Fixture Management & Data Component Testing

Mike Nolan
mnolanjr@gmail.com

Module Overview

- Fixture management
- Data challenges when testing a database
- DBUnit

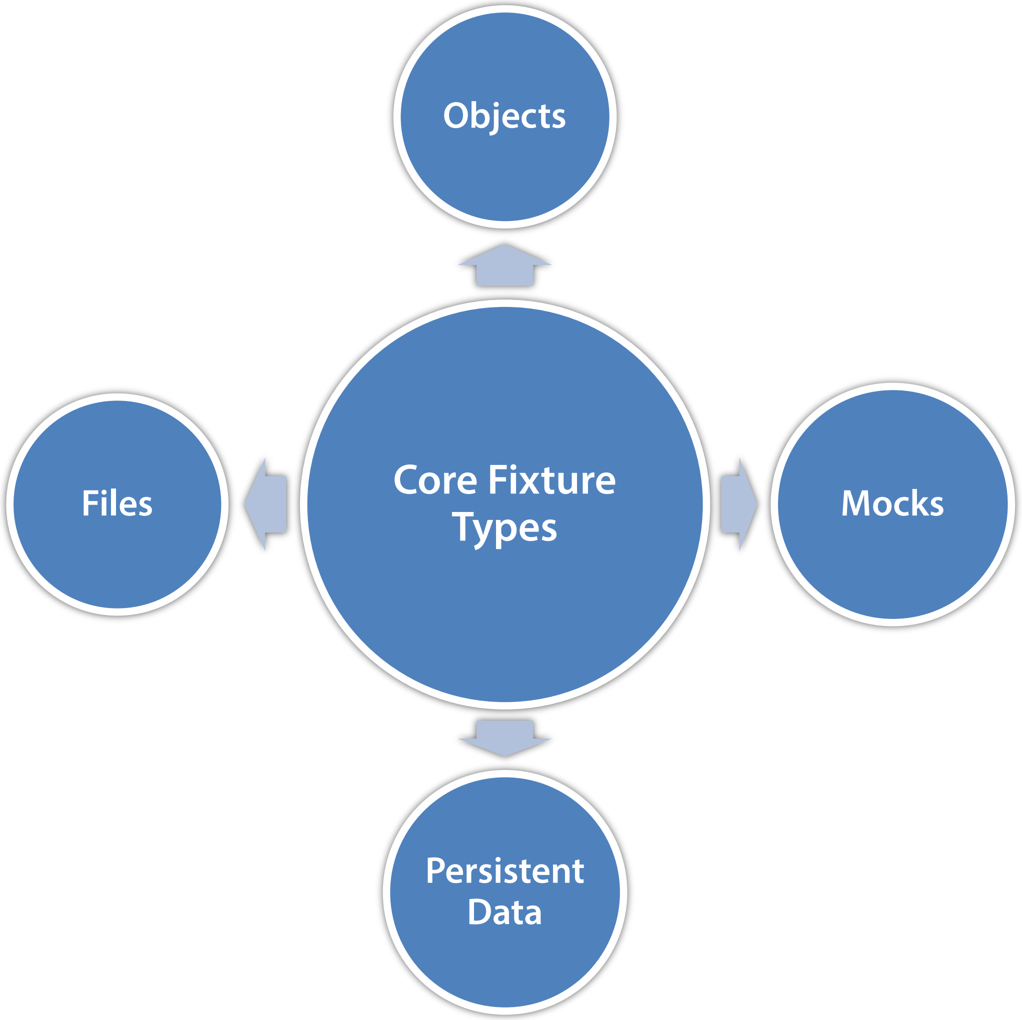
Fixture Management

Start Clean / Run Independentley

- **Fixture state initialization performed for each test**
  - Instantiate objects to pass into methods
  - Declare mock stubs / Initialize objects they return
  - Insert data in RDBMS for data access tests
  - Create files

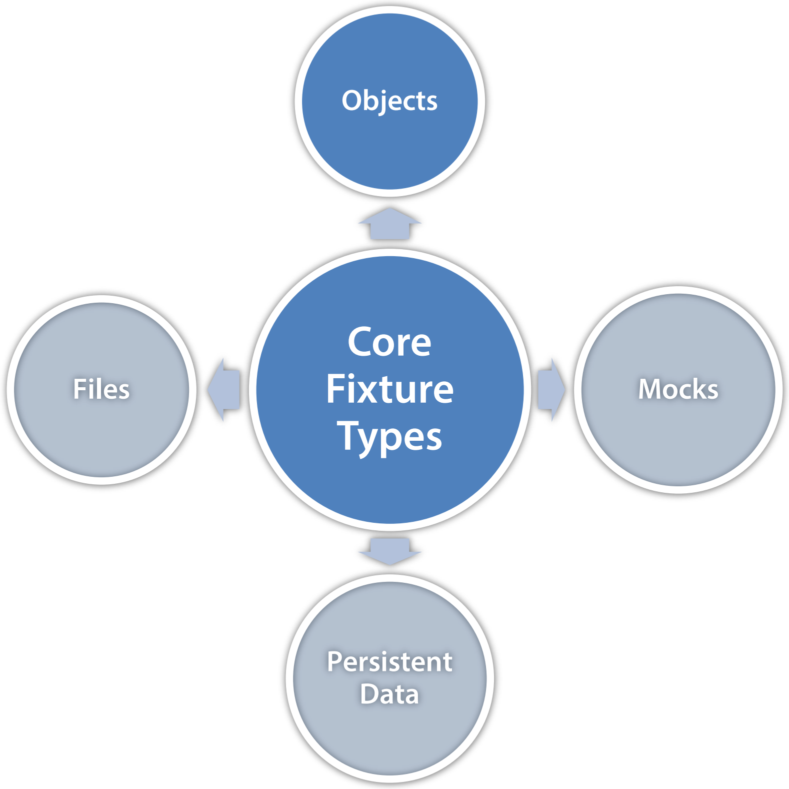
- **Teardown anything not purged by JVM Garbage Collector**
  - Data inserted into a database
  - Files that were manipulated

