Testing and JDBC

Ralf Goller
Christian Hochhold
Andreas Jagersberger
Martin Luksch

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

Introduction

- JDBC is a database connector
- Article by Richard Dallaway „Unit Testing Database Code“
- 4 types of databases:
  - production
  - local development
  - populated development
  - development or integration

Introduction

- database access in code → problems
  - execution time
  - code duplication
- database access by using
  - query builder
  - query executer
- No more testing JDBC provider.
  Test your code!

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

Test making domain objects from a ResultSet

Two things can go wrong when executing a query with JDBC:

- wrong SQL Command
- unmarshal data into objects incorrect

Test making domain objects from a ResultSet

unmarshal data into objects incorrect

create a resultSet object with known data , then invoke “make domain objects” method and compare the result with the expected values

JDBC has no standalone implementation of ResultSets, therefore mock objects are used.

- MockMultiRowResultSet
- MockSingleRowResultSet

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

Verify your SQL Commands

"Best way to verify is to try them out a few times"

"Golden Master": verify test unmarshalby hand and use as a baseline for future tests.

```
assertEquals(
    "insert into catalog.beans "
    + "(productID, coffeeName, unitPrice) values "
    + "(?,?,?)",
    store.getAddProductSqlString());
```

Verify your SQL Commands

externalize SQL Commands to a file.

if db schema changes you have to change just one single file and rerun tests.

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

Test your database schema

verify nullable columns, indices, foreign key constraint and triggers

2 ways of testing:

- make assertions on database meta data
- performing queries and check results

Test your database schema

make assertions on database meta data

```
dbmetadata = connection.getMetaData();
schemaRS = dbmetadata.getSchemas();
schemaNames = new LinkedList();
//fetch in list
assertTrue(schemaNames.contains(“CATALOG”);
```

verified that there is a CATALOG schema in the database

Test your database schema

running test against a live database

to achieve test isolation db has to be cleaned before each test.

2 consequences:

- Table dependencies grow quickly, because of foreign keys the order in which tables can be delete is important
- The database is an exclusive external resource, the more test you run against the db, the slower testing gets.

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

Verify your tests clean up JDBC resources

avoid leaking resources by ensuring that tests clean up themselves.

checklist:

- create the data source in setUp() method
- allocate connections in setUp()
- create collection in setUp() to store result sets, statements and connections – one collection for each kind of resource
- if you create a JDBC resource in the test, add it to the collection
- in tearDown() method, invoke close() to all resources of the collections

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

Testing Stored Procedures

- You need to test stored Procedures against a live database
- Use shell scripts. JUnit is too verbose for such a simple task.
- Verify that stored procedures are invoked by using Mock objects.

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

Manage external data in the test fixture

- Testing against a database can bring it in a different state.
- Add or remove data for each test individually.
- To reach the goal:
  - setUp() connects to the database and prime it with data
  - tearDown() deletes all data from the Database

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

Unit tests for DBapplications

- Business Logic Layer

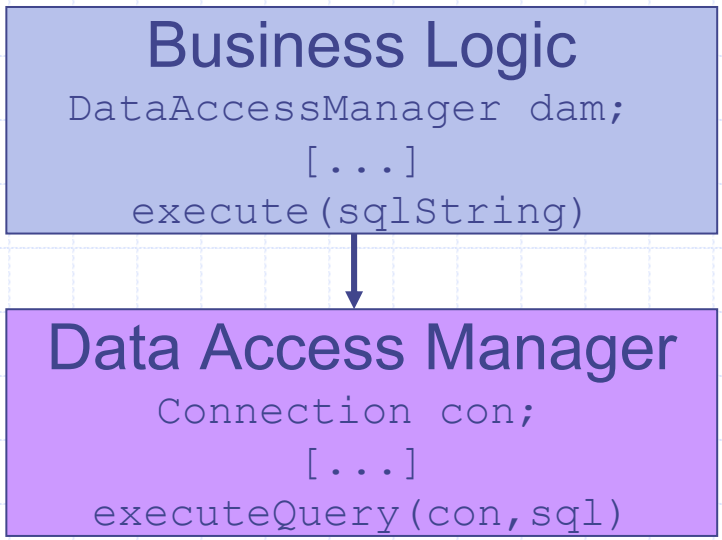
Business Logic

```
DataAccessManager dam;
[...]
execute(sqlString)
```

- fetches parameters for query
- brings parameters together
- builds a SQL-query

Unit tests for DBapplications

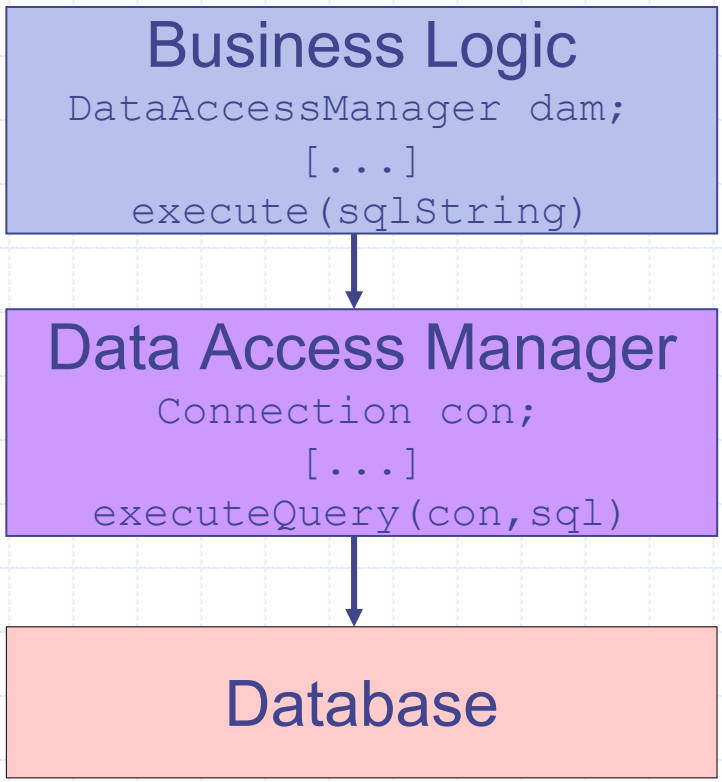
- Persistence Layer



- opens a connection (e.g.JDBC)
- sends query & receives result set
- closes connection

