Extending TPTP for Database Unit Testing

Wei Liu, Hong-Lee Yu, Der-Ping Chou, Don Clare
IBM Data Tools
Agenda

- Rationale
- Requirements
- Implementation
- DbUnit Comparison
- Use Cases
Customer Pain Points and Opportunities

**Agility**
- Ability to react to changing needs
- Ability to react to changing technology opportunities
- Treat change as an opportunity to be competitive
- Flexible sourcing and resources

**Data**
- Over 60% mission critical apps require data components, but most ALMs (Application Life Cycle Management) pay little attention to data.
- Common approach is to build a silo wall and discourages database changes

**Governance**
- Being Compliant
- Auditable processes
- Conforming to complex and changing mandates
- High governance and control

The need for a new approach to database life cycle management

*Ziff Davis: Over 40% of CIO’s report they are unable to react as rapidly as business needs change*

*Wall Street Tech: $5.1 Billion is the amount companies will spend in compliance-related projects in 2005*
Agile Model-Driven and Test-Driven Development

- Identify the high-level scope
- Identify initial "requirements stack"
- Identify an architectural vision

- Modeling is part of iteration planning effort
- Need to model enough to give good estimates
- Need to plan the work for the iteration

- Work through specific issues on a JIT manner
- Stakeholders actively participate
- Requirements evolve throughout project
- Model just enough for now, you can always come back later

- Develop working software via a test-first approach
- Details captured in the form of executable specifications

Copyright © IBM Corp., 2008. All rights reserved. Source code in this presentation is made available under the EPL, v1.0; remainder of the presentation is licensed under Creative Commons Att. Nc Nd 2.5 license.

EclipseCon 2008
Test Data Generation and Unit regression test are fundamental to Test-Driven Database Development

**Diagram:**
- Application
- OLTP Access
- Test Data Generator
- Test Data
- Data Load
- Batch Data
- Data Extract
- Batch Data

**Clear-box testing:**
- Stored procedures/functions
- Triggers
- Views
- Constraints
- Existing data quality
- Referential integrity/data consistency

**Black-box testing:**
- Data values being persisted
- Data values being retrieved
- Stored procedures/functions
- ...
Evolutionary database development – achieve agility while maintaining governance
Agenda

- Rationale
- Requirements
- Implementation
- DbUnit Comparison
- Use Cases
Unit Test Framework integrated with Database Views

- Facilitate test driven development

- Ease the creation of tests
  - Database metadata accessible during test creation
  - Create test from a manual execution (i.e., Stored Procedure)

- Associate test results to database objects
Generate JUnit test code

- Many database developers are not fluent in Java

- Provide a UI to allow for the creation and modification of tests and their behaviors

- Java test code artifacts should be modifiable by the user, and modifications should be preserved in subsequent test code regeneration at the user’s discretion
Provide layered database state setup

- Database testing will often require a substantial amount of data context
- Context data which will not be modified should be set up once per test suite
- Data modified by tests should be set up before each test case
- Allow the user to specify setup and teardown actions to be performed per test suite and per test case
Separate test data from test cases

- Test data should be easily modifiable without editing the test case
- Increase test coverage by adding test data
- Construct test data from external sources
Database Connectivity Support

- Default connectivity should use the database connection for the database object’s development project
- Additional connections should be supported when required
- Definition of connections by the user should be as simple as possible
- Running a test on a different, equivalent database should not require a change to the test
Support a full set of database actions

- Individual SQL statement execution and validation
- SQL script execution
- Stored Procedure execution and validation
- Transaction control
- Database command execution
Support a full set of validation options

- SQLState
- Update count
- ResultSet data
- ResultSet row count
- Stored Procedure output parameter
- All standard comparison operators
Agenda

- Rationale
- Requirements
- Implementation
- DbUnit Comparison
- Use Cases
Using and extending TPTP

- The Eclipse™ Test and Performance Tools Project (TPTP) provides a comprehensive framework and set of services for test
- Data models represent the structure, behavior, and execution of tests
- UI components which can be used as is or extended
- Execution environment for running locally or remotely
Using and extending the TPTP Platform test data models

- **UML2 Test Profile Data Model**
  - Define tests
  - Create and manage test artifacts
  - Unique test types defined for database testing