Core Fixture Types To Manage

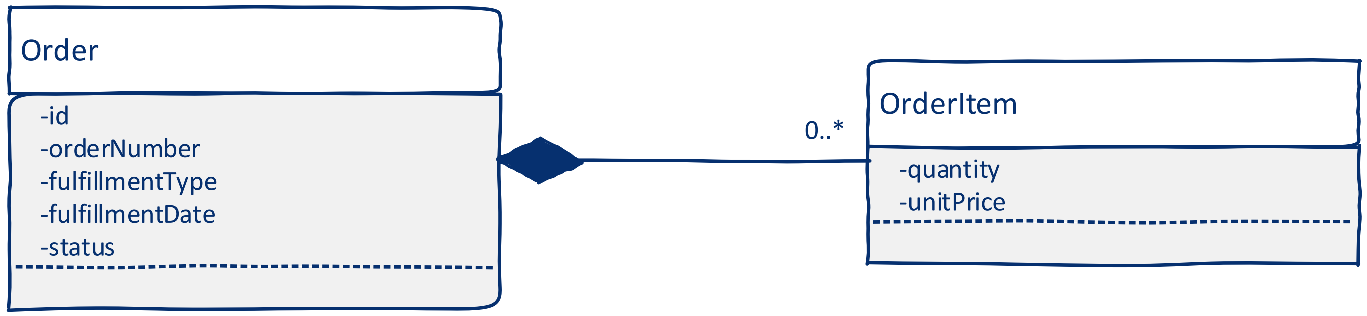


Managing Object Fixtures

- Only setup minimal amount of data



Limit Initialization Scope



Limit Initialization Scope

Functional Code

```
public void routeOrder(Order order) {

    if (“dropship”.equals(order.getFulfillmentType())) {

        routeOrderToDropshipper(order);
    }
    else if (“direct”.equals(order.getFulfillmentType())) {
        routeOrderToWarehouse(order);
    }
    else {
        // …
    }
}
```

Limit Initialization Scope

Fixture Setup In Test Class

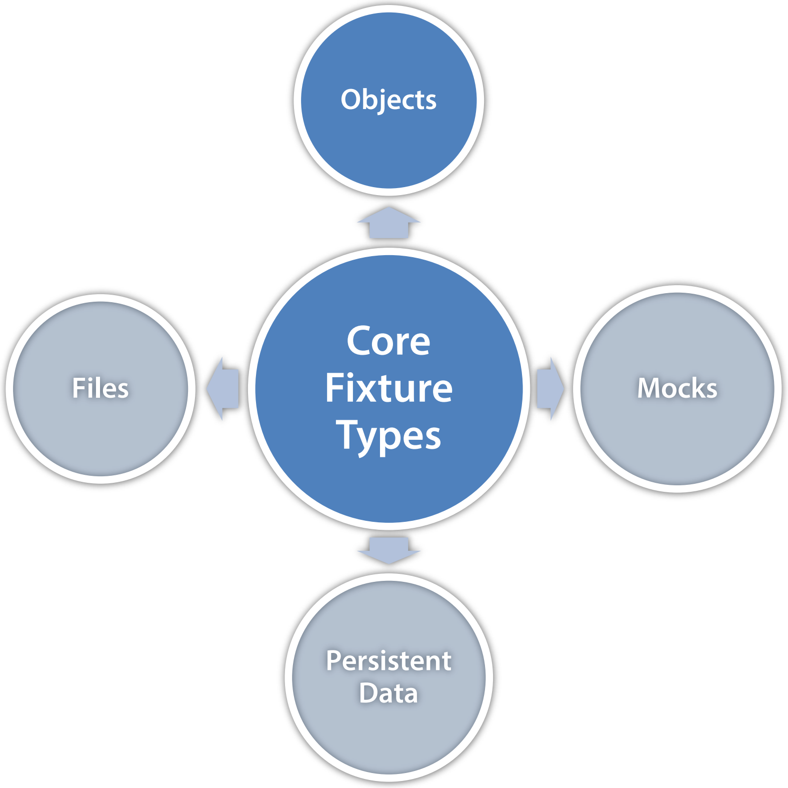
```
// Do!!!
Order orderFixture = new Order();
orderFixture.setFulfillmentType(“dropship”);

// Don’t
Order orderFixture = new Order();
orderFixture.setId(1);
orderFixture.setOrderNumber(“123455”);
orderFixture.setFulfillmentType(“dropship”);

OrderItem orderItemFixture = new OrderItem();
orderFixture.getOrderItems.add(orderItemFixture);
```

