



IBM™ Software Group, Information Management

# Extending TPTP for Database Unit Testing

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

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.

March 20, 2008

EclipseCon™ 2008

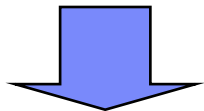
# Agenda

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

# Customer Pain Points and Opportunities

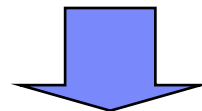
## Agility

- 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



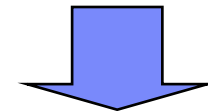
## Data

- Over 60% mission critical apps requires 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



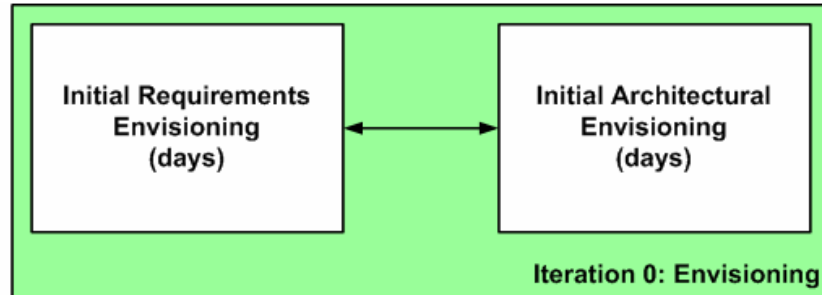
## The need for a new approach to database life cycle management

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

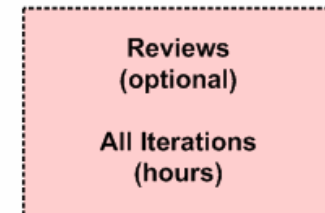
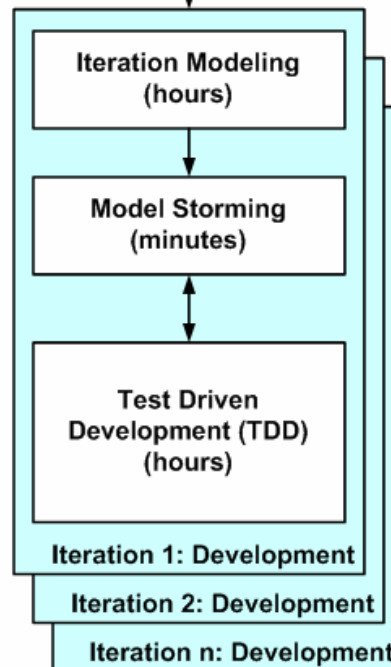
*Wall Street Tech: \$5.1 Billion is the amount companies will spend in compliance-related projects in 2005*

# Agile Model-Driven and Test-Driven Development

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

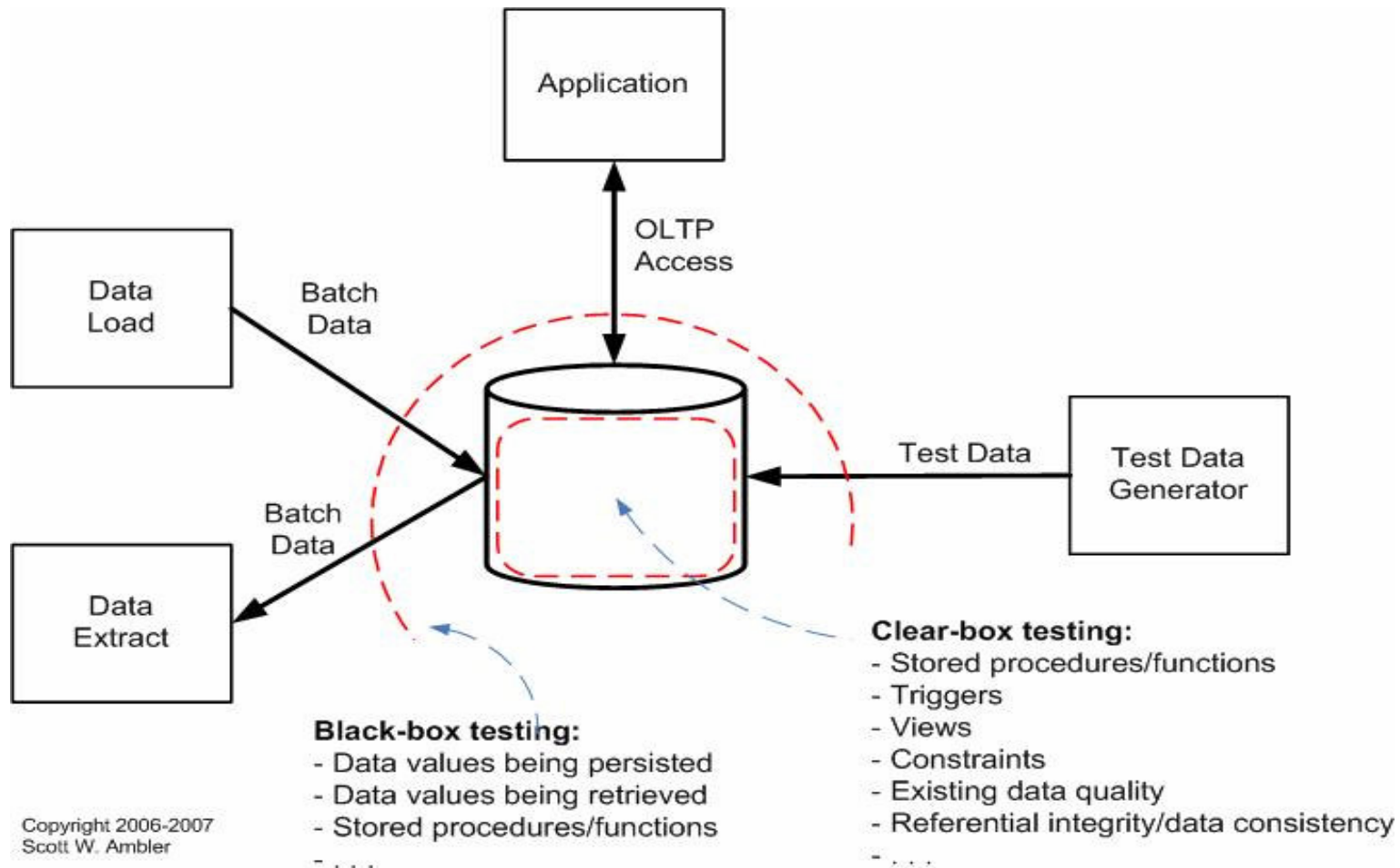


- Modeling is part of iteration planning effort
- Need to model enough to give good estimates
- Need to plan the work for the iteration
- Work through specific issues on a JIT manner
- Stakeholders actively participate
- Requirements evolve throughout project
- Model just enough for now, you can always come back later
- Develop working software via a test-first approach
- Details captured in the form of executable specifications



