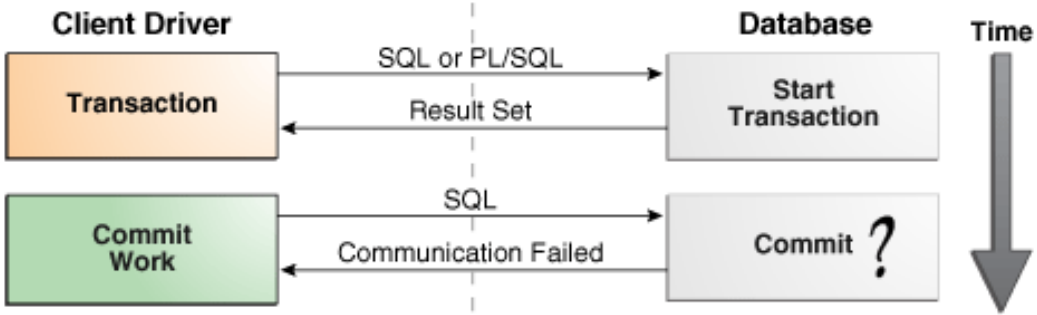
Transaction Guard, Application Continuity …

… choose your degrees of freedom!

Transaction Guard

Transaction Guard: The Problem

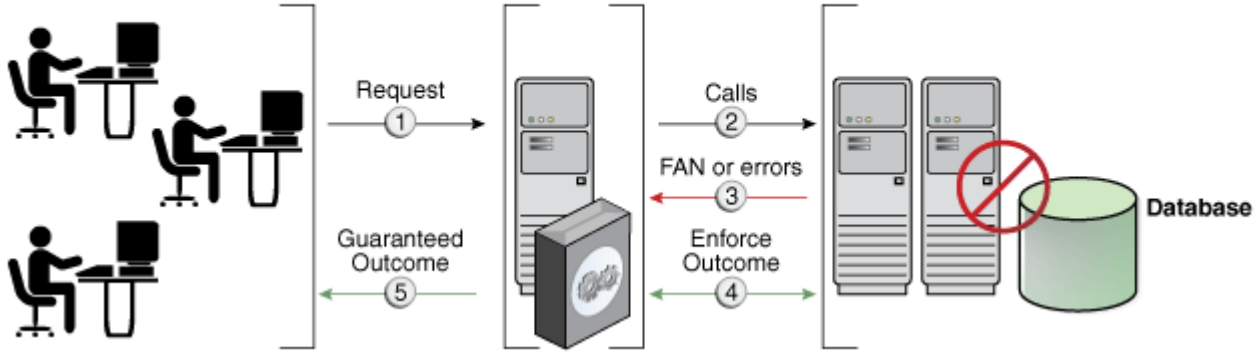
Figure 10-2 Lost Commit Message



Source: Oracle® Database Concepts 12c Release 1 (12.1) , October 2014

Transaction Guard: The Solution

Figure 10-3 Check of Logical Transaction Status



Source: Oracle® Database Concepts 12c Release 1 (12.1) , October 2014

Transaction Guard: Key Features

- Durability of commit *outcome* (extending the D in ACID to client-server realm)
- At-most-once execution semantics (transaction idempotence)
- Works for different transaction types (autocommit, non-autocommit, remote, distributed, embedded, DML/DDL/DCL – XA transactions excluded as of 12.1)
- Works with Single Instance, RAC, (Active) DataGuard, GDS, and Multitenant Architectures
- Detects when DB and client are not in sync