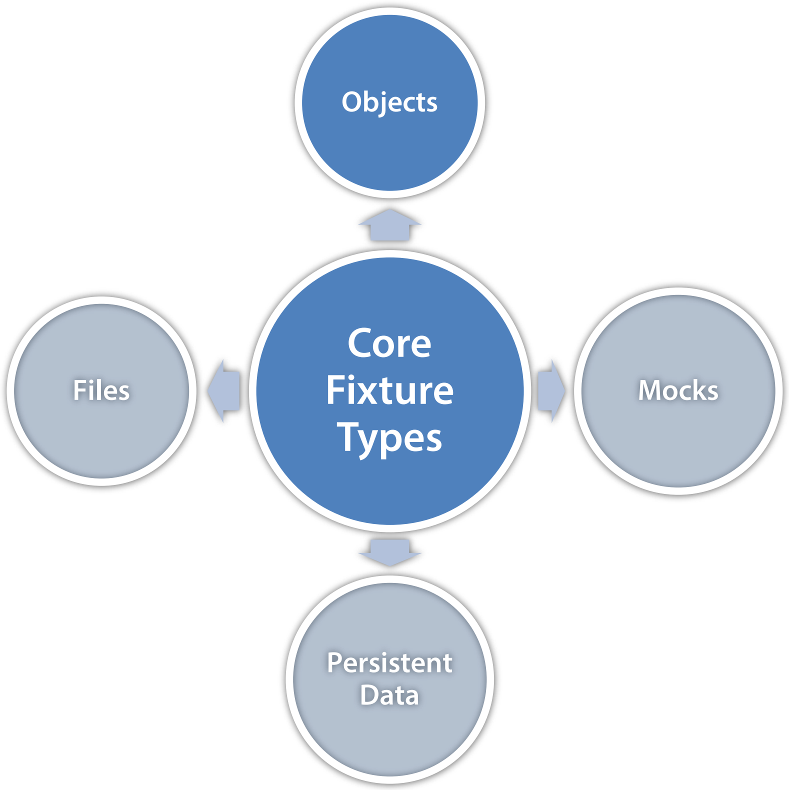
Managing Object Fixtures

- Only setup minimal amount of data
- Find balance between redundant setup and minimal field setup
- Consider a fixture factory or utility class



Managing Object Fixtures

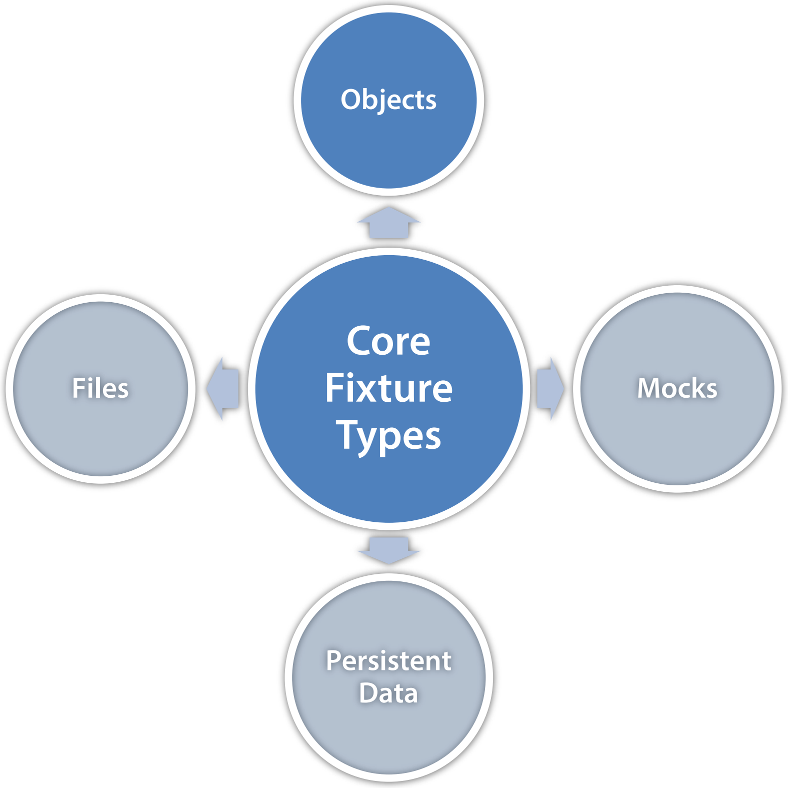
- Only setup minimal amount of data
- Find balance between redundant setup and minimal field setup
- Consider a fixture factory or utility class – at the cost of test readability



