2 or 1.1 etc.,
CONNECTION_PROPERTY_WALLET_LOCATION (oracle.net.wallet_location)	To specify the oracle wallet location. The wallet contains certificates and keys used by SSL. listener.ora or tnsnames.ora file contains the following. WALLET_LOCATION=(SOURCE=(METHOD=FILE)(METHOD_DATA=(DIRECTORY=/server/wallet/path))) SSL_CLIENT_AUTHENTICATION=TRUE
CONNECTION_PROPERTY_WALLET_PASSWORD	To specify the wallet password which is only required if auto-login is not enabled in the wallet.

Kerberos Authentication

Kerberos protocol uses strong cryptography so that a client or a server can prove its identity to its server or client across an insecure network connection. The Kerberos architecture is centered on a trusted authentication service called the Key Distribution Center (KDC). The users and services in a Kerberos environment are referred to as principals; each principal shares a secret (e.g., password) with KDC. A principal can be a user such as HR or a database server instance. Kerberos requires an initial KDC setup. Refer to *JDBC Developer's Guide*[21] for the sample code on Kerberos. Some of the connection level properties related to Kerberos setup are:

Name of the Connection property	**Detailed Description**

[21] JDBC Developer's Guide http://docs.oracle.com/database/121/JJUAR/toc.htm

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES	To specify the authentication service to be used. Use KERBEROS.
CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_CC_NAME	To specify the location of the Kerberos credential cache.
CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_MUTUAL	To turn on Kerberos mutual authentication, set this property to "true".

RADIUS Authentication

Remote Authentication Dial-In User Service (RADIUS) is a client/server security protocol that is most widely known for enabling remote authentication and access. RADIUS can be used with a variety of authentication mechanisms, including token cards and smart cards. Refer to *JDBC Developer's Guide*[22] for the sample code on RADIUS. Connection level property related to RADIUS setup is mentioned here.

Name of the Connection property	**Detailed Description**
CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES	To specify the authentication service to be used. Use RADIUS.

Recommendation 31: **Consider Proxy Authentication**

Proxy authentication[19] also called N-tier authentication uses middle tier for user authentication. A middle tier server can be designed to proxy clients in a secure fashion in any three forms (USERNAME, DISTINGUISHED NAME, and CERTIFICATE) of proxy authentication. In simple words, proxy authentication enables one JDBC connection to act as a proxy for other JDBC connections.

Example: In case of “USER NAME -Proxy Authentication”, if “appuser” is the proxy user name then “jeff” and “smith” are the proxied users created with specific roles as shown below. “jeff” and “smith” can connect to the database through “appuser”, where “appuser” is doing the authentication for them.

```
Rem Connect as system user to grant necessary roles to the DB users "jeff" and
"smith"
connect system/manager
grant select_role, insert_role, delete_role to jeff;
grant select_role, insert_role, delete_role to smith;

Rem grant the users "jeff" and "smith" to connect through “appuser” with
specified roles
alter user jeff grant connect through appuser with role select_role,
insert_role;
alter user smith grant connect through appuser with role select_role,
insert_role;
```

[22] JDBC Developer's Guide http://docs.oracle.com/database/121/JJUAR/toc.htm

Some of the connection level properties that are used in Proxy Authentication setup are:

Name of the Connection property	Detailed Description
PROXY_CERTIFICATE	Used for specifying the proxy certificate.
PROXY_DISTINGUISHED_NAME	Used for specifying the distinguished name of the user
PROXY_ROLES	Used for specifying the roles that the proxy will be granted access to.
PROXY_USER_NAME	Used for specifying the user name.
PROXY_USER_PASSWORD	Used for specifying the user password and should be used in conjunction with PROXY_USER_NAME.

Refer to **ProxySessionSample.sql** and **ProxySessionSample.java** in connection management code samples.

