

---

**ORACLE®**

ORACLE  
**OPEN**  
WORLD

experience

**OPENWORLD**

November 11–15, 2007

ORACLE®

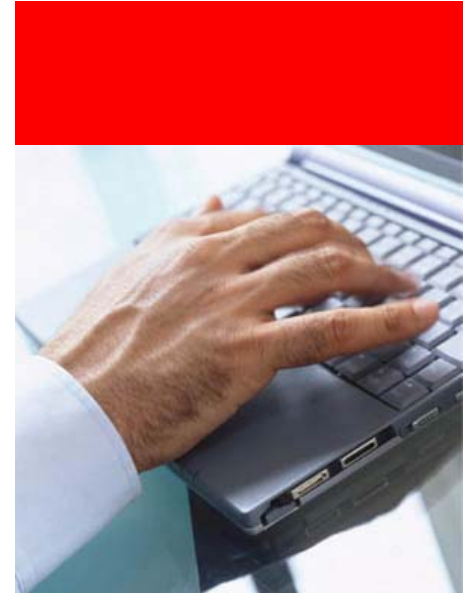


# Oracle Database 11g Release 1 JDBC Pearls

Douglas Surber  
Jean de Lavarene

# Agenda

- Introduction to 11g JDBC
- JDBC 4.0
- Advanced Security
- Advanced Queuing
- Database Change Notification
- Startup/Shutdown



# Oracle 11g JDBC Pop Quiz





# Question 1

What is wrong with this line of code?

```
import oracle.jdbc.driver.OracleStatement;
```

- A. It will not compile.
- B. It will not execute.
- C. It has been deprecated for over seven years.
- D. All of the above.**



## Question 2

What is the correct replacement for this line of code?

```
import oracle.jdbc.driver.OracleStatement;
```

- A. `import oracle.jdbc.driver.Statement;`
- B. `import oracle.jdbc.OracleStatement;`**
- C. `import oracle.jdbc.driver.*;`
- D. `import java.sql.Statement;`



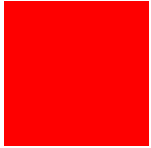
# Question 3

Which of the following are not supported in 11g JDBC?

- A. J2SE 1.2
- B. J2SE 1.3
- C. J2SE 1.4
- D. OracleConnectionCacheImpl
- E. All of the above

J2SE 5 and JSE 6 are supported

Oracle Database 9*i* and later are supported



# JDBC 4.0





# JDBC 4.0

## Supported Environments

- JDBC 4.0 is supported only in Java SE 6
  - ojdbc6.jar
  - ojdbc6\_g.jar
- JDBC 4.0 features are not available in Java SE 5
  - ojdbc5.jar
  - ojdbc5\_g.jar



# JDBC 4.0

## Drivers Load Automatically

- JSE 6 uses the Service Provider mechanism to load JDBC drivers

```
Connection c = DriverManager.getConnection(url);
```

- No need to load the driver

```
Class.forName("oracle.jdbc.OracleDriver");
```



# JDBC 4.0

## ROWID Support

- Standard support
  - `java.sql.RowId`
  - `getRowId(int)`
  - `setRowId(int, RowId)`
- Identical to Oracle proprietary types, methods
  - `oracle.sql.ROWID`
  - `getROWID(int)`
  - `setROWID(int, ROWID)`



# JDBC 4.0

## National Character Set Support

- New methods for Strings and Streams
  - `getNString(int)`
  - `setNString(int, String)`
  - `getNCharacterStream(int)`
  - `setNCharacterStream(int, Reader)`
- New NClob type and methods
  - `java.sql.NClob` is a subinterface of `java.sql.Clob`
  - `getNClob(int)`
  - `setNClob(int, NClob)`
- NClob factory method
  - `conn.createNClob()`
- No more `FormOfUse!`