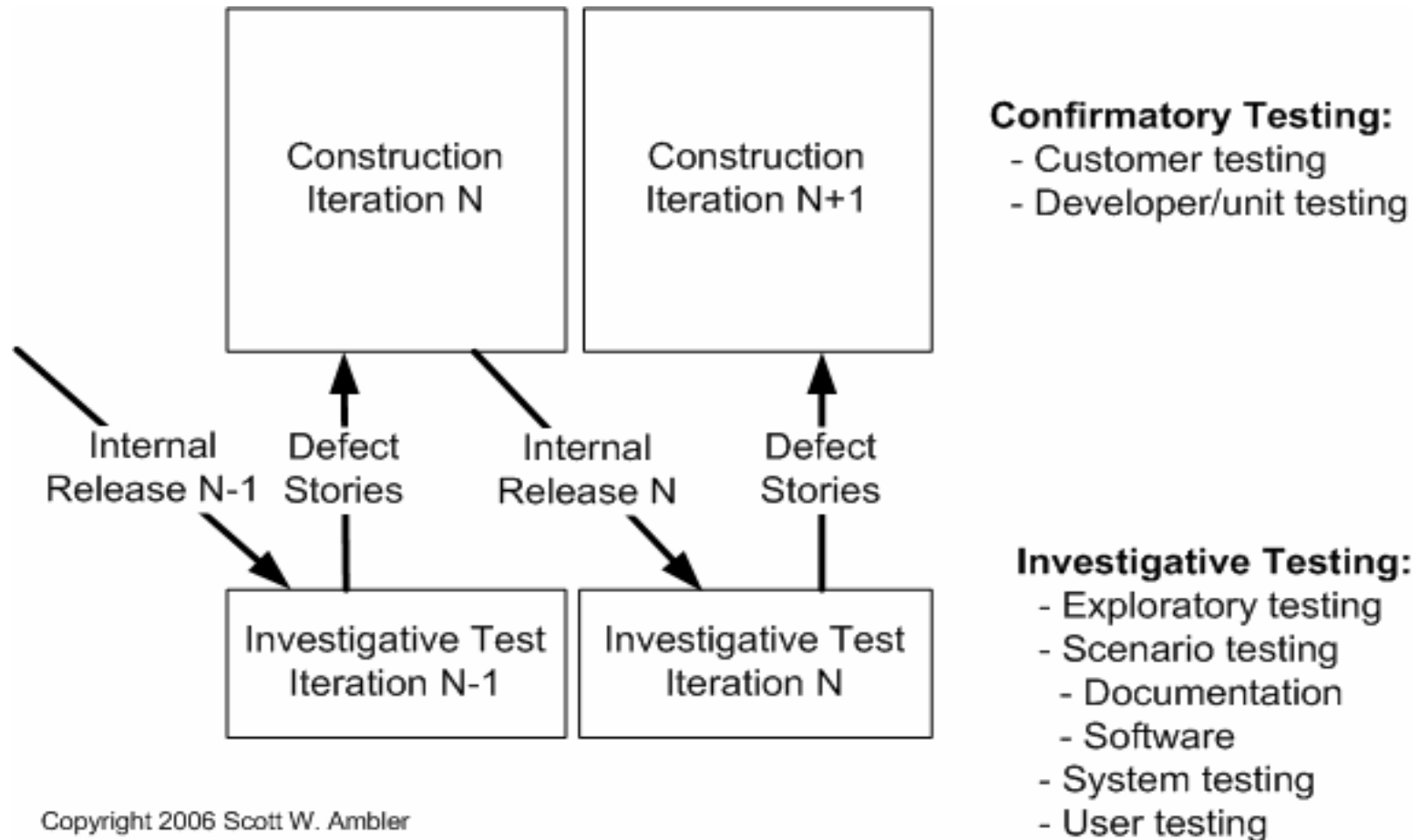
Copyright 2003-2007  
Scott W. Ambler

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



Copyright 2006-2007  
Scott W. Ambler

# Evolutionary database development – achieve agility while maintaining governance



Copyright 2006 Scott W. Ambler

# 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

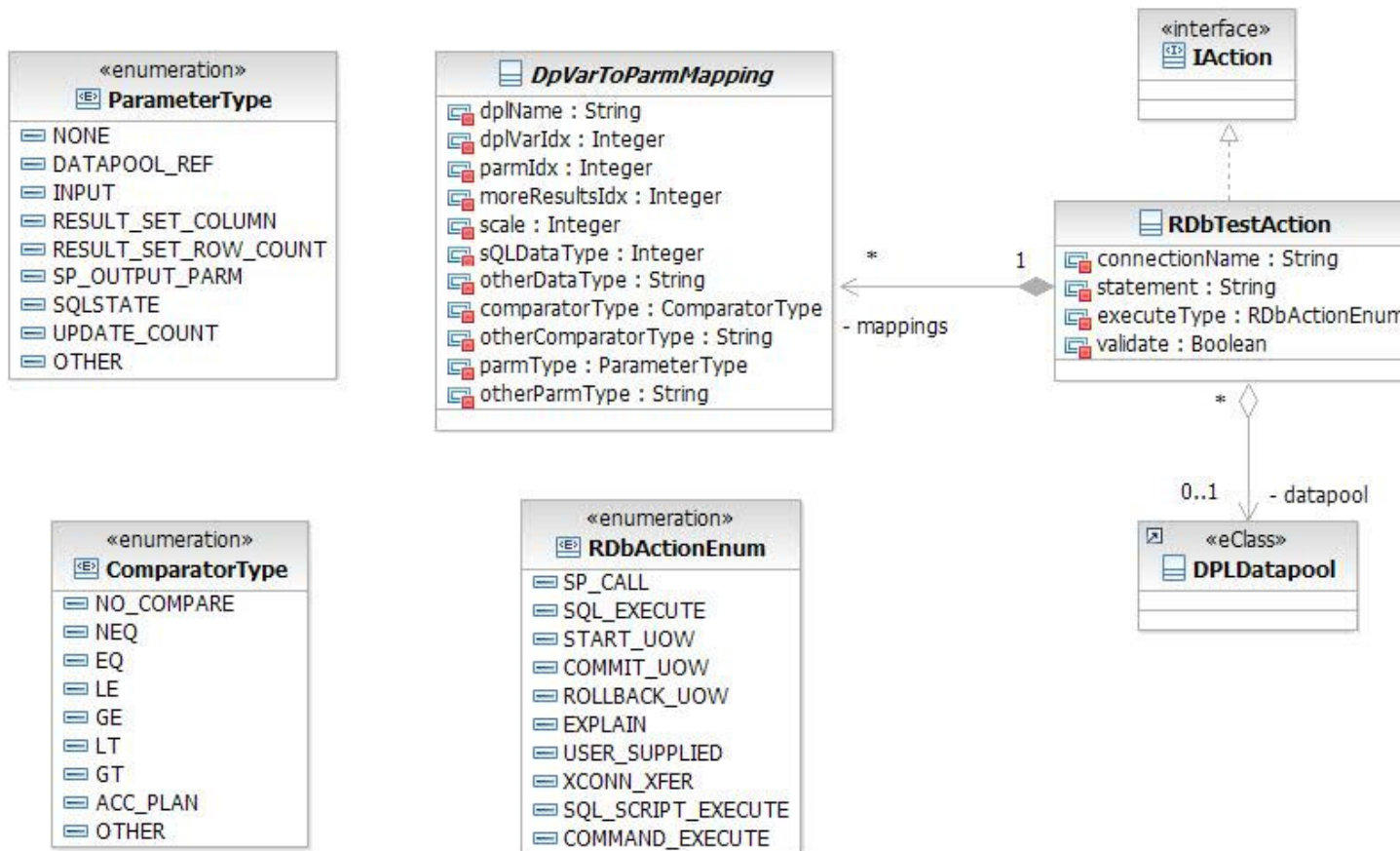
- The Eclipse™ Test and Performance Tools Project (TPTP) provides a comprehensive framework and set of services for test
- Data models represent the structure, behavior, and execution of tests
- UI components which can be used as is or extended
- Execution environment for running locally or remotely

## 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



# Database Test Actions Model



## 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

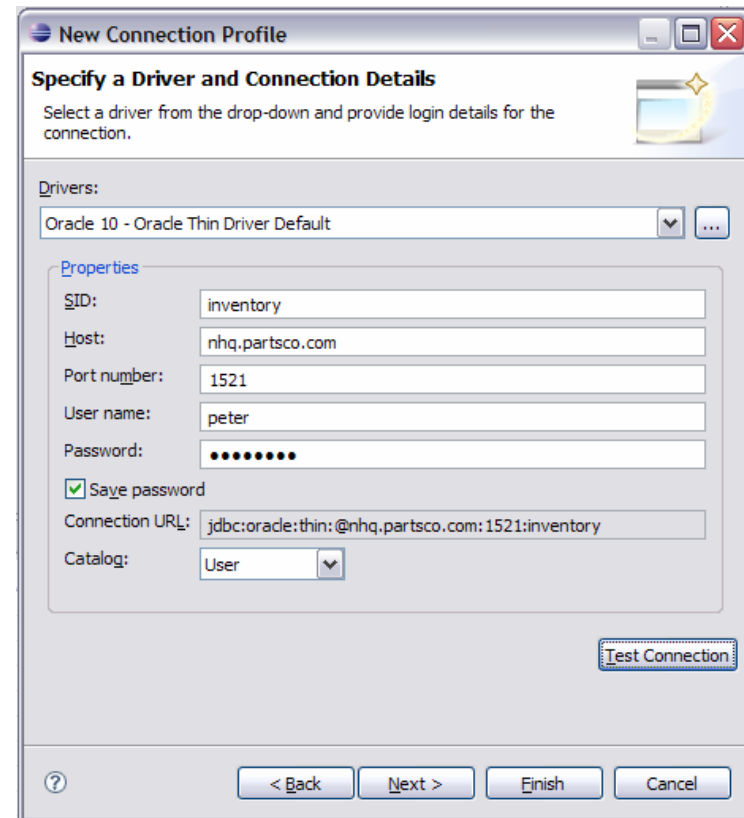