Connection Management Strategy for **Manageability**

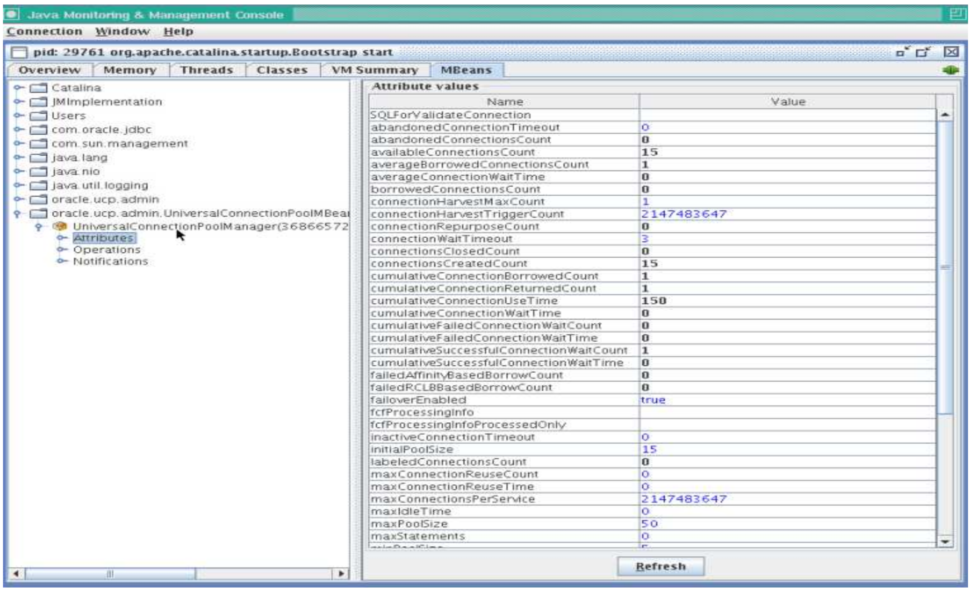
Manageability in Java applications consists of using appropriate monitoring tools to monitor the health of applications with respect to connections and best practices that will ease maintaining and managing database servers.

4.1 Single Instance Database

Recommendation 32: **Use JMX based management tools**

JMX (Java Management Extensions) is used to monitor and manage resources as they are created, installed, and implemented. In JMX, a given resource is instrumented by one or more Java objects known as MBeans (Managed Beans). Universal Connection Pool (UCP) can be managed through MBeans as well. The **`UniversalConnectionPoolManagerMBean`** is a manager MBean that includes all the functionalities of a conventional connection pool manager. The **`UniversalConnectionPoolMBean`** is a pool MBean that covers dynamic configuration of pool properties and pool statistics.

When you start JConsole (or MBeans enabled tool), it lists all processes to choose from, pick the process id (pid) of the application or program that is using UCP. A sample JConsole screenshot of the MBean is shown here.



Name	Value
SQLForValidateConnection	
abandonedConnectionTimeout	0
abandonedConnectionsCount	0
availableConnectionsCount	15
averageBorrowedConnectionsCount	1
averageConnectionWaitTime	0
borrowedConnectionsCount	0
connectionHarvestMaxCount	1
connectionHarvestTriggerCount	2147483647
connectionRepurposeCount	0
connectionWaitTimeout	3
connectionsClosedCount	0
connectionsCreatedCount	15
cumulativeConnectionBorrowedCount	1
cumulativeConnectionReturnedCount	1
cumulativeConnectionUseTime	150
cumulativeConnectionWaitTime	0
cumulativeFailedConnectionWaitCount	0
cumulativeFailedConnectionWaitTime	0
cumulativeSuccessfulConnectionWaitCount	1
cumulativeSuccessfulConnectionWaitTime	0
failedAffinityBasedBorrowCount	0
failedRCLBBasedBorrowCount	0
failoverEnabled	true
fcfProcessingInfo	
fcfProcessingInfoProcessedOnly	
inactiveConnectionTimeout	0
initialPoolSize	15
labeledConnectionsCount	0
maxConnectionReuseCount	0
maxConnectionReuseTime	0
maxConnectionsPerService	2147483647
maxIdleTime	0
maxPoolSize	50
maxStatements	0

4.2 Real Application Clusters (RAC)

All connection management strategies discussed under “4.1: Single Instance Database” are applicable to Real Application Clusters (RAC). In addition to that make sure to use SCAN as described here.

Recommendation 33: **Use Single Client Access Name (SCAN)**

SCAN is an Oracle RAC feature that provides a single name for clients to access the Oracle database cluster. SCAN does not change throughout the life of the cluster thus providing better manageability. Oracle recommends that all connections to the Oracle RAC database use SCAN in their client connection string.