Managing Object Fixtures

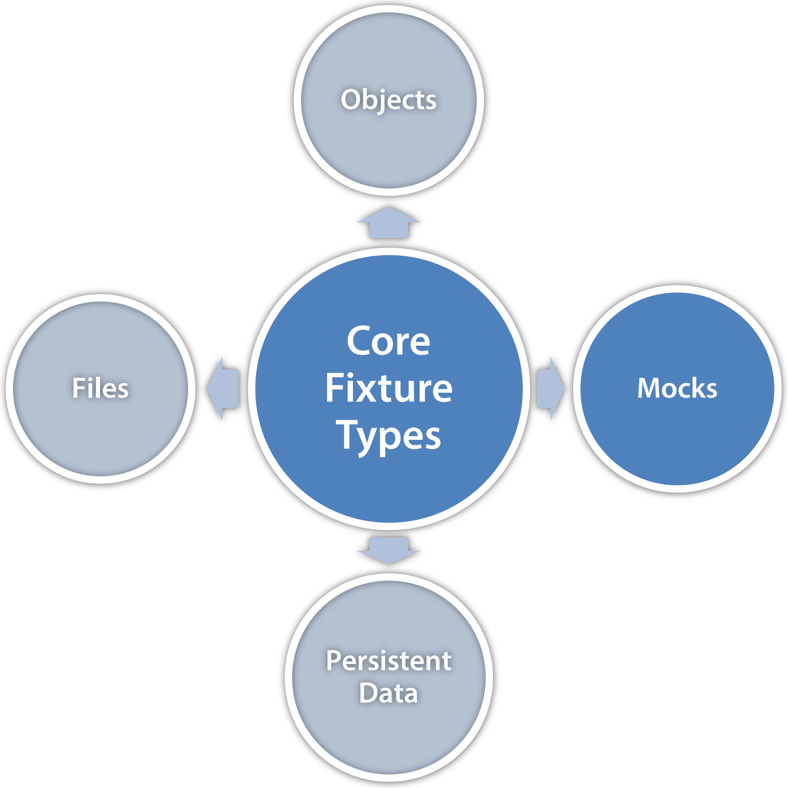
- Create fresh instances per test – don’t reuse an instance across tests

- Never undo the state of another test



Managing Mock Fixtures

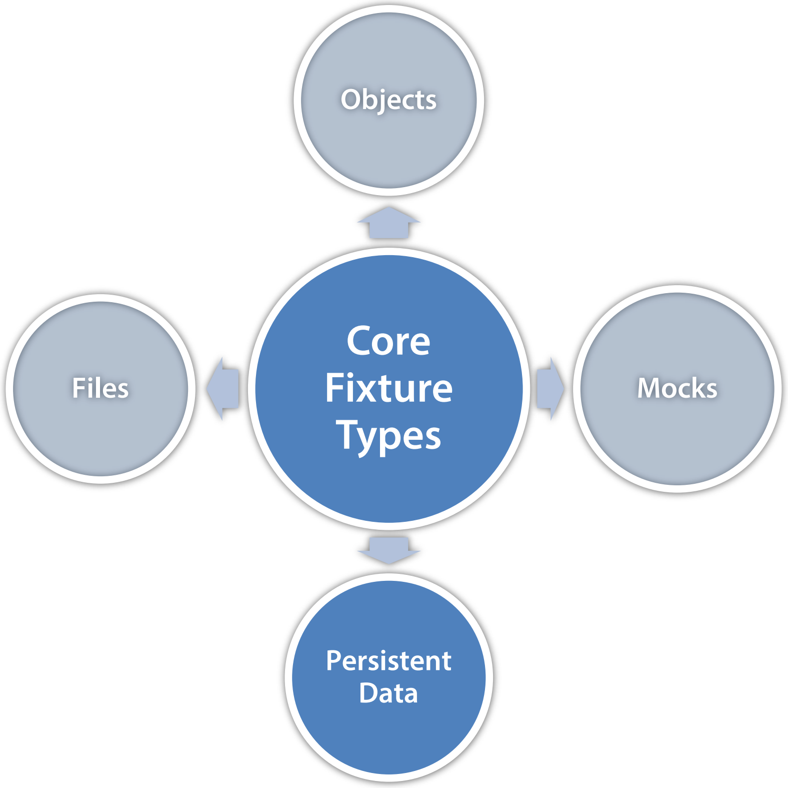
- Declaring mock as a variable vs. as a field in the test
- Mocking in setup methods (@Before) creates obscure tests
- Don't over-initialize fixtures returned from stubbed calls



Managing Persistent Data Fixtures

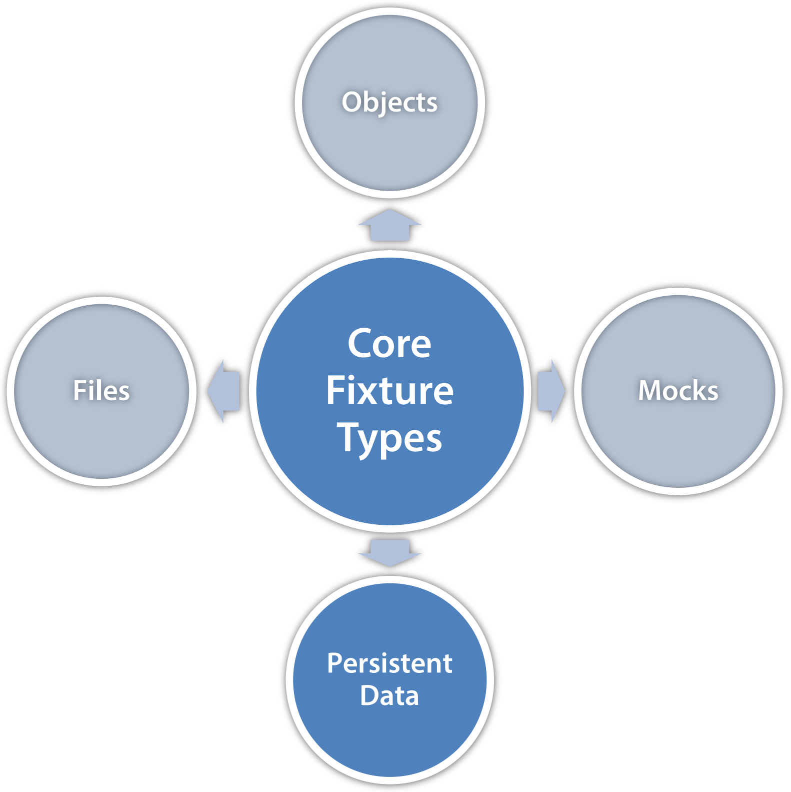
- **Persistent data introduces challenges**
  - Shared databases – commercial platforms are costly
  - Not everyone will have their own dedicated instance or schema