Transaction Guard: Implementation

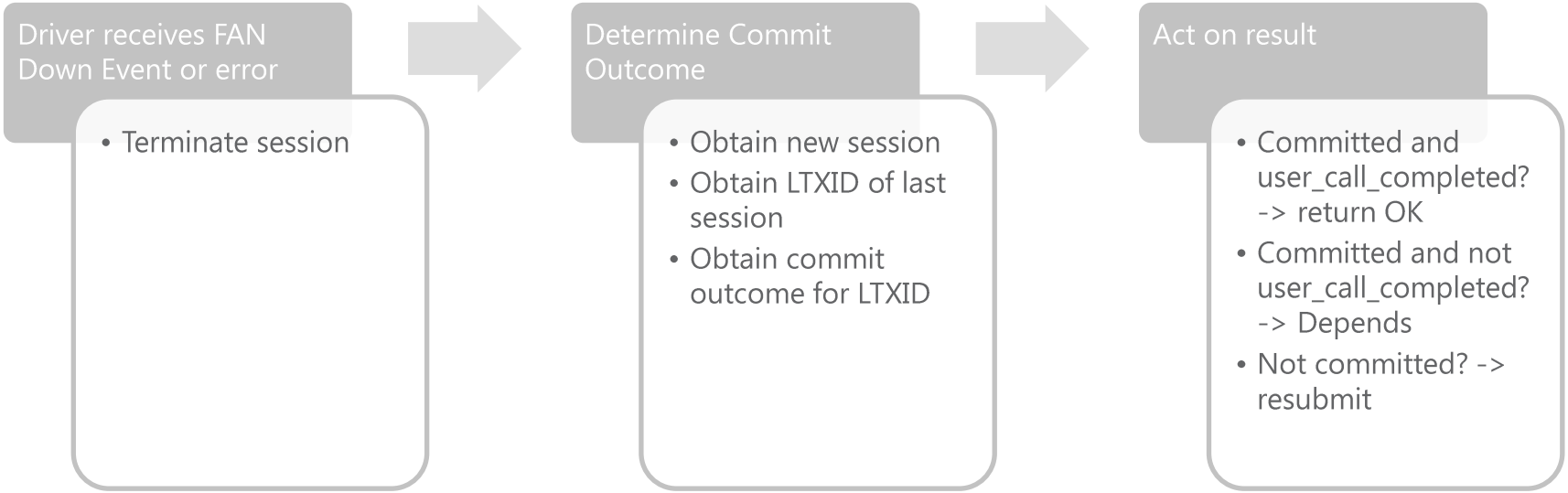
- Logical transaction id (LTXID) assigned at session establishment and incremented at commit
- A callback mechanism allows to retrieve new LTXIDs on change (oracle.jdbc.LogicalTransactionIdEventListener)
- On receiving a recoverable error, client may obtain commit outcome using the failed session's LTXID (DBMS_APP_CONT.GET_LTXID_OUTCOME)
- Obtaining commit outcome results in all other instances of this LTXID being blocked
- LTXIDs are persisted in LTXID_TRANS table in SYSAUX tablespace (one table per PDB)

Transaction Guard: DBA Steps

- Ensure FAN is working
- Create application service with COMMIT_OUTCOME = true
- Possibly modify RETENTION_TIME (default 24h)
- Grant execute on DBMS_APP_CONT to application user
- Possibly move LTXID_TRANS to faster storage

```
srvctl add service -db CDB1 -pdb pdb1 -service pdb1_tg
-commit_outcome true -retention 259200 -preferred
CDB11,CDB12 -clbgoal short -rlbgoal service_time
```

Transaction Guard: Developer Steps



Transaction Guard: DON'Ts

- Don't use GET_LTXID_OUTCOME with the LTXID of the current session. (If it were possible, it would block that session from committing.)
- Don't save LTXID from exception handling. GET_LTXID_OUTCOME is valid only for the last open or completed submission.
- Special handling is required when used with TAF (not recommended for Java applications)

Transaction Guard: Sounds Nice But …

- *… do you really want to put all your transactions in while loops like this???*

```
Connection jdbcConnection = getConnection();
boolean isJobDone = false;
while(!isJobDone) {
  try {
    updateEmployeeSalaries(jdbcConnection);
    isJobDone = true;
  } catch (SQLRecoverableException recoverableException) {
    try {
      jdbcConnection.close();
    } catch (Exception ex) {}
  }
  Connection newJDBCConnection = getConnection();
  LogicalTransactionId ltxid =
((OracleConnection)jdbcConnection).getLogicalTransactionId();
  isJobDone = getTransactionOutcome(newJDBCConnection, ltxid);
  jdbcConnection = newJDBCConnection;
 }
}
```

Java 8 Functional Interfaces to the Rescue!

- Need to decouple the replay logic from the business logic
- Business methods just specify business logic
- TransactionProcessor implements replay logic
- Business methods get passed to TransactionProcessor wrapped in lambda expressions
- TransactionProcessor executes the Functional Interface implementation

Java 8 Functional Interfaces to the Rescue!

- Functional Interface *Transaction*

```
@FunctionalInterface
public interface Transaction {
    public void execute(Connection connection) throws SQLException;
}
```

- *conn -> updateSalaries(conn)* is one implementation

```
TGTransactionProcessor tp = new TGTransactionProcessor(url, appUser, appPasswd);
if (tp.process(conn -> updateSalaries(conn))) {
    logger.fine("Salaries updated.");
}
```

Java 8 Functional Interfaces to the Rescue!