- **UML2 Interaction Data Model**
  - Define test behavior
  - Simpler form provided by the Behavioral Façade Model
    - Extend to define database test actions, which will associate or identify
      - Database connections
      - Datapools
TPTP Behavioral Façade Model
Database Test Actions Model

**ParameterType**
- NONE
- DATAPool_REF
- INPUT
- RESULT_SET_COLUMN
- RESULT_SET_ROW_COUNT
- SP_OUTPUT_PARM
- SQLSTATE
- UPDATE_COUNT
- OTHER

**ComparatorType**
- NO_COMPARE
- NEQ
- EQ
- LE
- GE
- LT
- GT
- ACC_PLAN
- OTHER

**RDbActionEnum**
- SP_CALL
- SQL_EXECUTE
- START_UOW
- COMMIT_UOW
- ROLLBACK_UOW
- EXPLAIN
- USER_SUPPLIED
- XCONN_XFER
- SQL_SCRIPT_EXECUTE
- COMMAND_EXECUTE

**RDbTestAction**
- connectionName : String
- statement : String
- executeType : RDbActionEnum
- validate : Boolean

**DPLDatapool**

**IAction**

Association:
- mappings
- datapool

Copyright © IBM Corp., 2008. All rights reserved. Source code in this presentation is made available under the EPL, v1.0; remainder of the presentation is licensed under Creative Commons Att. Nc Nd 2.5 license.
Using and extending the TPTP Platform UI components

- Many UI components will be utilized as is from TPTP, including the common editors for test metadata and behavior, deployment, and execution history, as well as the datapool editor.

- A New Database Unit Test Wizard will be added.

- Additional editor capability will be provided for test case and setup/teardown behavior (adding and specifying database test actions).
Extending TPTP test execution

- Utilize components of the TPTP Java execution environment

- Extend ExecutionEnvironmentAdapter, ExecutableObjectAdapter, and ExecutionDeploymentAdapter to support the database unit test type

- Extend hyades.test.common.junit classes as required
Leveraging DTP

- The Eclipse Data Tools Project (DTP) provides frameworks to simplify the configuration and management of the database development environment.
- The Model Base subproject provides a rich, generic SQL model which is extensible with vendor specific database definitions.
- The Connectivity subproject provides a framework for managing database connections and their details, and UI components for creating and manipulating database connection profiles.
Using the DTP SQLModel

- The database unit under test is represented in the model instance.

- During testcase editing, the model can provide metadata to expedite the creation of the test behavior and the structure of datapools.

- The model can further be used to identify associated database objects which either impact or are impacted by the unit under test.
DTP Connectivity UI Components

- The DTP Connectivity UI components facilitate creating new database connection profiles, and editing and selecting existing connection profiles

- When creating a new connection profile
  - The properties UI is tailored for the database driver selected
  - Some properties are seeded with default values
  - The connection URL is computed from other properties
  - The password can be saved
  - The connection profile can be tested before saving
DTP Connectivity UI Components

- Connections window showing options like CS, SAMTSAMP, and SAMPLE.
- New Connection Profile window with details:
  - Drivers: Oracle 10 - Oracle Thin Driver Default
  - Properties:
    - Name: SAMPLE
    - Description: Databases
    - Category: SAMPLE
    - JDBC Driver Class: com.ibm.db2.jcc.DB2Driver
    - Class Location: C:\Program Files\IBM\SQLLIB\java\db2\cc.jar
    - Connection URL: jdbc:oracle:thin:@hq.partesco.com:1521:inventory
    - User ID: peter
    - Password: *********
  - Connection URL: jdbc:oracle:thin:@hq.partesco.com:1521:inventory
  - Catalog: User

Test Connection button available.
Database Connectivity for a Test

- DTP Connection Profiles will be used to create database connections

- A Connection Profile within the Connection Profile Repository is retrieved by name and employed within a database test action’s generated code

- By default, the Connection Profile associated with the database object’s data project or data source is used for all test actions

- Connection Profile substitution can be specified via a property of the test suite deployment artifact
Java Code Generation