- **Unpredictable execution of tests & suits when a database is shared**



Managing Persistent Data Fixtures

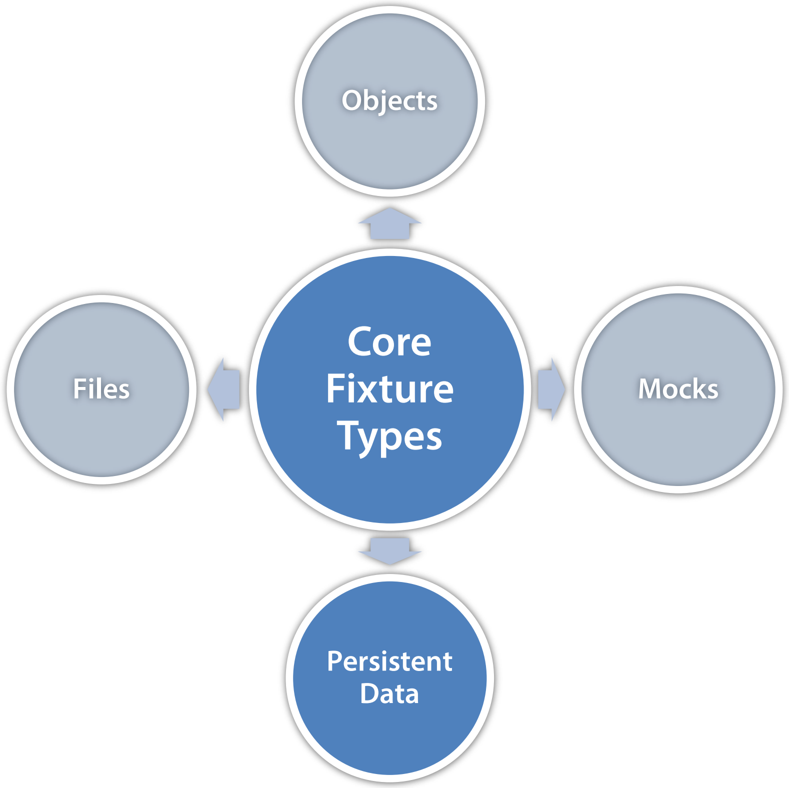
- **Tests that insert & manipulate data**
- **"Stealing" existing data is a bad idea**
  - Databases are periodically refreshed/cleaned
  - Data and date ranges may no longer match
  - Data manipulations may cause conflicts in your reads



Managing Persistent Data Fixtures

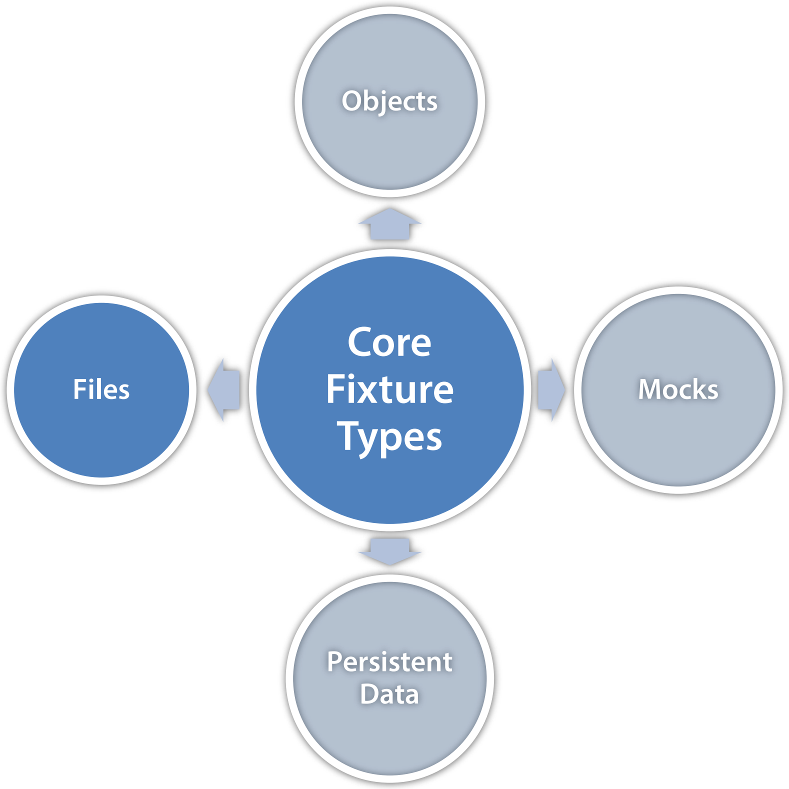
- **Ideal practices**
  - Dedicated database instance per developer
  - Setup and teardown data for each test scenario executed