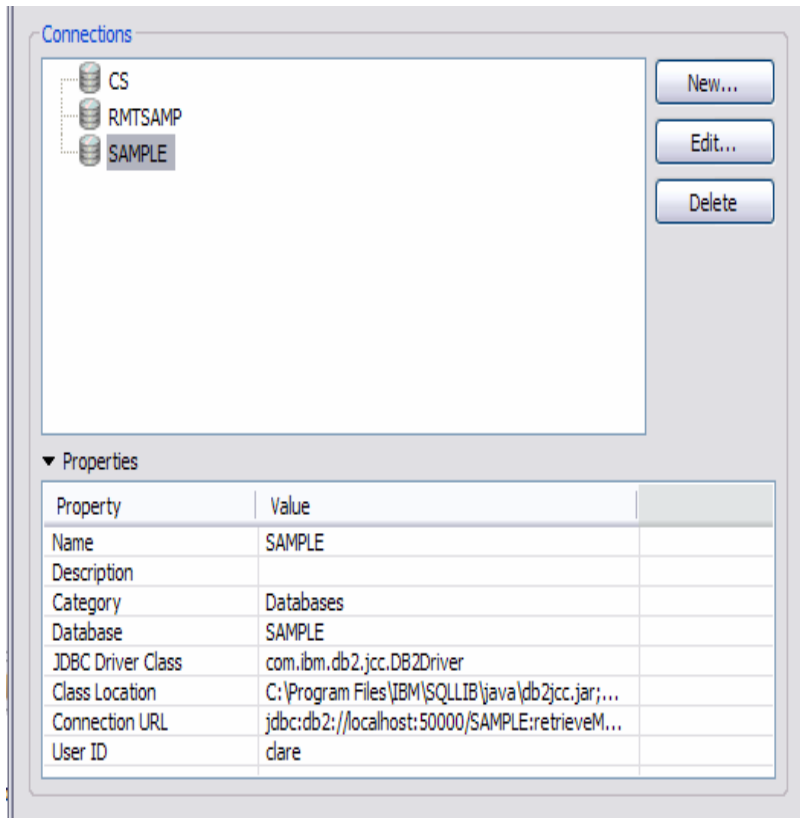
## Using the DTP SQLModel

- 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



## 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

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

## Comparison with DbUnit

	DbUnit	Database Unit Test Framework
Test Data Container	Datasets provide hierarchical structure for multiple tables.	Datapools provide tabular structure defined by user, equivalence classes share metadata.
Test Data External Sources	Excel file, load from database or query.	Csv file, action to load datapool from query.
User Interface – Test Data	Edit dataset using XML or text editor.	Edit datapool using TPTP datapool editor.