Unit tests for DBapplications

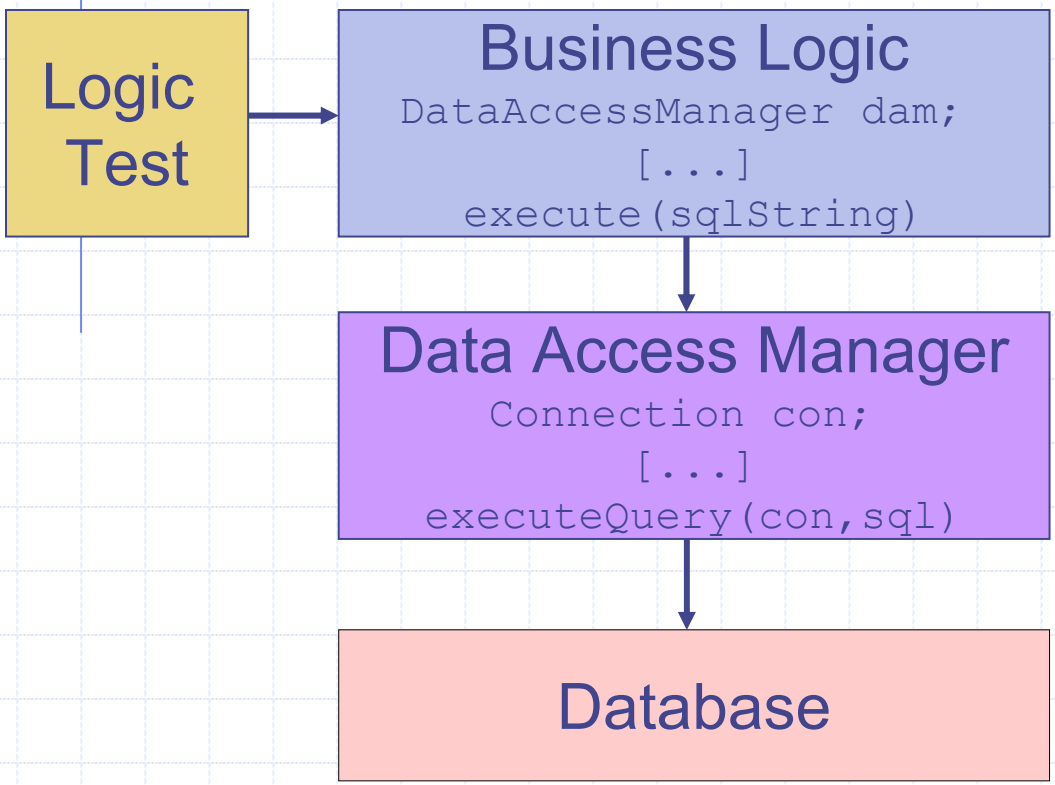
- Database Layer



- executes queries, triggers, stored procedures etc.
- returns result set

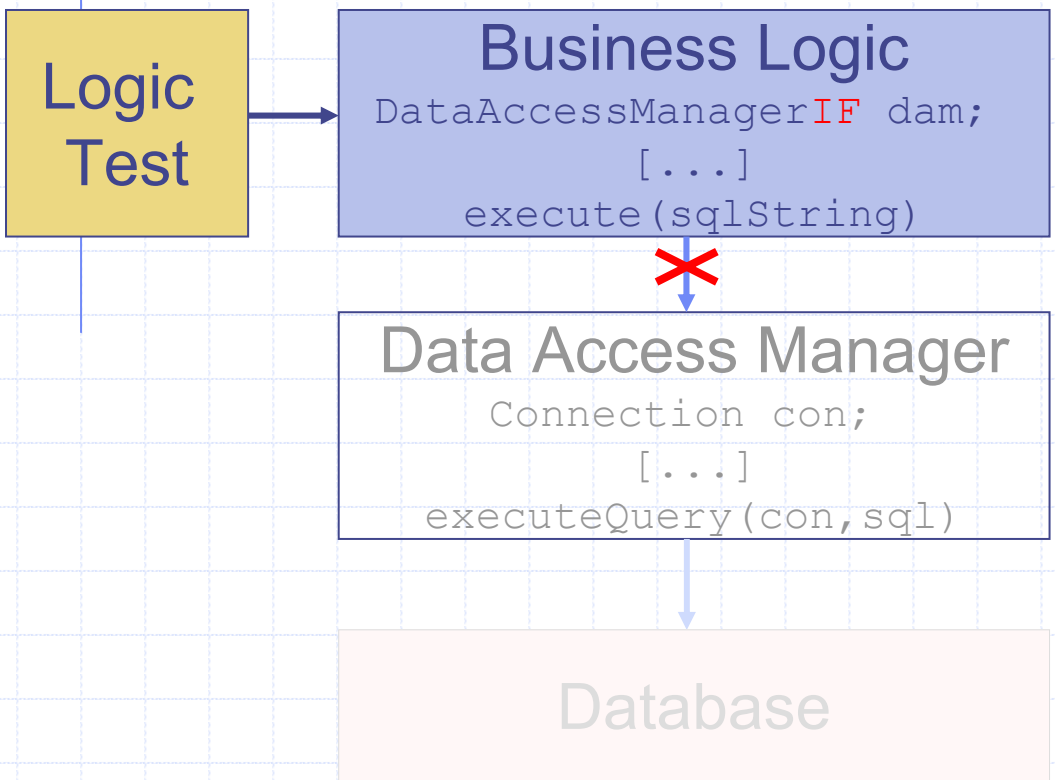
Unit tests for DBapplications

- Business Logic Layer Test



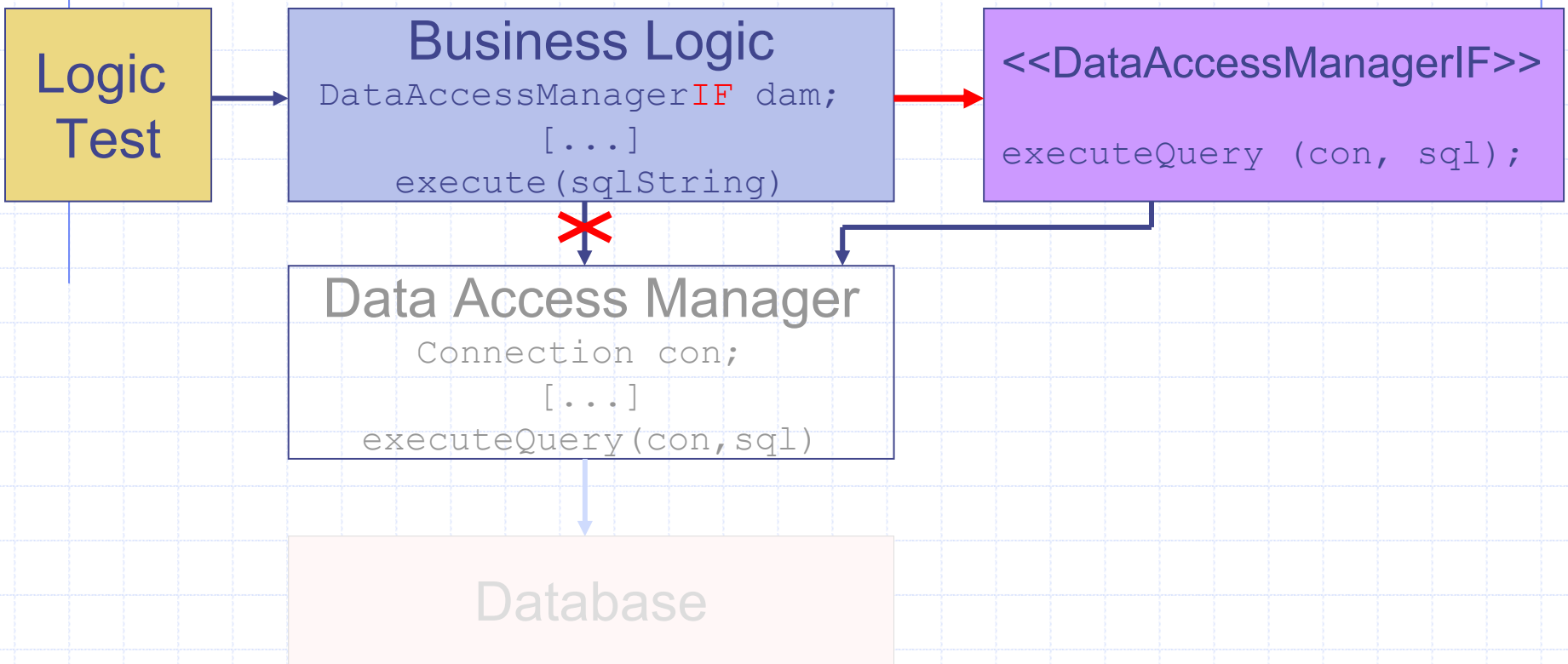
Unit tests for DBapplications