Digging deeper, SCAN is defined either in Domain Name Server (DNS) or Grid Naming Service (GNS) that always resolves to three IP addresses. Example, if there are 10 instances then, SCAN setup leads to an entry in */etc/hosts* with 3 SCAN IP addresses. When a connection is requested, Net services round robins to pick a SCAN listener from these 3 IP addresses. Later, SCAN (remote) listener will choose any one of the10 instances (local listeners) from the least-loaded instance where the service is available.

```
# Scan IP's – contents of /etc/hosts
10.20.30.40 primaryscan.us.oracle.com primaryscan
10.20.30.41 primaryscan.us.oracle.com primaryscan
10.20.30.42 primaryscan.us.oracle.com primaryscan
```

Connection String without SCAN:
In this connection string, there are four hosts/nodes, each host has its own domain name mentioned explicitly in the URL. Imagine, if Oracle RAC has 10 instances then, the connection string will have 10 hostnames that can be error prone and tedious to manage.

```
jdbc:oracle:thin:@
(DESCRIPTION =
     (CONNECT_TIMEOUT= 15) (RETRY_COUNT=20) (RETRY_DELAY=3)
       (ADDRESS_LIST =
           (LOAD_BALANCE=ON)
              (ADDRESS = (PROTOCOL = tcp)(HOST = myhost-a-vip)(PORT = 1521))
              (ADDRESS = (PROTOCOL = tcp)(HOST = myhost-b-vip)(PORT = 1521))
              (ADDRESS = (PROTOCOL = tcp)(HOST = myhost-c-vip)(PORT = 1521))
              (ADDRESS = (PROTOCOL = tcp)(HOST = myhost-d-vip)(PORT = 1521)))
        (CONNECT_DATA=(SERVICE_NAME= myorcldbservicename)))
```

Connection String with SCAN:
SCAN simplifies the client connection string and makes it more manageable. The connection string does not change even when the nodes/instances are added or removed from the cluster. If Oracle RAC has 10 instances then, SCAN name makes the connection string simple and manageable.

```
jdbc:oracle:thin:@
(DESCRIPTION=
  (CONNECT_TIMEOUT=15)(RETRY_COUNT=20)(RETRY_DELAY=3)
       (ADDRESS_LIST =
           (LOAD_BALANCE=ON)
               (ADDRESS=(PROTOCOL=tcp)(HOST=primaryscan)(PORT=1521)))
  (CONNECT_DATA=(SERVICE_NAME=myorcldbservicename)))
```

4.3 DG or ADG

All connection management strategies discussed under “4.1 Single Instance Database” are applicable to DG or ADG.

4.4 DG or ADG with RAC

All connection management strategies discussed under “4.1 Single Instance Database” and “4.2 Real Application Clusters (RAC)” are applicable to DG or ADG with RAC.

4.5 Global Data Services (GDS)

All connection management strategies discussed under “4.1: Single Instance Database’’ are applicable to Global Data Services (GDS). In addition to that make sure to use GSM listeners as described here.

Recommendation 34: **Use Global Service Managers (GSM) listeners**

Global Service Managers (GSM) listener in Oracle Global Data Services (GDS) is analogous to SCAN in an Oracle RAC database. A GDS configuration contains multiple Global Service Managers (GSM) per region. The GSMs are “Global Listeners” which understand real-time load characteristics and the user-defined service placement policies on the replicated databases.
Connection String with GSM Listeners:

```
jdbc:oracle:thin:@(DESCRIPTION= (FAILOVER=on)
     (CONNECT_TIMEOUT=15) (RETRY_COUNT=20) (RETRY_DELAY=3)
     (ADDRESS_LIST=
        (LOAD_BALANCE=ON)
        (ADDRESS= (PROTOCOL= TCP) (HOST=myorclgsm1) (PORT=1571)))
      (ADDRESS_LIST=
         (LOAD_BALANCE=ON)
         (ADDRESS= (PROTOCOL= TCP) (HOST=myorclgsm2) (PORT=1572)))
     (CONNECT_DATA= (SERVICE_NAME=gold-cloud-service-name) (REGION=region1))
```

Conclusion

This white paper furnishes a comprehensive and practical coverage of connection management strategies while using Oracle JDBC and UCP with Oracle database 12c. The paper discusses best practices, configurations, and properties to achieve performance, availability, scalability, security, and manageability with various database configurations such as Single Instance Database, Oracle Real Application Clusters (RAC), Oracle Multitenant, Oracle Data Guard (DG), Oracle Active Data Guard (ADG), and Oracle Global Data Services (GDS). Oracle Database Administrators, Java Architects, and web application designers can leverage the strategies from this whitepaper to design robust, reliable, high performant, highly scalable, highly available, secure and manageable web applications for better user experience. The complete connection management code samples referenced in this paper will be posted on OTN[23] and also on GITHUB (https://github.com/oracle/jdbc-ucp).

[23] Code samples are not uploaded and will be uploaded soon on OTN @ http://www.oracle.com/technetwork/database/features/jdbc/default-2345085.html

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

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

Integrated Cloud Applications & Platform Services

Oracle is committed to developing practices and products that help protect the environment