## Comparison with DbUnit

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

## Comparison with DbUnit

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

# Agenda

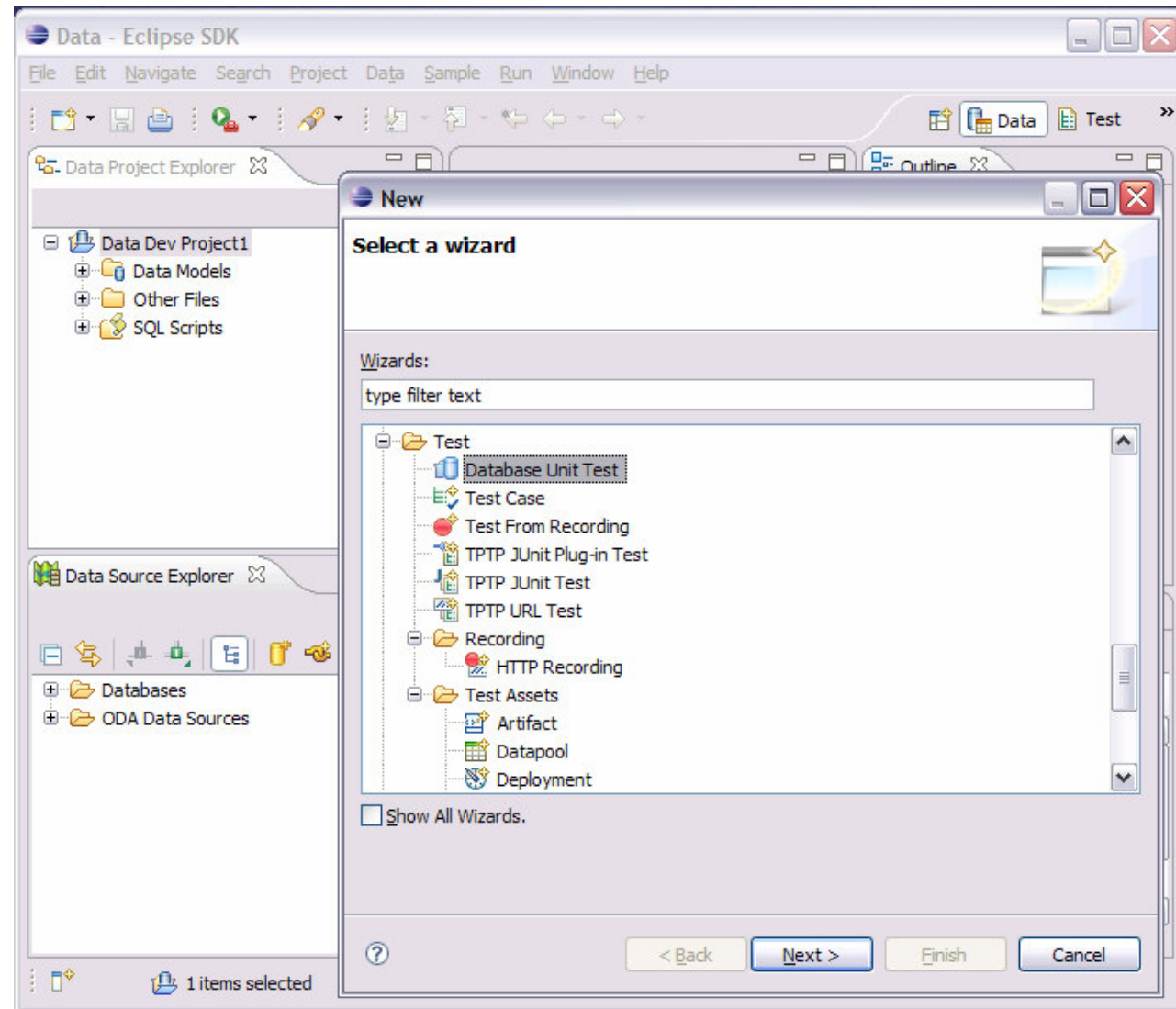
- Rationale
- Requirements
- Implementation
- DbUnit Comparison
- Use Cases

## Use Case

- Create a database unit test suite
- Create a database unit test case for a stored procedure