- **DBUnit to the rescue!**



Managing File Resource Fixtures

- **Be sure to manage your files**
  - Store a base template to create copies
  - Have code generate a unique copy
- **Cleanup**
  - Ensure you remove any files created
  - Avoid manipulating template files
- **Use JUnit's @Rule and TemporaryFolder class to facilitate file cleanup**



Data Access Testing

Database Independence

- **Commercial databases may be cost prohibitive for unit testing**
- **Consider mixing in an open-source database for your unit testing**
- **H2 is a good option**
    - Supports in-process or out-of-process, and in-memory
    - JDBC-4 Driver
    - Open-source and no licensing cost

Weighing Your Options

- **Good to use when**
  - Running unit tests on workstation
  - You want a development database to run locally
  - The system is not heavily dependent on commercial RDBMS features (ie. Stored procedures)

+	−
Each developer can run in isolation, avoiding data and constraint conflicts	**You will be dealing with multiple database vendors, and may need to maintain multiple sets of DDL to accommodate**

Running H2

- **Prefer In-Memory mode for unit testing**
  - Ensures data is removed when tests are done
  - Test cleanup doesn’t run during abrupt process termination

- **Setup of tables is required per execution when running In-Memory**

- **H2’s RunScript class**

DBUnit Overview

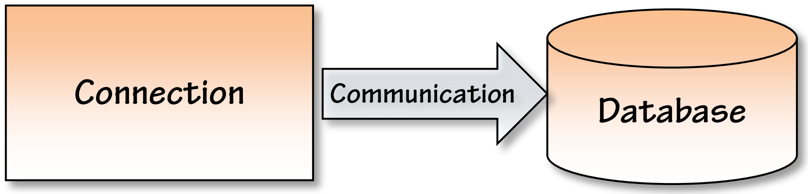
DBUnit Overview

- **DBUnit helps solve challenges of managing database test fixture data**

- **Three core abstractions**
  - Database connection
  - Data set management
  - Database operations

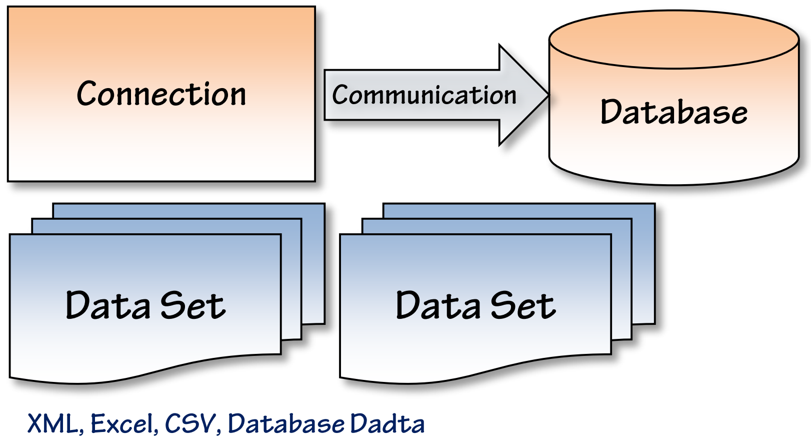
Core Components

- Connection – IDatabaseConnection



Core Components

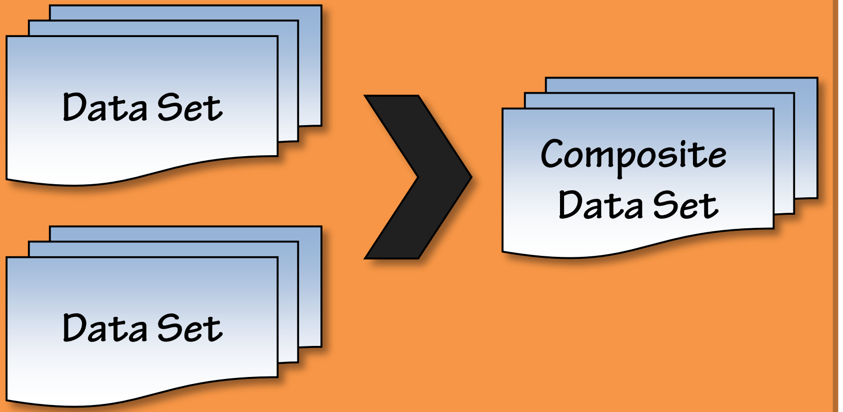
- Connection – IDatabaseConnection
- Data Set – IDataSet



Core Components

- Connection – IDatabaseConnection
- Data Set – IDataSet

- Collection of tables stored in-memory in JVM
- Implementations work with source data store to load into memory
- Sources
  - XML, Excel, CSV
  - Programmatically created
  - Database
- Composite Data Sets