- *TransactionProcessor.process* executes *Transactions*

```
@Override
public boolean process(Transaction transaction) throws SQLException {
    boolean done = false;
    int tries = 0;
    Connection conn = getConnection();
    while (!done && tries <= MAXRETRIES) {
        try {
            transaction.execute(conn);
            conn.commit();
            conn.close();
            done = true;
        } catch (SQLRecoverableException e) {
            try {
                conn.close();
            } catch (Exception ex) {
            }
            LogicalTransactionId ltxid = ((OracleConnection) conn).getLogicalTransactionId();
            Connection newconn = getConnection();
            done = isLTxIdCommitted(ltxid, newconn);
            if (!done) {
                tries++;
                conn = newconn;
            }
        }
    }
    return true;
}
```

And … Does It Work?

- Using Functional Interfaces, implementing Transaction Guard is much less of a hassle
- So we just need to see that it works - time for a **DEMO**!
- Coz ~~*malicious admins*~~ bad things can happen …

```
CDB1.PDB1 SQL> alter system kill session '29,17573,@1';

System altered.
```

Transaction Guard: Demos

- Normal case: it just works!
- GET_LTXID_OUTCOME with the LTXID of the current session

Application Continuity

Application Continuity: Key Features

- Automatic replay of non-committed transactions
- Two forms:
  - **Fully automatic**
    - Oracle Universal Connection Pool (UCP), standalone or as a datasource for a third party application server
    - Oracle WebLogic Server 12c (12.1.2)
    - Using oracle.jdbc.replay.OracleConnectionPoolDataSourceImpl with Oracle JDBC Replay Driver 12c
  - **Explicit request boundaries**
    - Using oracle.jdbc.replay.OracleDataSourceImpl with Oracle JDBC Replay Driver 12c

Application Continuity: Implementation

- Begin and end of requests are tagged either automatically or manually
- Replayable database calls are held by driver until request ends, commit is successful, or replay is disabled
- If a recoverable error occurs, a new connection is automatically obtained
- Transaction Guard is used to obtain commit outcome
- If the transaction has not been committed, calls are replayed and at-most-once outcome is enforced, again using Transaction Guard

Application Continuity: Restrictions

- Available only with JDBC Thin Driver
- Does not support XA
- Does not support oracle.sql concrete classes (BLOB, CLOB, BFILE …)
- Does not support 3rd party statement cache
- Does not support ALTER SYSTEM / ALTER DATABASE statements

Application Continuity: DBA Steps

- Ensure FAN
- Create application service with COMMIT_OUTCOME = true and FAILOVERTYPE = transaction
- Possibly modify REPLAY_INIT_TIME, FAILOVERRETRY, FAILOVERDELAY

```
srvctl add service -service pdb1_appcont -db cdb1 -pdb pdb1
-preferred CDB11,CDB12 -commit_outcome true -retention
259200 -failovertype transaction -replay_init_time 300 -
failoverretry 30 -failoverdelay 3 -clbgoal short -rlbgoal
service_time
```

Application Continuity: Developer Steps - Preparation

- Identify and ensure correct handling of mutable values

```
grant keep [date_time|sys_guid] to appuser;
```

```
alter sequence appuser.<seq> keep;
```

- Identify places where replay should be disabled (e.g., side effects like sending mail, writing files)

```
(( oracle.jdbc.replay.ReplayableConnection)connection).disableReplay();
```

Application Continuity: Developer Steps – Use Replay

- **CASE 1: Fully automatic version**
  - Use, e.g., UCP with oracle.jdbc.replay.OracleDataSourceImpl:

```
import oracle.ucp.jdbc.PoolDataSource;
import oracle.ucp.jdbc.PoolDataSourceFactory;
//
private static final PoolDataSource pds;
pds = PoolDataSourceFactory.getPoolDataSource();
try {
  pds.setConnectionFactoryClassName("oracle.jdbc.replay.OracleDataSourceImpl");
  pds.setMinPoolSize(2);
  // further connection pool properties
  pds.setFastConnectionFailoverEnabled(true);
} catch (SQLException e) {
```

  - **No other action required!**

Application Continuity: Developer Steps – Use Replay

- **CASE 2: Define manual request boundaries**
  - Use oracle.jdbc.replay.OracleDataSourceImpl
  - Manually define request begin and end

```
import oracle.jdbc.replay.OracleDataSource;
import oracle.jdbc.replay.OracleDataSourceFactory;
import oracle.jdbc.OracleConnection;
//
public boolean process(Transaction transaction) throws SQLException {

 Connection conn = getConnection();
 ((OracleConnection) conn).beginRequest();
 //  …
 conn.commit();
 ((OracleConnection) conn).endRequest();
```