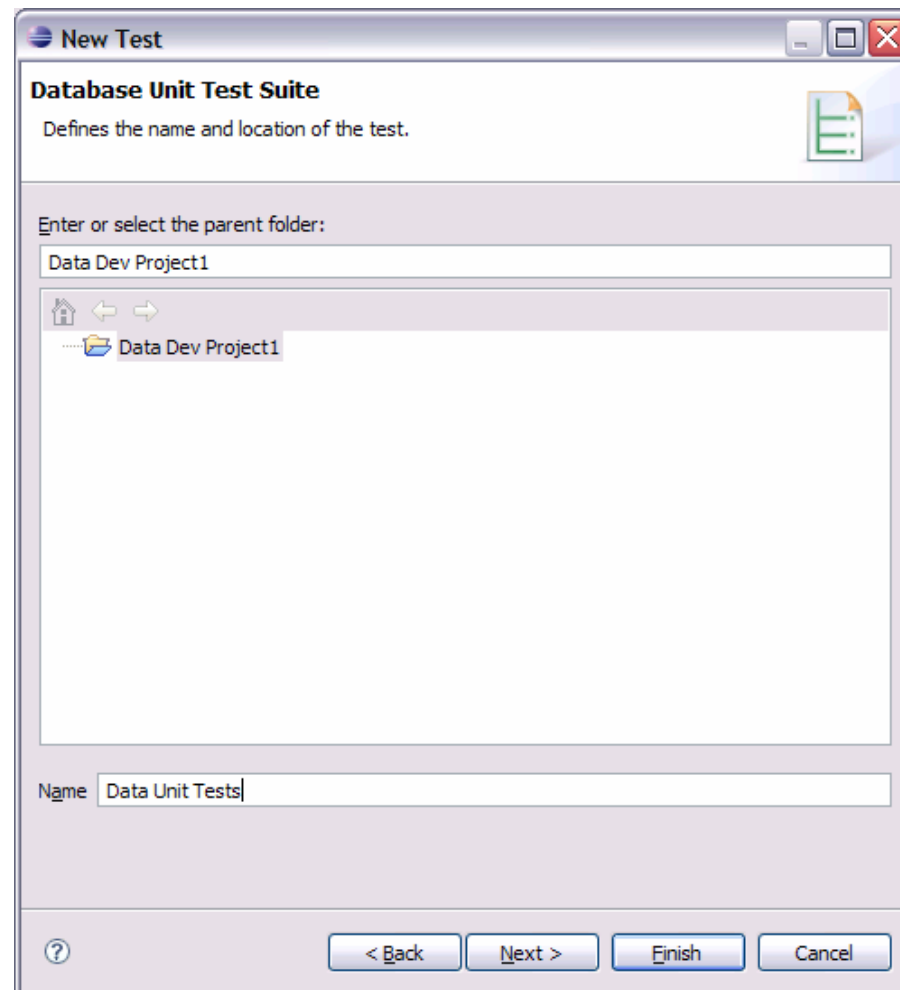
## Use Case – create a database unit test suite

- Right click on a Data Development Project and select **New** -> **Test Element**
- Select Test -> Database Unit Test wizard



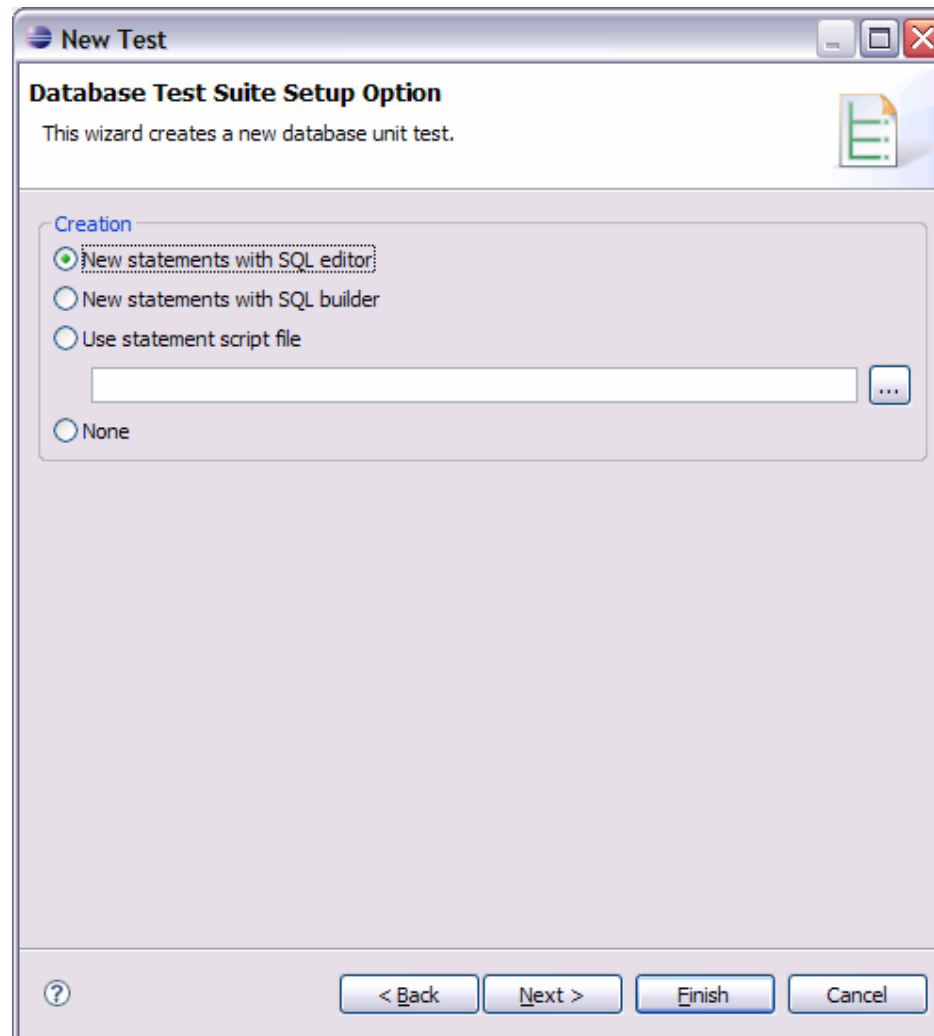
## Use Case – create a database unit test suite