# JDBC 4.0

## Factory Methods on `java.sql.Connection`

- New factory methods on `Connection`
  - `createArrayOf` -- not supported, but ...
  - `createBlob`
  - `createClob`
  - `createNClob`
  - `createSQLXML` -- not supported
  - `createStruct`
- Oracle method for creating `java.sql.Array`
  - `createArray(String)`
  - argument is name of array type
  - argument to `createArrayOf` is name of element type
  - Oracle does not support anonymous array types



# JDBC 4.0

## SQL XML Type Support

- JDBC 4.0 spec includes support for the SQL XML type, `java.sql.SQLXML`
- Oracle 11gR1 JDBC does not **yet** support SQLXML
- Use `oracle.xdb.XMLType` instead



# JDBC 4.0

## Wrapper Pattern

- Given a Tomcat JDBC connection object, how do you call `oracle.jdbc.OracleConnection.createArray`?
- JDBC 4.0 provides a standard way

```
Connection c = ds.getConnection();  
OracleConnection oc =  
    c.unwrap(OracleConnection.class);  
Array a = oc.createArray("MyArrayType");
```

- Requires JSE 6 support by container



# JDBC 4.0

## Wrapper Pattern for implementors (1/2)

```
OracleConnection oc =  
    oc.unwrap(OracleConnection.class);
```

- Argument to unwrap is an interface
- Return value must implement that interface
- It can be or even should be a proxy
- Proxies are required to be cached so multiple calls all get the same proxy object for a given interface

(cont.)



# JDBC 4.0

## Wrapper Pattern for Implementors

- (2/2) The proxy should
  - Call the wrapper object if the method exists there
    - This is not a way to avoid calling the wrapper object, just a way to access functionality the wrapper does not provide
  - Wrap return values
    - Be sure and cache wrappers
- The proxy can
  - Unwrap arguments
  - Forbid bad methods
  - Log calls
  - Check security constraints
  - Anything else you can think of



# JDBC 4.0

## SQLException Hierarchy

- SQLException
  - SQLTransientException
    - Still connected to the database
    - Try again and it might work
    - Timeout, rollback
  - SQLNonTransientException
    - Still connected to the database
    - Will fail even if you try again
    - Syntax error, no such table, bad password
  - SQLRecoverableException
    - Not connected to the database
    - Get a new connection, redo the transaction and it might work
    - Network failure, node down



# Advanced Security





# Advanced Security

## Encryption

- Need for security: Sensitive information that travels over enterprise networks and the Internet can be protected by encryption algorithms. An encryption algorithm transforms information into a form that can be deciphered with a decryption key.
- Supported Encryption Algorithms in 10.2: 3DES with 168-bit and 112-bit keys.
- New Encryption Algorithm in 11.1: AES (Advanced Encryption Standard) with 128-bit, 192-bit and 256-bit keys.

```
prop.setProperty(
```

```
OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES,
```

```
"( AES256, AES192, AES128, 3DES168, 3DES112 )");
```



# Advanced Security

## Checksumming or Data Integrity

- Need for checksumming: It ensures the integrity of data packets during transmission. It prevents the following attacks: Data modification, Deleted packets, Replay attacks.
- Supported checksumming Algorithm in 10.2: MD5. 16 byte hash of the message.
- New checksumming Algorithm in 11.1: SHA1. 20 byte hash of the message.

```
prop.setProperty(
```

```
    OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES,  
    "( SHA1, MD5 )");
```



# Advanced Security

## Authentication adaptors support in JDBC

Authentication: prove identity of the user.