- Generate a JUnit-based test class from the test data model instance
  - Generated test class extends RDbJUnitTestCase which provides common database test behavior
  - Inner class extends junit.extensions.TestSetup to implement setup and teardown at the suite level

- Test code artifacts are generated using “JET1”

- JMerge provides a mechanism to preserve user modification to generated code
Agenda

- Rationale
- Requirements
- Implementation
- DbUnit Comparison
- Use Cases
## Comparison with DbUnit

<table>
<thead>
<tr>
<th></th>
<th>DbUnit</th>
<th>Database Unit Test Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Target User Population</strong></td>
<td>Java database application developers</td>
<td>Database developers</td>
</tr>
<tr>
<td><strong>Database Connectivity</strong></td>
<td>Connections are defined in the test case or Ant script.</td>
<td>Connections are specified in DTP connection profiles.</td>
</tr>
<tr>
<td><strong>Database Operations Support</strong></td>
<td>Abstract command classes encapsulate common and composite DML for setup/teardown.</td>
<td>User-supplied SQL scripts, single statements, or stored procedure calls.</td>
</tr>
</tbody>
</table>
## Comparison with DbUnit

<table>
<thead>
<tr>
<th></th>
<th>DbUnit</th>
<th>Database Unit Test Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test Data Container</strong></td>
<td>Datasets provide hierarchical structure for multiple tables.</td>
<td>Datapools provide tabular structure defined by user, equivalence classes share metadata.</td>
</tr>
<tr>
<td><strong>Test Data External Sources</strong></td>
<td>Excel file, load from database or query.</td>
<td>Csv file, action to load datapool from query.</td>
</tr>
<tr>
<td><strong>User Interface – Test Data</strong></td>
<td>Edit dataset using XML or text editor.</td>
<td>Edit datapool using TPTP datapool editor.</td>
</tr>
</tbody>
</table>
## Comparison with DbUnit

<table>
<thead>
<tr>
<th></th>
<th>DbUnit</th>
<th>Database Unit Test Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Interface – Test Behavior</strong></td>
<td>- Java API provides function and interfaces for database manipulation and validation. User writes Java code to implement tests.</td>
<td>- Eclipse UI editors to create tests of ordered database actions, specifying database connections and datapools, without writing Java code.</td>
</tr>
<tr>
<td><strong>Result Validation</strong></td>
<td>- Assertion support for equality comparison between datasets or ITable instances.</td>
<td>- Generated results validation code per datapool variable, all common comparators supported.</td>
</tr>
</tbody>
</table>
## Comparison with DbUnit

<table>
<thead>
<tr>
<th>Automated Testing Support</th>
<th>DbUnit</th>
<th>Database Unit Test Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The DbUnit Ant task is an alternative to java code invocation of DbUnit functions.</td>
<td>The TPTP Ant task provides an alternative execution harness to run test cases “headless”.</td>
</tr>
<tr>
<td>Test Execution History</td>
<td>Maintaining test results and execution history is outside the scope of DbUnit.</td>
<td>The TPTP framework has rich support for viewing test results and execution history.</td>
</tr>
</tbody>
</table>
Agenda

- Rationale
- Requirements
- Implementation
- DbUnit Comparison
- Use Cases
Use Case

- Create a database unit test suite
- Create a database unit test case for a stored procedure
Use Case – create a database unit test suite

- Right click on a Data Development Project and select **New** -> **Test Element**
- Select Test -> Database Unit Test wizard
Use Case – create a database unit test suite

- Define name and location of the test suite
Use Case – create a database unit test suite

- Define once per test suite setup option
Use Case – create a database unit test suite

- Define once per test suite teardown option
- Finish
Use Case – create a database unit test suite

- Edit the setup/teardown SQL script with SQL Editor
Use Case – create a database unit test case

- Right click on a stored procedure and Create Unit Test-case
Use Case – create a database unit test case

- Select the test in which the test case will be generated
Use Case – create a database unit test case

- Select the Database Unit Test Case wizard
Use Case – create a database unit test case

- Define the name and description of the test case
Use Case – create a database unit test case

- Define a database action for the test case
- Finish
Use Case – create a database unit test case