- Define name and location of the test suite



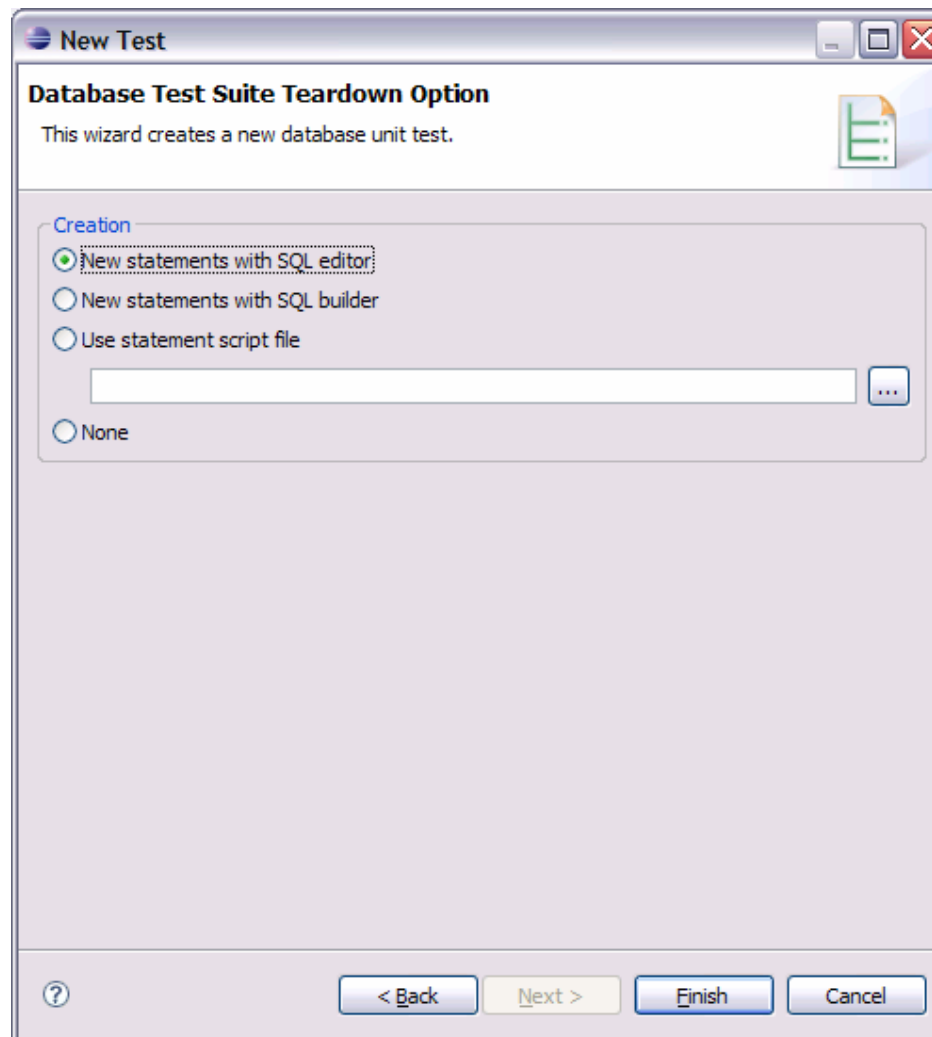
## Use Case – create a database unit test suite

- 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

The screenshot shows the Eclipse IDE interface. The main window is titled 'PROC\_COLS' and contains a file named 'DefaultSetUp.sql'. The SQL script in the editor is as follows:

```
CREATE TABLE ACT (
  ACTNO SMALLINT NOT NULL,
  ACTKWD CHAR(6) NOT NULL,
  ACTDESC VARCHAR(20) NOT NULL
)
DATA CAPTURE NONE
IN USERSPACE1
DISTRIBUTE BY HASH(ACTNO);

CREATE UNIQUE INDEX XACT2 ON ACT (ACTNO ASC, ACTKWD ASC) PCTFRE

ALTER TABLE ACT ADD CONSTRAINT PK_ACT PRIMARY KEY (ACTNO);

INSERT INTO ACT VALUES (100, 'ADMSYS', 'SYSTEM ADMINISTRATION');
INSERT INTO ACT VALUES (200, 'USERMGR', 'USER MANAGEMENT');
INSERT INTO ACT VALUES (300, 'APPSPT', 'APPLICATION SUPPORT');
```

A context menu is open over the editor, listing various actions and their keyboard shortcuts:

- Undo Typing (Ctrl+Z)
- Revert File
- Save
- Show In (Alt+Shift+W)
- Cut (Ctrl+X)
- Copy (Ctrl+C)
- Paste (Ctrl+V)
- Shift Right
- Shift Left
- Add to Snippets...
- Run As
- Debug As
- Profile As
- Validate
- Team
- Compare With
- Replace With
- Preferences...
- Content Assist (Ctrl+Space)
- Content Tip (Ctrl+Shift+Space)
- Format SQL (Ctrl+Shift+F)
- Use Database Connection...
- Run SQL
- Set Statement Terminator

At the bottom of the IDE, there is a 'Data Output' window showing the execution results:

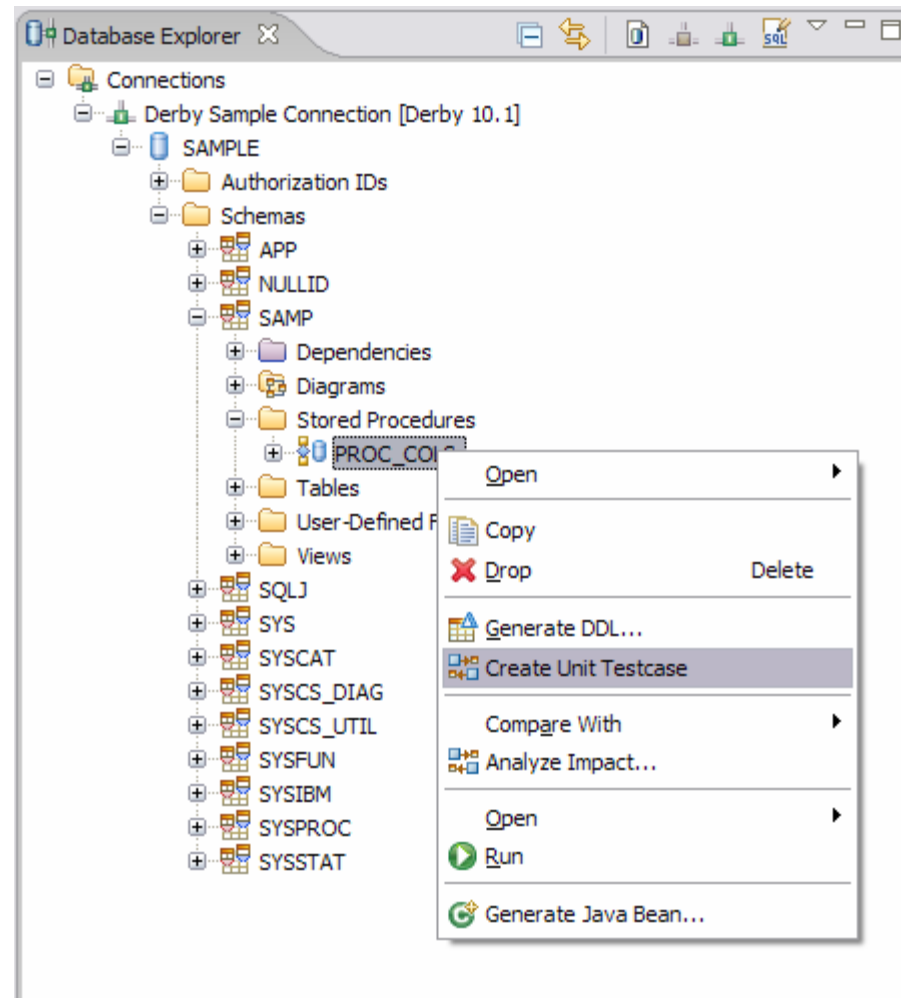
Status	Action	Object Name
✓ Success	Run	SQL Editor
✓ Success	Run	SQL Editor
✓ Success	Run	Sample Conte...
✓ Success	Run	Sample Conte...
✓ Success	Run	Sample Conte...