- Business Logic Layer Test



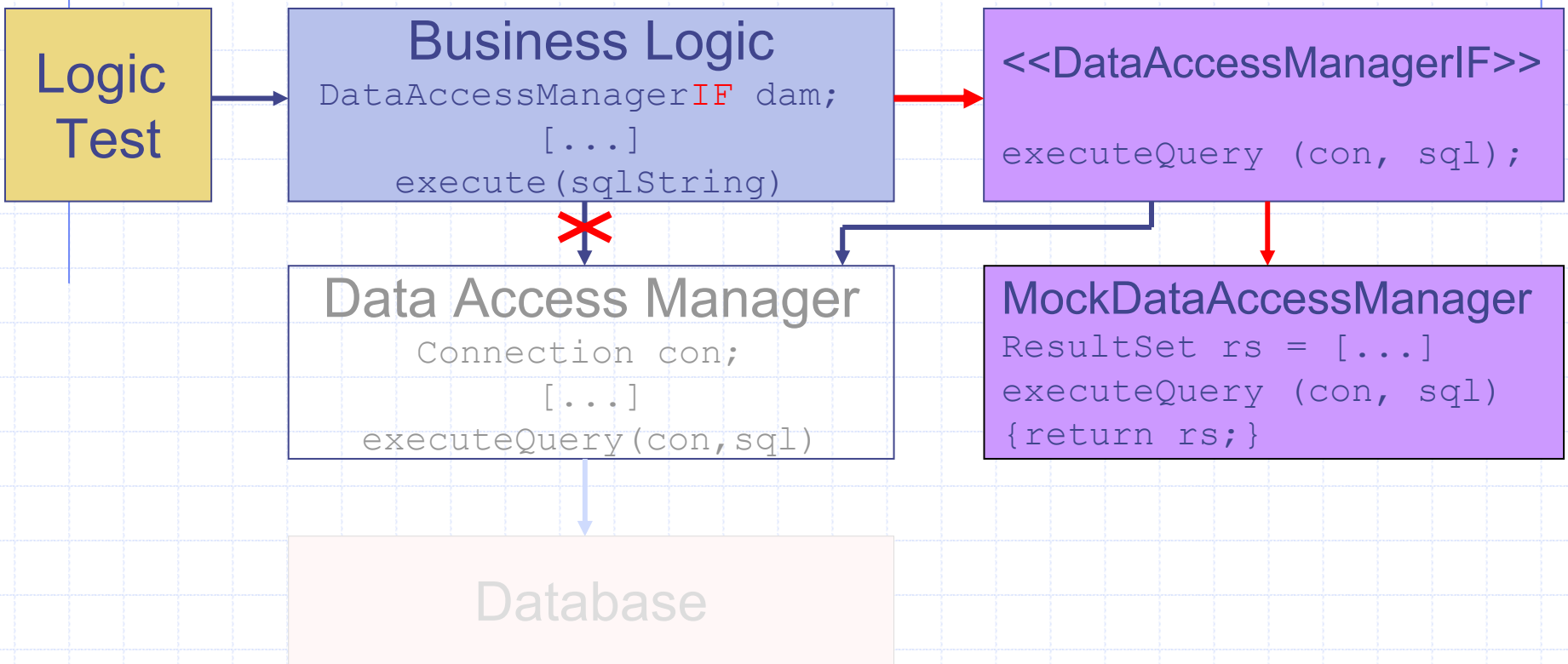
Unit tests for DBapplications

- Business Logic Layer Test



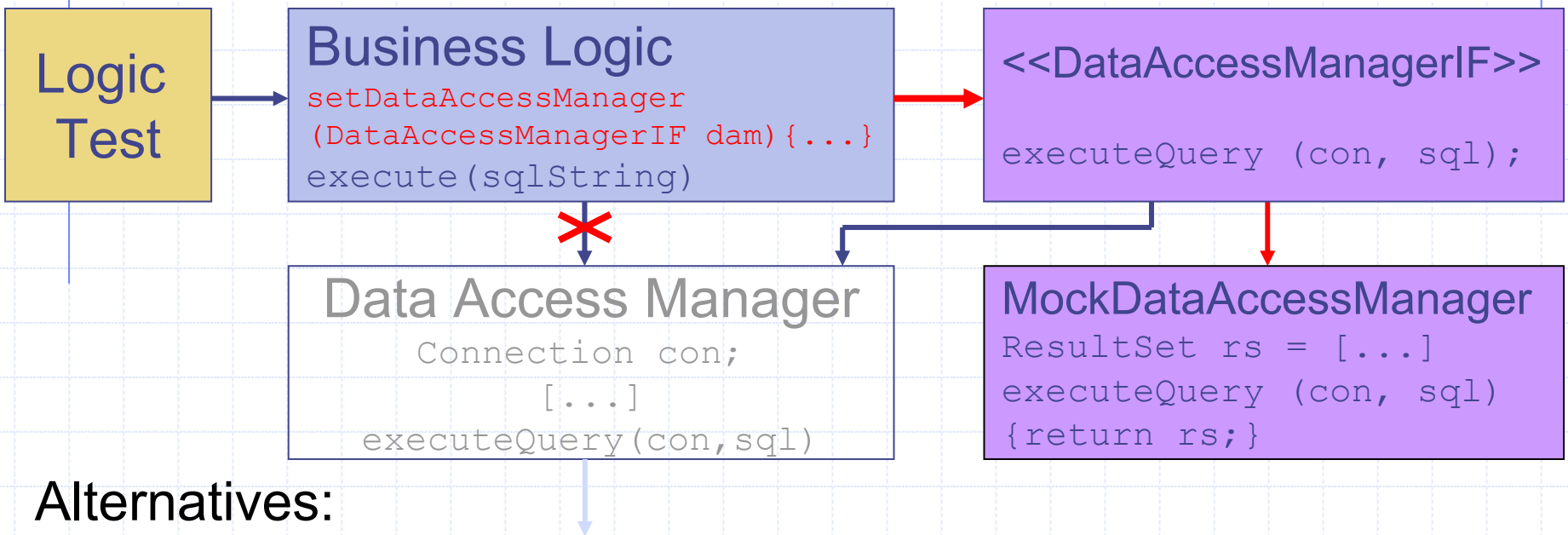
Unit tests for DBapplications

- Business Logic Layer Test



Unit tests for DBapplications

- Business Logic Layer Test



Alternatives:

-Logic constructor accepts DAM-Interface as parameter

-Subclass of Logic class overriding execute-method

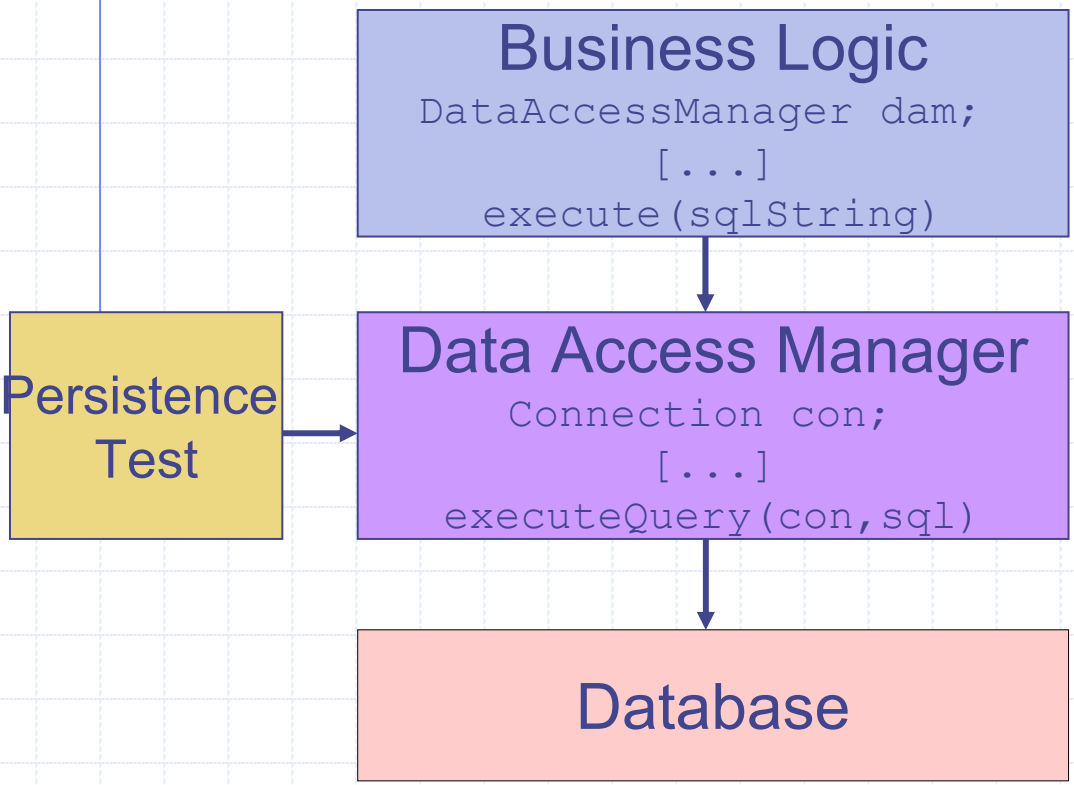
-DAM as parameter of application (e.g. in web.xml)