- Password authentication (scott/tiger) using O5Logon.
- String Authentication through third-party authentication adaptors:
  - Kerberos (SSO and centralized authentication)

```
create user "BOB@US.ORACLE.COM" identified externally;  
prop.setProperty(  
OracleConnection.CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES,  
"( KERBEROS )");
```



# Advanced Security

## Authentication adaptors support in JDBC

- Radius (Remote Authentication Dial-In User Service)

```
create user BOB identified externally;  
  
prop.setProperty(  
    OracleConnection.CONNECTION_PROPERTY_THIN_NET_AUTHENTICATIO  
N_SERVICES, "( RADIUS )");
```

- SSL with digital certificates

```
create user BOB identified externally as  
    'CN=Bob,OU=ST,O=Oracle,ST=CA,C=US';  
  
prop.setProperty(  
    OracleConnection.CONNECTION_PROPERTY_THIN_NET_AUTHENTICATIO  
N_SERVICES, "( TCPS )");
```

# Advanced Queuing (AQ)





# Advanced Queuing

## Access queues from Java

- AQ concepts
  - a producer enqueues a message into a queue
  - a consumer dequeues the message from the queue
- New fast JDBC AQ APIs: native RPC (as opposed to PL/SQL)
- Enqueue and dequeue
- RAW, ANYDATA and ADT queues are supported
- Support for buffered queues (persistence vs. performance)
- Commit-time ordering
- Asynchronous AQ notification



# Advanced Queuing

## Overview of the new APIs

- On oracle.jdbc.OracleConnection:

```
AQMessage dequeue(String queueName,  
                  AQDequeueOptions opt,  
                  String typeName) throws SQLException
```

```
void enqueue(String queueName,  
             AQEnqueueOptions opt,  
             AQMessage msg) throws SQLException
```

- New oracle.jdbc.aq package: *AQAgent*, *AQMessage*,  
*AQMessageProperties*, *AQDequeueOptions*,  
*AQEnqueueOptions*, *AQFactory*



# Advanced Queuing

## Enqueue a RAW message

```
// Step1: create the message properties:
AQMessageProperties msgprop = AQFactory.createAQMessageProperties();
msgprop.setExceptionQueue("MY_EXCEPTION_QUEUE");

// Step2: Create the actual AQMessage instance:
AQMessage mesg = AQFactory.createAQMessage(msgprop);
// and add a payload:
byte[] rawPayload = { 0x01, 0x02, 0x03, 0x04 };
mesg.setPayload(new oracle.sql.RAW(rawPayload));

// Step3: set the enqueue options:
AQEnqueueOptions opt = new AQEnqueueOptions();
opt.setRetrieveMessageId(true);

// Step4: execute the actual enqueue operation:
conn.enqueue("SCOTT.MY_RAW_Q", opt, mesg);

byte[] mesgId = mesg.getMessageId();
```



# Advanced Queuing

## Dequeue a RAW message

```
// Step1: set the dequeue options:
AQDequeueOptions deqopt = new AQDequeueOptions();
deqopt.setRetrieveMessageId(true);

// Step2: dequeue operation:
AQMessage msg = conn.dequeue("SCOTT.MY_RAW_Q",
                             deqopt,
                             "RAW");

// Step3: retrieve the payload:
byte[] payload = msg.getPayload();
byte[] msgId = msg.getMessageId();
```



# Advanced Queuing

## Be notified of a new message

- Register your interest on a queue and be notified whenever a new message is enqueued
- No need to maintain any JDBC connection
- Notification grouping (notifications can be grouped and sent every 5 minutes)
- System queues or user defined queues
- For RAW queues, the event contains the payload. For other types of queues you need to dequeue the new message.



# Advanced Queuing

## AQ notification in a nutshell

- You need to implement the interface `oracle.jdbc.aq.AQNotificationListener`
- Get an instance of `AQNotificationRegistration` with the following method on `oracle.jdbc.OracleConnection`:

```
AQNotificationRegistration[] registerAQNotification(  
    String[] queueNames,  
    Properties[] options,  
    Properties globaloptions) throws SQLException
```

- Add an instance of your listener to the registration. `AQNotificationRegistration` has the following method:

```
void addListener(AQNotificationListener listener)
```

# Database Change Notification (DCN)





# Database Change Notification

## Principles

- Be notified when the result of your “select” query changes
- Build a data cache in the client tier: DCN provides cache invalidations
- It uses the AQ notification mechanism
- With a 10.2 database, you have table change notifications
- You get the ROWID of the rows that have changed. You must then query the database to refresh your cache