The 'Messages' tab in the Data Output window shows the following output:

```
Starting run
CREATE UNIQUE INDEX DEP_IDX ON DEPARTMENT (
Run successful
```

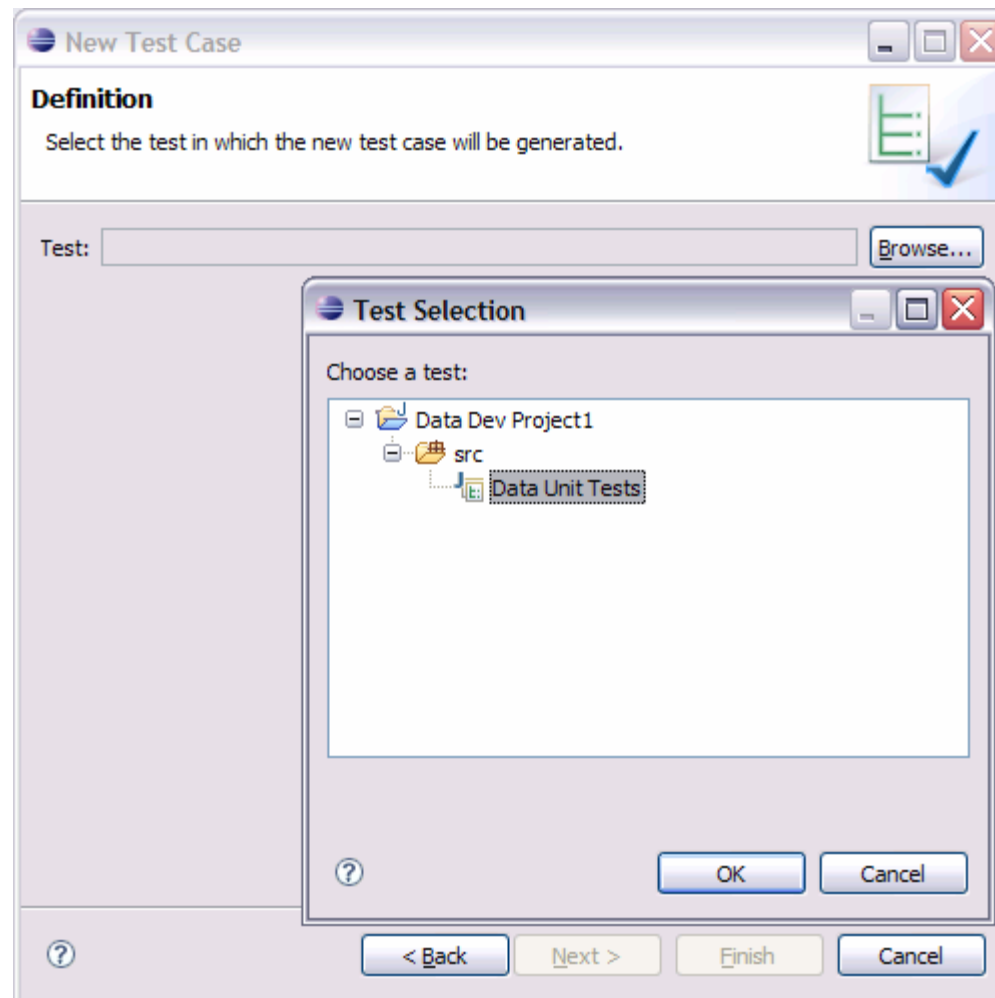
## Use Case – create a database unit test case

- Right click on a stored procedure and Create Unit Testcase



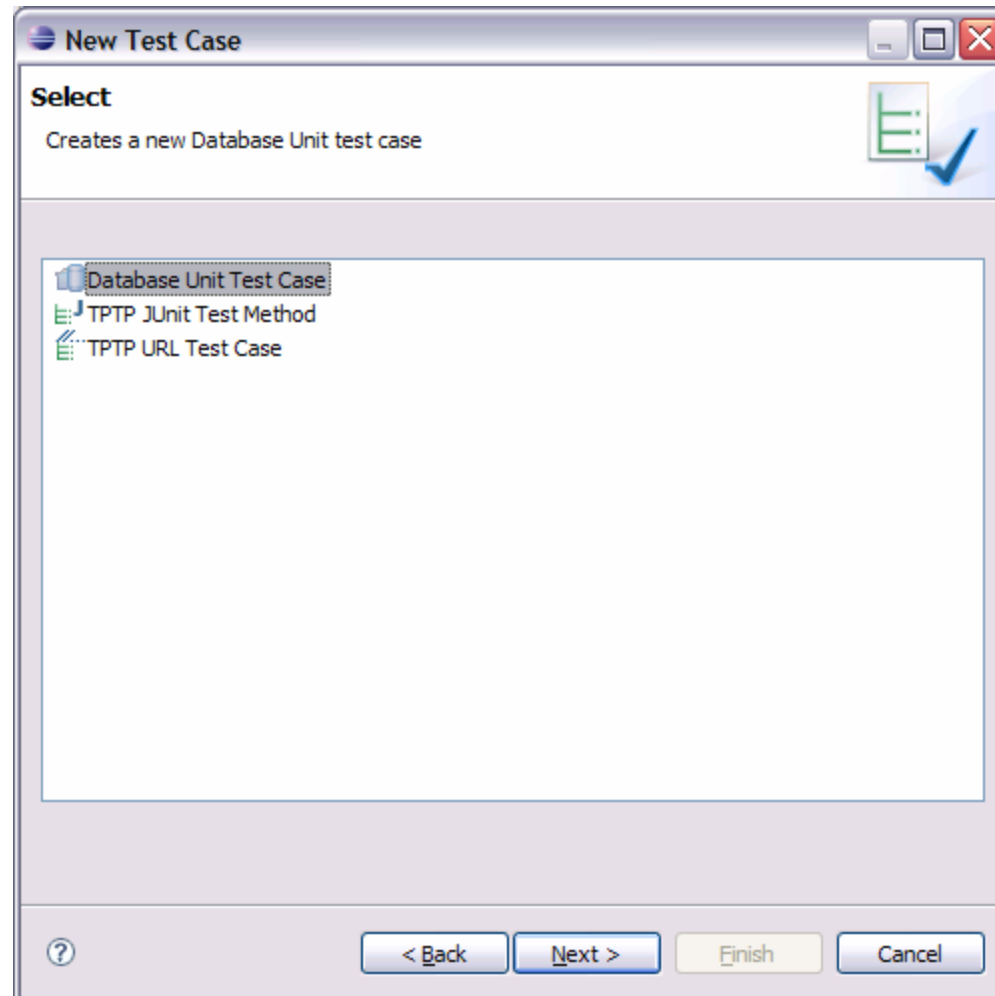
## Use Case – create a database unit test case