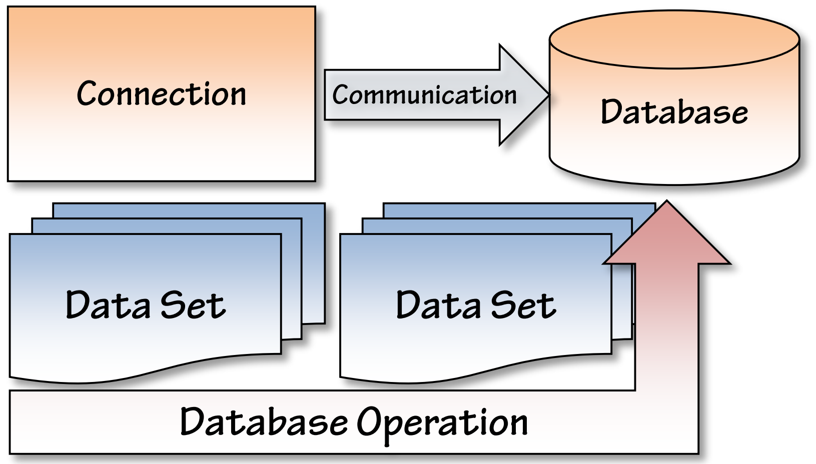
Core Components

- Connection – IDatabaseConnection
- Data Set – IDataSet

- Common scenarios –
  - Use data set(s) to source data for read query testing
  - Use two data sets when testing data transformations – one as source and one as expected result

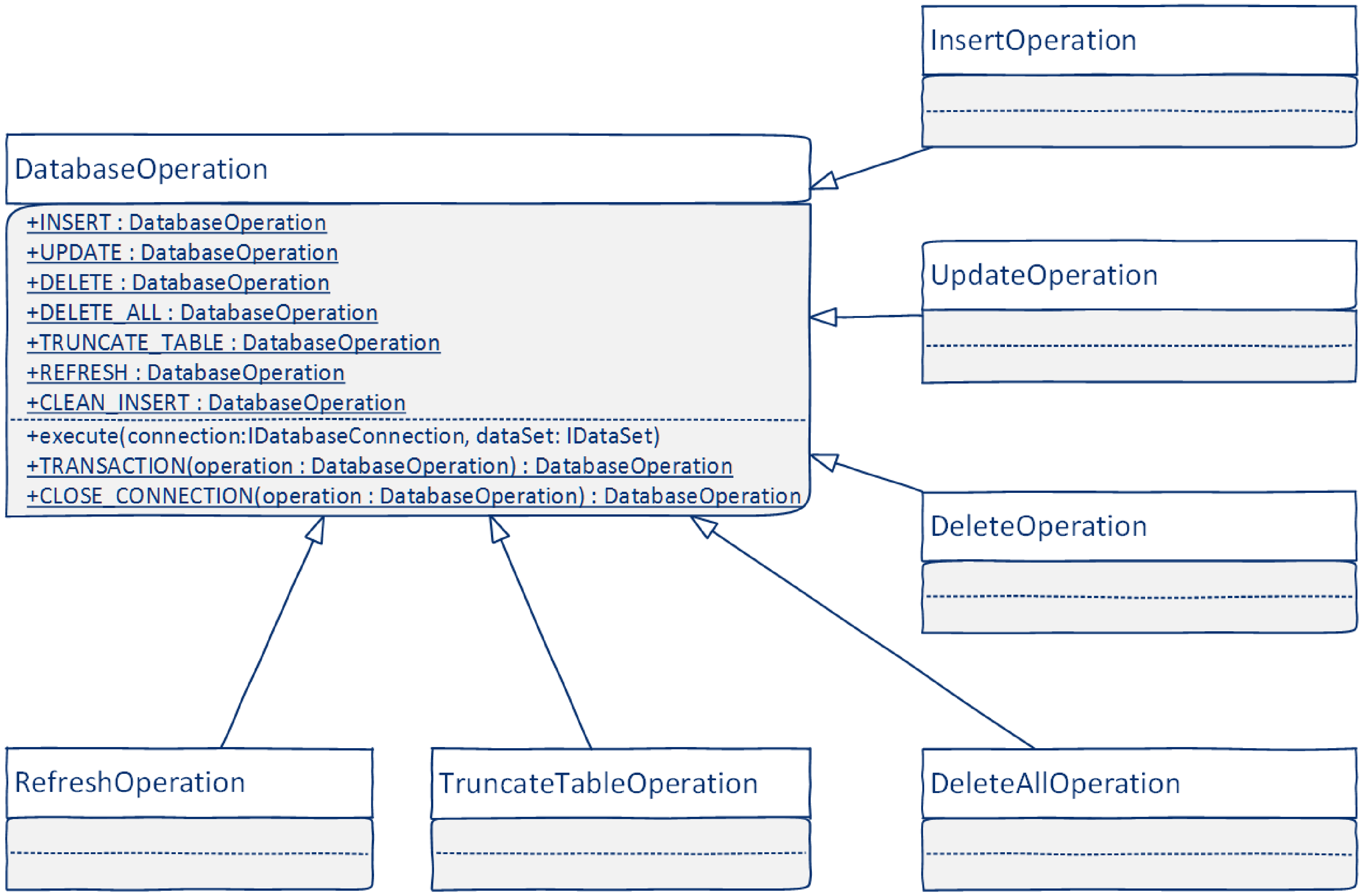
Core Components

- Connection – IDatabaseConnection
- Data Set – IDataSet
- Manipulation - DatabaseOperation