# Database Change Notification

## Steps

- Create a registration
- Add the Java listeners to the registration
- Associate a statement with the registration
- Execute a select query with the statement
- You will now be notified when the result of the query changes



# Database Change Notification

## Registration

- On the server, a registration identifies an end-point address (the JDBC driver)
- It keeps the list of SQL queries that are registered
- You can have multiple queries in a single registration
- You can have multiple registrations
- It's a persistent entity that is recorded in the database and visible to all instances of a RAC database
- At creation time, you can set options: ignore inserts or updates, purge after notification, etc.



# Database Change Notification

## Notifications

- Name of the tables that were modified
- Operation type (insert, update, delete, alter table, drop table)
- Rowid of the changed rows and the associated operation (insert, update, delete)
- Global database events (startup, shutdown)
- You DON'T get the data
- You can choose to make the notifications persistent (performance cost involved)



# Database Change Notification

**Implement listener interface (1/3)**

```
class DCNDemoListener implements DatabaseChangeListener
{
    public void
    onDatabaseChangeNotification(DatabaseChangeEvent e)
    {
        System.out.println("DCNDemoListener: got an event");
        System.out.println(e.toString());
    }
}
```



# Database Change Notification

## Create the registration (2/3)

```
Properties prop = new Properties();
// Ask the server to send the ROWIDs as part of the DCN events
prop.setProperty(OracleConnection.DCN_NOTIFY_ROWIDS, "true");

// The following operation does a roundtrip to the database to create
// a new registration for DCN. It sends the client address (ip address
// and port) that the server will use to connect to the client and
// send the notification
// Required: "GRANT CHANGE NOTIFICATION TO ..."
DatabaseChangeRegistration dcr =
    conn.registerDatabaseChangeNotification(prop);

// add the listener
dcr.addListener(new DCNDemoListener());
```



# Database Change Notification

## Associate a query (3/3)

```
Statement stmt = conn.createStatement();  
  
// associate the statement with the registration:  
(OracleStatement)stmt.setDatabaseChangeRegistration(dcr);  
  
// execute a select query:  
ResultSet rs = stmt.executeQuery("select * from dept");
```



# Database Change Notification

## Example: run it

Connection information :

```
local=clienthost.domain.com/10.10.10.101:47632,  
remote=serverhost.domain.com/10.12.12.101:16564
```

```
Registration ID          : 15
```

```
Notification version    : 1
```

```
Event type              : OBJCHANGE
```

```
Database name          : dbj11
```

```
Table Change Description (length=1)
```

```
operation=[INSERT], tableName=SCOTT.DEPT, objectNumber=53268
```

```
Row Change Description (length=2):
```

```
ROW: operation=INSERT, ROWID=AAANAUAABAAALRzAAE
```

```
ROW: operation=INSERT, ROWID=AAANAUAABAAALRzAAF
```



# Startup Shutdown





# Startup Shutdown

## Shutdown

- Must be connected as SYSDBA or SYSOPER
- New method on `oracle.jdbc.OracleConnection`:

```
void shutdown(  
    OracleConnection.DatabaseShutdownMode mode)  
throws java.sql.SQLException
```

- Multiple modes (immediate, abort, final, transactional, etc.)
- May have to call the shutdown method twice depending on the mode



# Startup Shutdown

## Startup

- Must be connected as SYSDBA or SYSOPER in the PRELIM\_AUTH mode
- New method on `oracle.jdbc.OracleConnection`:  

```
void startup(  
    OracleConnection.DatabaseStartupMode mode)  
throws java.sql.SQLException
```
- Multiple modes (NO\_RESTRICTION, FORCE, RESTRICT)



## Other New Features





# Other New Features

- Commit options
- ANYTYPE and ANYDATA support
- Better visibility of the connection properties



**ORACLE IS THE INFORMATION COMPANY**