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

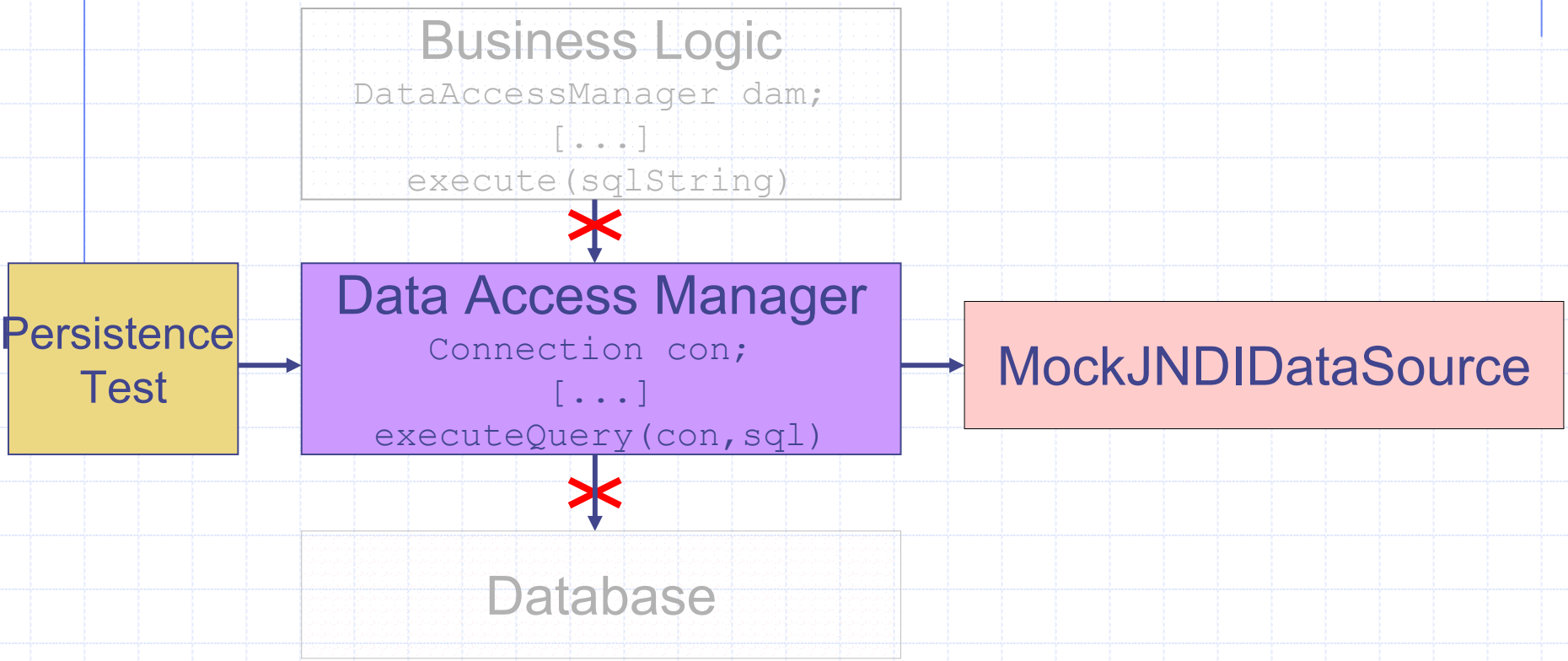
Unit tests for DBapplications

- Persistence Layer Test



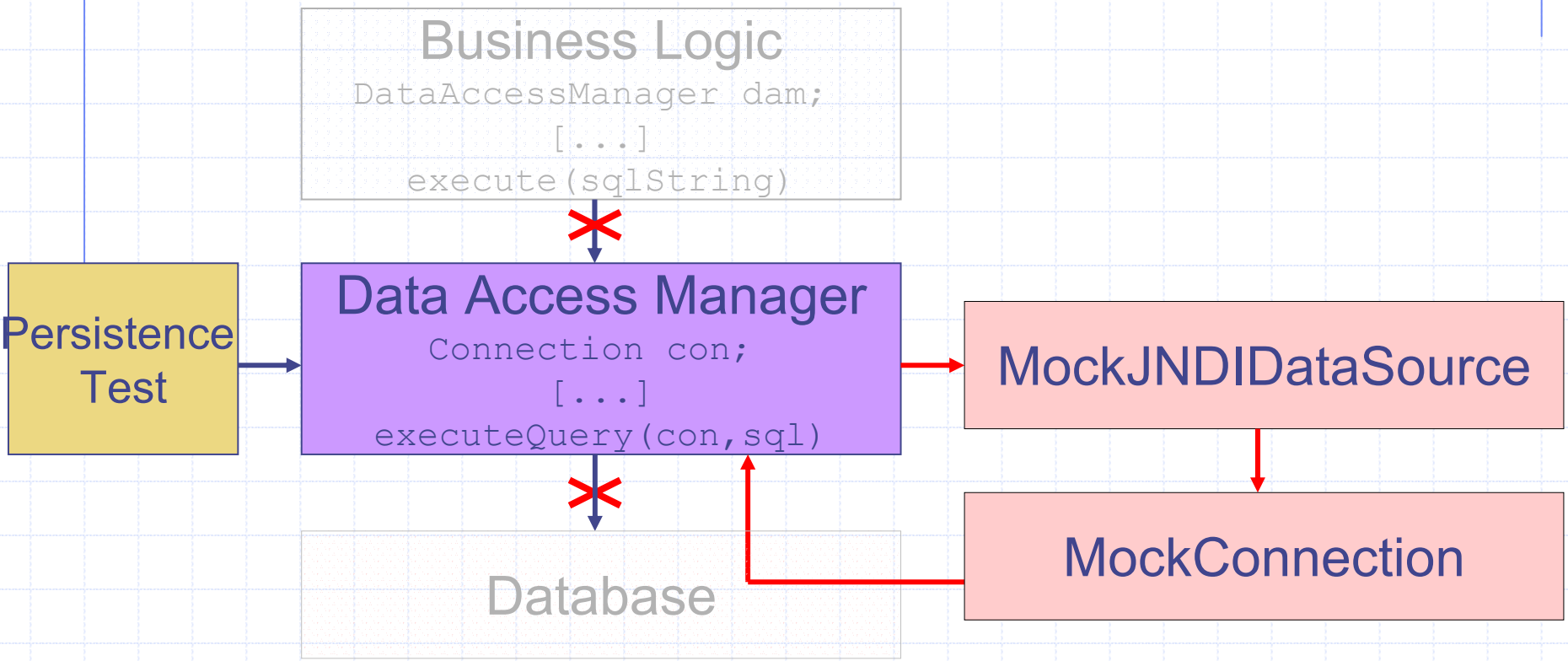
Unit tests for DBapplications

- Persistence Layer Test



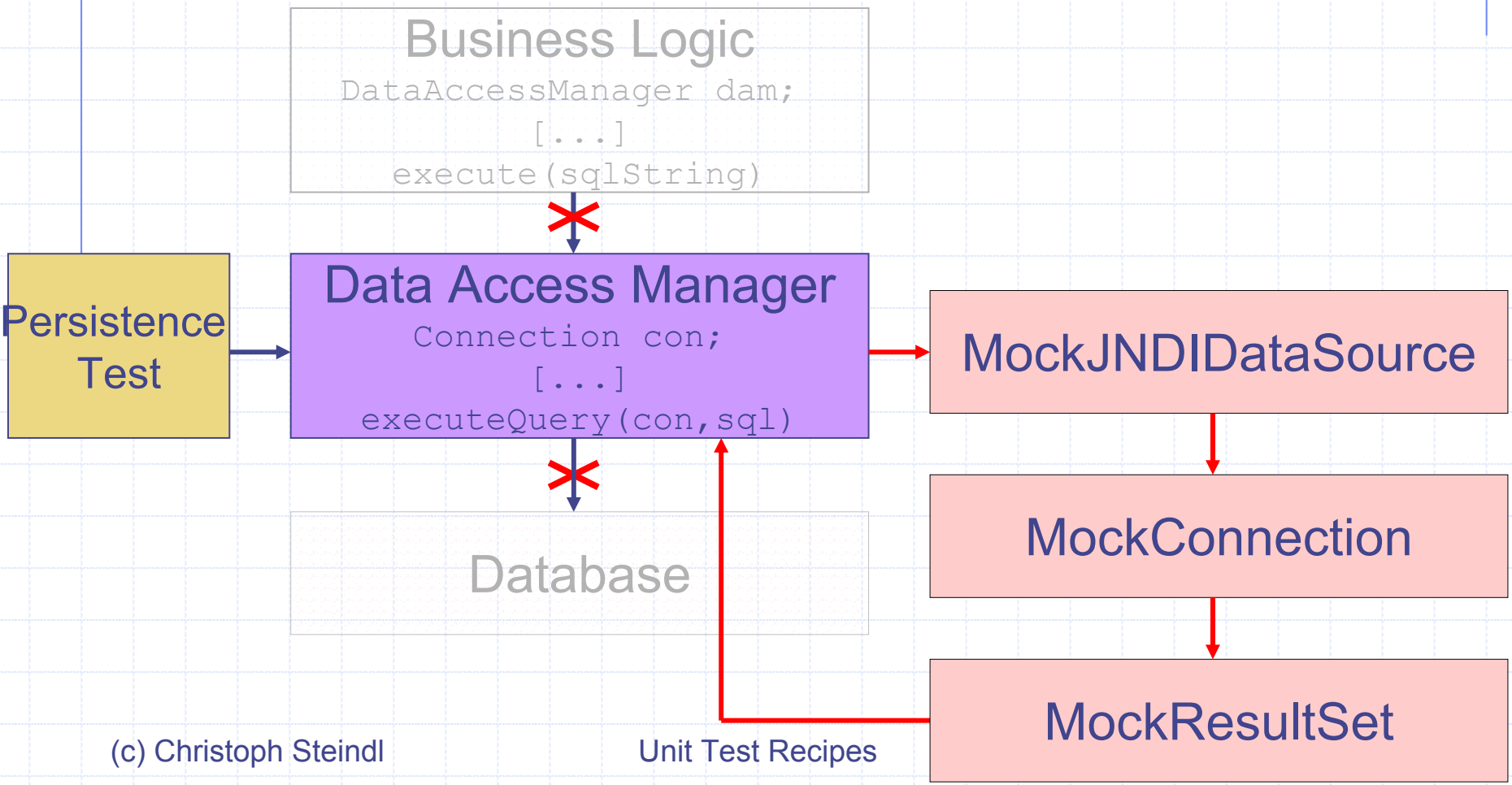
Unit tests for DBapplications

- Persistence Layer Test



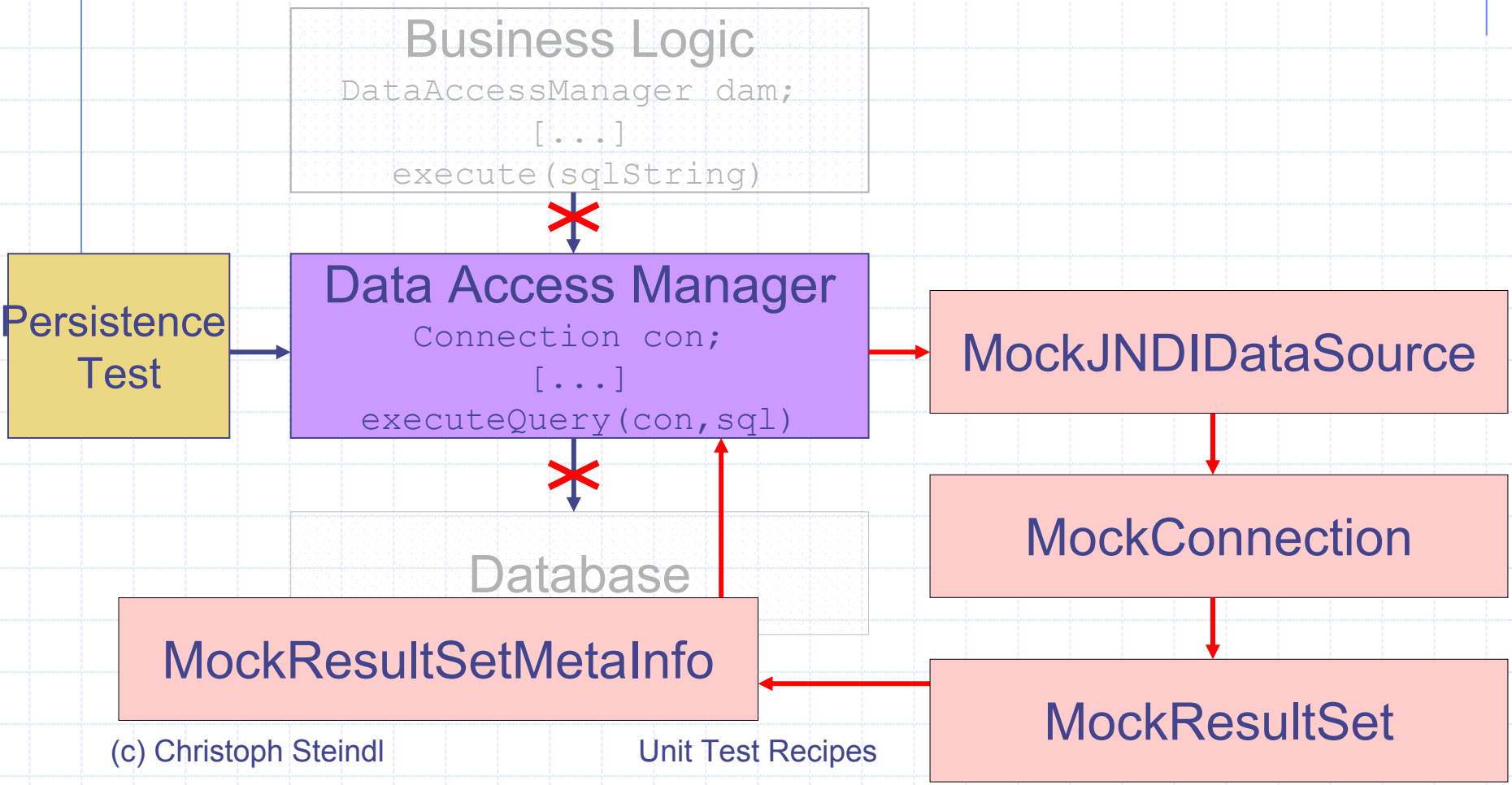
Unit tests for DBapplications

- Persistence Layer Test



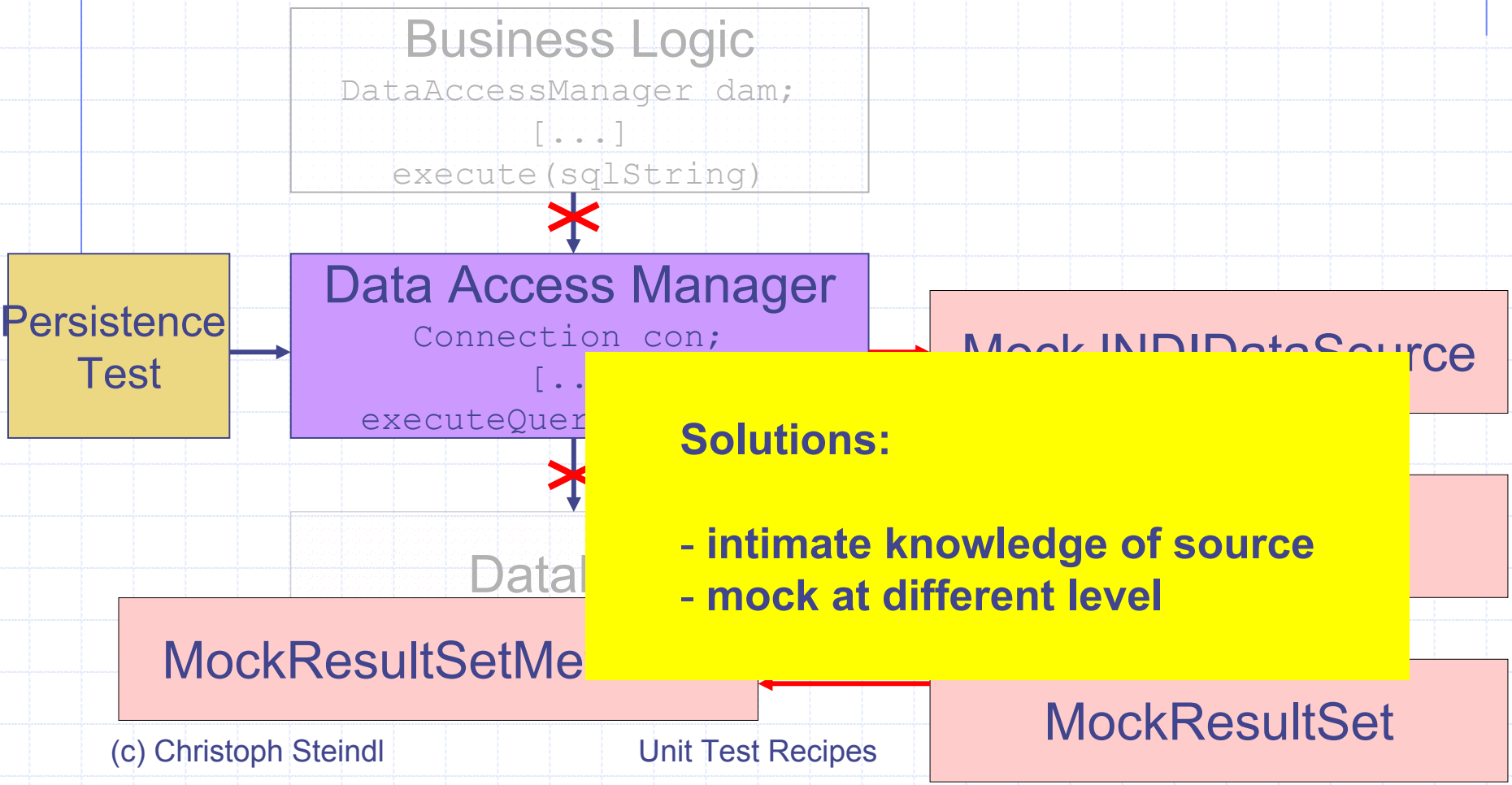
Unit tests for DBapplications

- Persistence Layer Test



Unit tests for DBapplications

- Persistence Layer Test

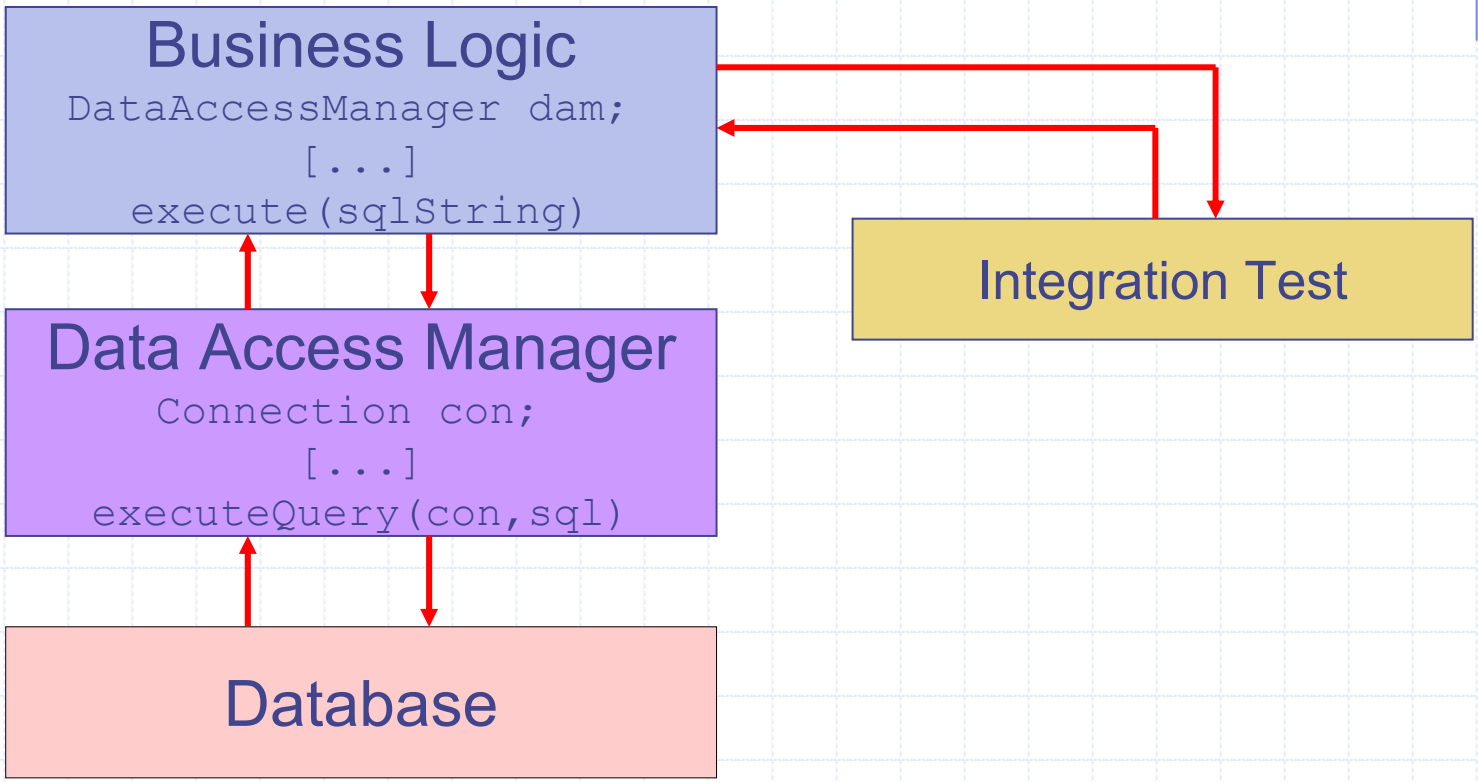


Content

- Introduction