Application Continuity: Automatic Replay in Action

- Step 1: We just got a connection from the pool …

```
try {
  conn = pds.getConnection();
} catch (SQLException r) { ... }
conn.setClientInfo("OCSID.MODULE", moduleName);
conn.setAutoCommit(false);
```

- The driver says … (among other things)

```
FINER: On CONN$$$Proxy@4b3ed2f0,Entering beginRequest()
FINER: Sent BEGIN_REQUEST to server
FINER: transaction state: [TRANSACTION_READONLY]
FINER: On CONN$$$Proxy@4b3ed2f0,Recording begins
FINER: On CONN$$$Proxy@4b3ed2f0,Exiting beginRequest()
FINER: On CONN$$$Proxy@4b3ed2f0,recording method setClientInfo
FINER: On CONN$$$Proxy@4b3ed2f0,recording method setAutoCommit
```

Application Continuity: Replay in Action

- Step 2: We're doing our stuff …

```
PreparedStatement stmt = conn.prepareStatement(query,
ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery();
while (rs.next()) {
  //
  rs.updateRow();
}
```

- The driver says … (among other things)

```
FINER: On CONN$$$Proxy@4b3ed2f0,recording method prepareStatement
FINER: On CONN$$$Proxy@4b3ed2f0,recording method executeQuery
FINER: On CONN$$$Proxy@4b3ed2f0,recording method next
FINER: On CONN$$$Proxy@4b3ed2f0,recording method getInt
FINER: On CONN$$$Proxy@4b3ed2f0,recording method getInt
FINER: On CONN$$$Proxy@4b3ed2f0,recording method updateInt
FINER: On CONN$$$Proxy@4b3ed2f0,recording method updateRow
```

Application Continuity: Replay in Action

- Step 3: Our session gets killed …
- The driver says … (among other things)

```
FINER: On CONN$$$Proxy@4b3ed2f0,Entering replay,original
error=java.sql.SQLRecoverableException: ORA-00028: your session has been killed
FINER: Reconnecting: RETRY 1
FINER: Reconnect succeeded,new connection=oracle.jdbc.driver.T4CConnection@71cea1b8
FINER: On CONN$$$Proxy@4b3ed2f0,PREPARE_REPLAY RPC code: 94,SQL text: update
(select rowid as "__Oracle_JDBC_interal_ROWID__", empno, comm from AC.emp) set COMM
= :rowid0 WHERE  ROWID = :rowid1
FINER: PREPARE_REPLAY succeeded,committed: false,embedded: false
...
FINER: BEGIN_REPLAY succeeded
FINER: On CONN$$$Proxy@4b3ed2f0,replaying method setClientInfo
FINER: On CONN$$$Proxy@4b3ed2f0,replaying method setAutoCommit
```

Application Continuity: Replay in Action

- Step 4: We sit and watch …
- The driver says … (among other things)

```
FINER: On RSET$$$Proxy@32177fa5,replaying method next
FINER: On RSET$$$Proxy@32177fa5,server replay context set: NOT NULL
FINER: On RSET$$$Proxy@32177fa5,replaying method getInt
FINER: On RSET$$$Proxy@32177fa5,replaying method getInt
FINER: On RSET$$$Proxy@32177fa5,replaying method updateInt
FINER: END_REPLAY succeeded
FINER: On CONN$$$Proxy@4b3ed2f0,replaying last call
INFO: On CONN$$$Proxy@4b3ed2f0,replay succeeds
```

- … and done!

Application Continuity: Explicit request boundaries

- When would an application want to do this?
  - Custom connection pool in place
  - Fewer transactions should be replayed than exempted from replay
- Shouldn't any application use UCP for its HA features (Fast Application Notification, Fast Connection Failover)?
  - Probably yes, but starting from 12.1.0.2, clients may receive a subset of FAN events through the Oracle RAC FAN APIs included in simplefan.jar

Application Continuity: Demos

- UCP: Normal case
- UCP: ORA-41412 (sysdate)
- Manual: ORA-41412 (general result changed)
- Manual: Commit outside request
- Manual: Set autocommit off outside request

References

- Transaction Guard with Oracle Database 12c
  http://www.oracle.com/technetwork/database/database-cloud/private/transaction-guard-wp-12c-1966209.pdf
- Application Continuity with Oracle Database 12c
  http://www.oracle.com/technetwork/database/options/clustering/application-continuity-wp-12c-1966213.pdf
- Oracle Database JDBC Developers Guide, Appendix B: Oracle RAC Fast Application Notification
  http://docs.oracle.com/database/121/JJDBC/apxracfan.htm#JJDBC28934

Questions / Recoverable Errors?

Thank you!

Sigrid Keydana

Tech Event, February 27 2016

trivadis
makes IT easier.