- 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

**New Database Unit Test Case**

Defines the name and description of the test case.

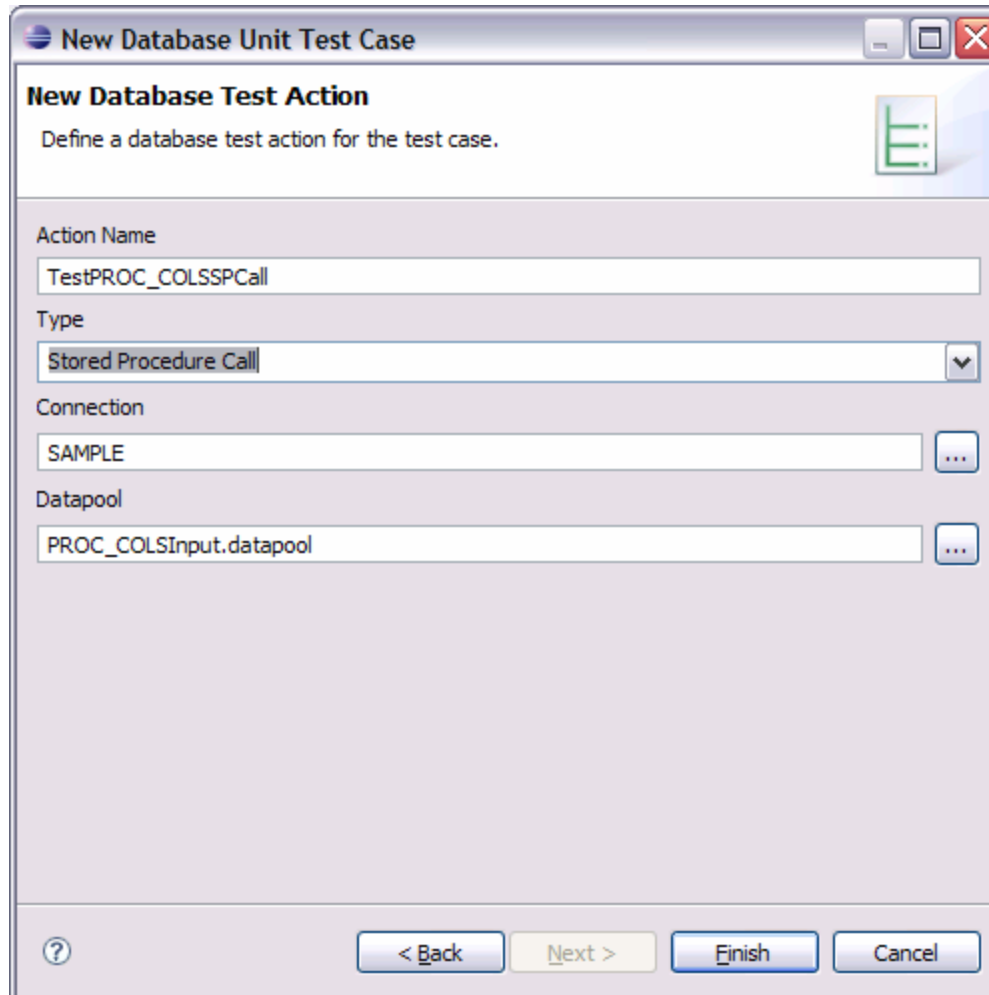
Name  
testPROC\_COLS

Description

< Back Next > Finish Cancel

## Use Case – create a database unit test case

- Define a database action for the test case
- Finish



**New Database Unit Test Case**

**New Database Test Action**  
Define a database test action for the test case.

Action Name  
TestPROC\_COLSSPCall

Type  
Stored Procedure Call

Connection  
SAMPLE

Datapool  
PROC\_COLInput.datapool

< Back   Next >   Finish   Cancel

## Use Case – create a database unit test case

- Test method sample

```
/**
 * 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

```
/**
 * 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 dpliterator = 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);
        dpliterator = dpFactory.open(datapool,
            "org.eclipse.hyades.datapool.iterator.DatapoolIteratorSequentialPrivate");
        dpliterator.dplInitialize(datapool,-1);
        stmt = conn.prepareStatement("CALL SAMPLE.PROC_COLS(?,?,?)");
    }
}
```

## Use Case – create a database unit test case

- Database test action method sample – cont.

```
((CallableStatement)stmt).registerOutParameter(3,java.sql.Types.INTEGER);
while (!dpIterator.dpDone()) {
    IDatapoolRecord dpRecord = dpIterator.current();
    Object cellValue;
    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);
    }
}
```

## Use Case – create a database unit test case

- Database test action method sample – cont.

```
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
  - <http://www.eclipse.org/tptp/>
  
- Agile Data
  - <http://www.agiledata.org/>
  
- Eclipse Data Tools Project
  - <http://www.eclipse.org/datatools/>
  
- Introduction to JET
  - [http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.emf.doc/tutorials/jet1/jet\\_tutorial1.html](http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.emf.doc/tutorials/jet1/jet_tutorial1.html)
  
- DbUnit Project
  - <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.