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

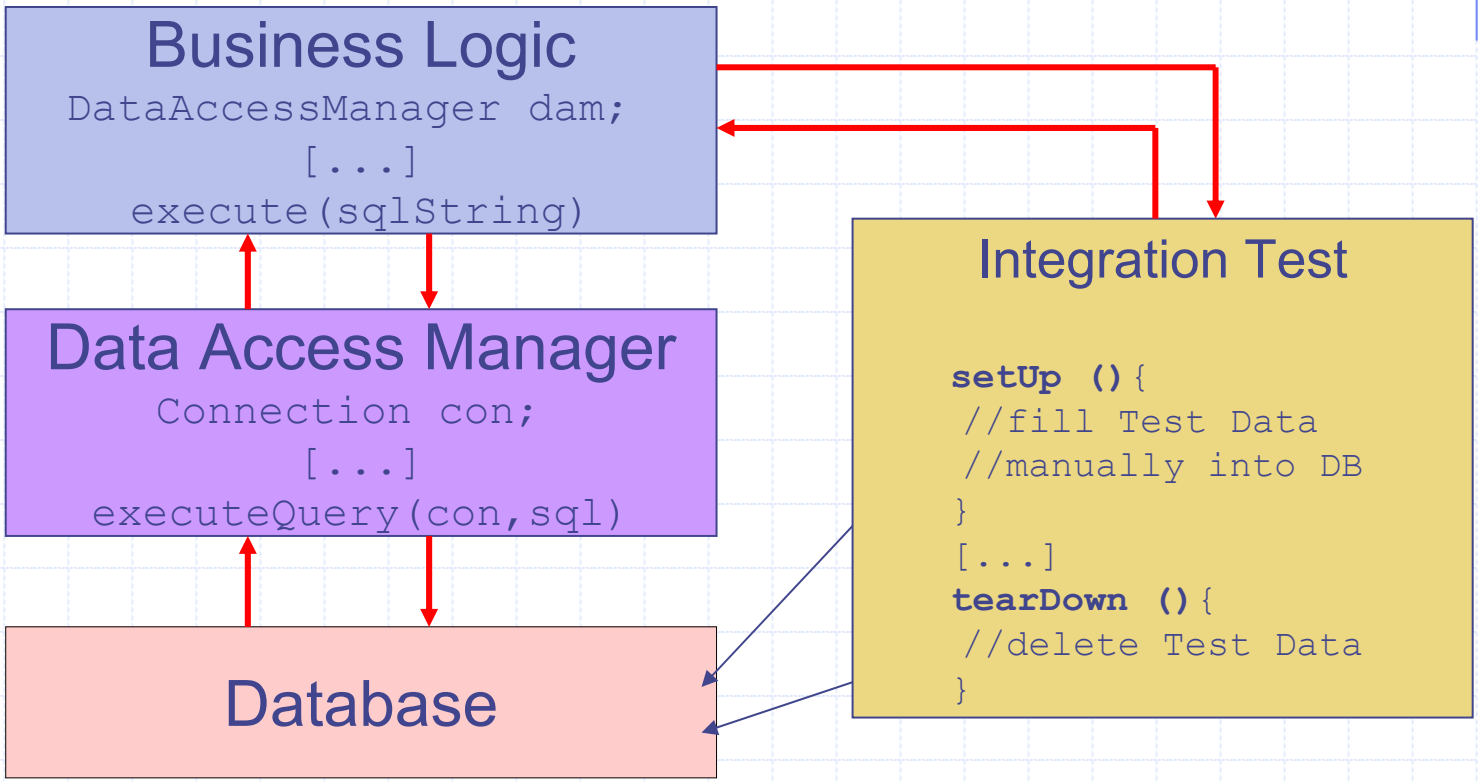
Unit tests for DBapplications

- Database Layer – integration tests



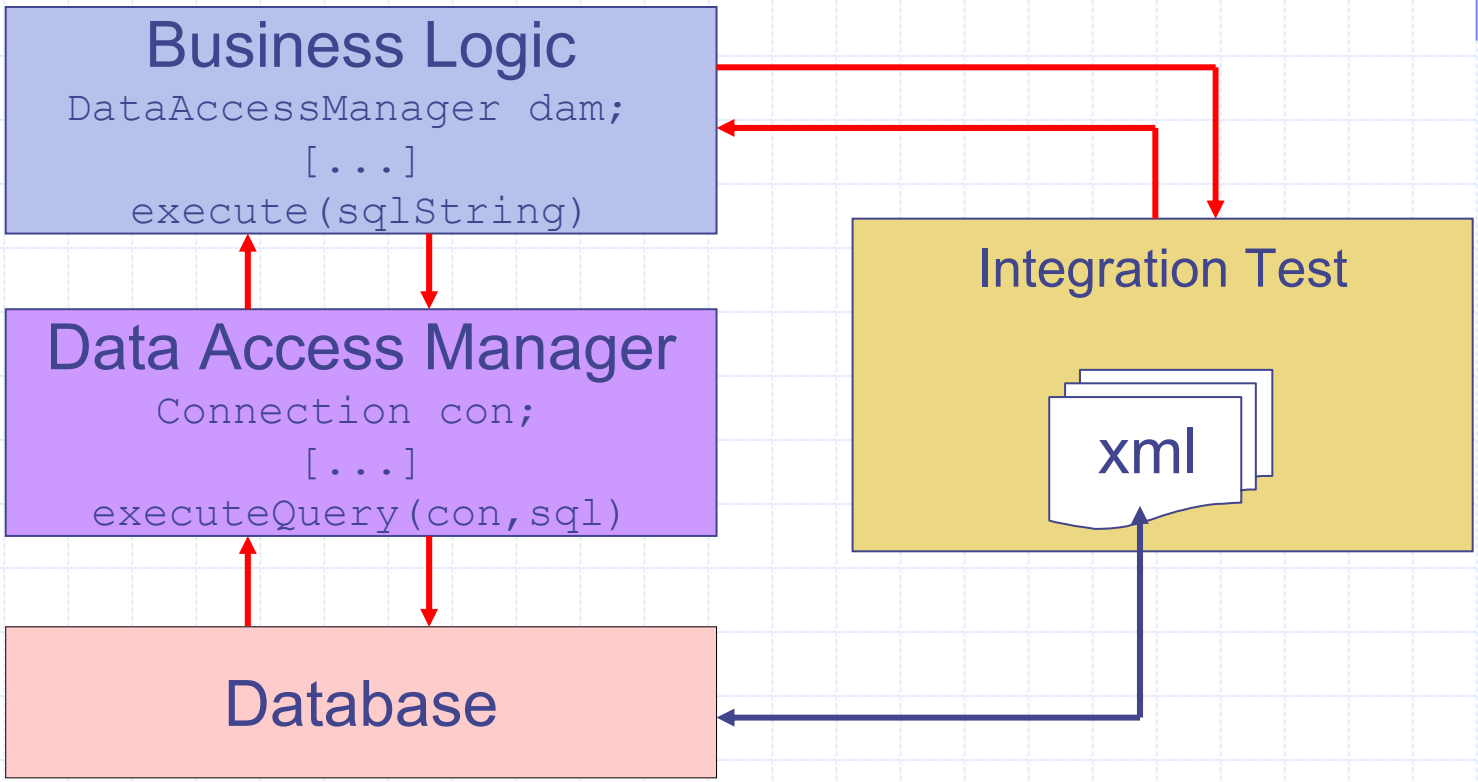
Unit tests for DBapplications

- Database Layer – integration tests



Unit tests for DBapplications

- Database Layer – integration tests



Unit tests for DBapplications

- Mocking Frameworks:

EasyMock: http://www.easymock.org/

DynaMock: http://www.mockobjects.com/DynaMock.html

MockObjects: http://sourceforge.net/projects/mockobjects

- Predefined MockObjects like ResultsetMock, StatementMock,...
- Predefined mechanism for expectations
  MockObject.addExpectedXXX [MockConnection.addExpectedQuery(String qy)]
  MockObject.setExpectedXXX [MockConnection.setExpectedCloseCalls(1)]

- DB integration test Frameworks:

- Cactus: http://jakarta.apache.org/cactus/
- DBUnit: http://dbunit.sourceforge.net/

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

Running the Cactus test using Ant

- Cactus is a simple test framework for unit testing server-side java code
- uses JUnit and extends it
- Cactus/Ant integration module
- Ant build file

- http://jakarta.apache.org/cactus

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

Tuning for build performance

- strategies
  - Factor out read-only data
  - Grouping test in functional test suites
  - Using an in-memory database
- Overall database unit-testing strategy
  - Mock objects, functional testing
  - Mock objects, database integration unit tests, functional tests

Content

- Introduction
- Test making domain objects from a ResultSet
- Verify your SQL Commands
- Test your database schema
- Verify your tests clean up JDBC resources
- Testing Stored Procedures
- Manage external data in the test fixture
- Testing business logic isolated from Databases
- Testing persistence code isolated from Databases
- Writing Database integration tests
- Running the Cactus test using Ant
- Tuning for build performance
- Summary

Summary

- No more testing JDBC provider. Test your code!