- **Test method sample**

```java
/**
 * PROC_COLS Test
 *
 * Call the PROC_COLS stored procedure, and verify
 * the output parameter is correct.
 *
 * @throws java.lang.Exception
 * @generated
 */
public void testPROC_COLSTest() throws java.lang.Exception
{
    doActionTestPROC_COLSTestPROC_COLSSPCall();
}
```
Use Case – create a database unit test case

- Database test action method sample

```java
/**
 * PROC_COLS Test PROC_COLS SP Call
 * *
 * @throws java.lang.Exception
 * @generated
 */

public void doActionTestPROC_COLSTestPROC_COLSSPCall() throws java.lang.Exception {
    IConnection iConn = null;
    Connection conn = null;
    Statement stmt = null;
    IDatapoolIterator dpIterator = null;

    try {
        iConn = getConnection("SAMPLE");
        conn = (Connection)iConn.getRawConnection();
        IDatapoolFactory dpFactory = new Common_DatapoolFactoryImpl();
        File dpFile = new File("PROC_COLSInput.datapool");
        IDatapool datapool = dpFactory.load(dpFile,false);
        dpIterator = dpFactory.open(datapool,
            "org.eclipse.hyades.datapool.iterator.DatapoolIteratorSequentialPrivate");
        dpIterator.dpInitialize(datapool,-1);
        stmt = conn.prepareCall("CALL SAMPLE.PROC_COLS(?,?,?)");
    }
```

Use Case – create a database unit test case

- Database test action method sample – cont.

```java
((CallableStatement)stmt).registerOutParameter(3, java.sql.Types.INTEGER);
while (!dpIterator.dpDone()) {
    IDatapoolRecord dpRecord = dpIterator.current();
    Object cellValue = dpRecord.getCell(0).getCellValue();
    if (cellValue == null) {
        (CallableStatement)stmt.setNull(1, java.sql.Types.VARCHAR);
    } else {
        (CallableStatement)stmt.setObject(1, dpRecord.getCell(0).getCellValue(),
                                           java.sql.Types.VARCHAR);
    }
    cellValue = dpRecord.getCell(1).getCellValue();
    if (cellValue == null) {
        (CallableStatement)stmt.setNull(2, java.sql.Types.INTEGER);
    } else {
        (CallableStatement)stmt.setObject(2, dpRecord.getCell(1).getCellValue(),
                                           java.sql.Types.INTEGER);
    }
```

Copyright © IBM Corp., 2008. All rights reserved. Source code in this presentation is made available under the EPL, v1.0; remainder of the presentation is licensed under Creative Commons At. Nc Nd 2.5 license.
Use Case – create a database unit test case

- Database test action method sample – cont.

```java
stmt.execute();
while(true) {
    String expString = dpRecord.getCell(2).getStringValue();
    Object actObject = ((CallableStatement)stmt).getObject(3);
    if (expString == null) {
        assertTrue(((CallableStatement)stmt).wasNull());
    } else {
        Integer expValue = new Integer(expString);
        assertTrue(expValue.compareTo(actObject) == 0);
    }
    break;
}
```

finally {
    if (stmt != null) {
        stmt.close();
        stmt = null;
    }
    dpIterator.reset();
    dpIterator = null;
    conn = null;
    iConn = null;
}
```
Resources

- Eclipse Test and Performance Tools Project

- Agile Data
  - [http://www.agiledata.org/](http://www.agiledata.org/)

- Eclipse Data Tools Project

- Introduction to JET

- DbUnit Project
  - [http://www.dbunit.org/index.html](http://www.dbunit.org/index.html)
Questions?
Legal Notices

IBM and the IBM logo are trademarks or registered trademarks of IBM Corporation, in the United States, other countries or both.

Rational and the Rational logo are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries or both.

Java and all Java-based marks, among others, are trademarks or registered trademarks of Sun Microsystems in the United States, other countries or both.

Eclipse and the Eclipse logo are trademarks of Eclipse Foundation, Inc.

Other company, product and service names may be trademarks or service marks of others.

THE INFORMATION DISCUSSED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, AND IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, SUCH INFORMATION. ANY INFORMATION CONCERNING IBM'S PRODUCT PLANS OR STRATEGY IS SUBJECT TO CHANGE BY IBM WITHOUT NOTICE.