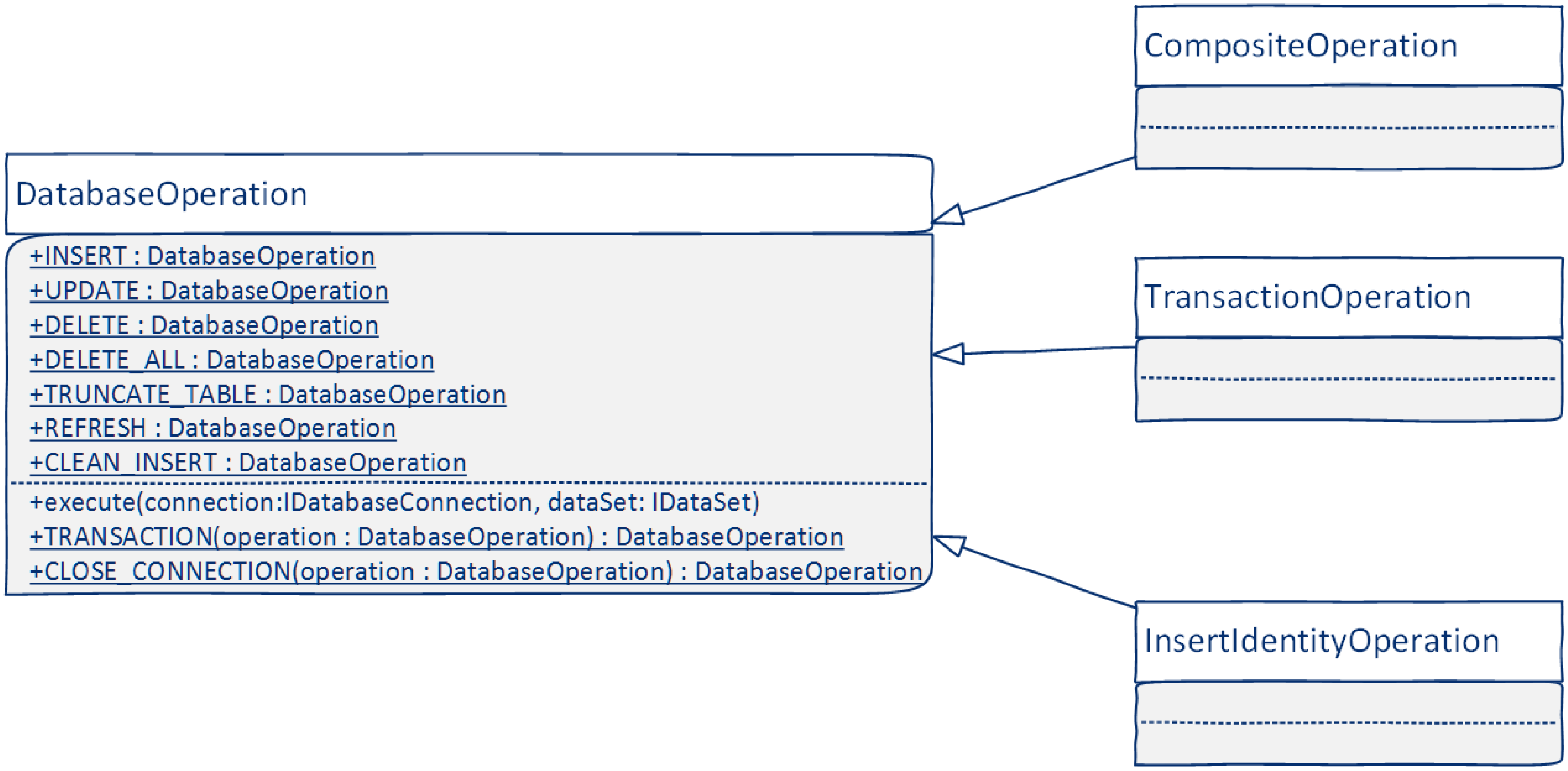


Database Operations

Database Operations


DatabaseOperation
+INSERT : DatabaseOperation
+UPDATE : DatabaseOperation
+DELETE : DatabaseOperation
+DELETE_ALL : DatabaseOperation
+TRUNCATE_TABLE : DatabaseOperation
+REFRESH : DatabaseOperation
+CLEAN_INSERT : DatabaseOperation
+execute(connection:IDatabaseConnection, dataSet: IDataSet)
+TRANSACTION(operation : DatabaseOperation) : DatabaseOperation
+CLOSE_CONNECTION(operation : DatabaseOperation) : DatabaseOperation
InsertOperation
UpdateOperation
DeleteOperation
RefreshOperation
TruncateTableOperation
DeleteAllOperation

Database Operations


DatabaseOperation
+INSERT : DatabaseOperation
+UPDATE : DatabaseOperation
+DELETE : DatabaseOperation
+DELETE_ALL : DatabaseOperation
+TRUNCATE_TABLE : DatabaseOperation
+REFRESH : DatabaseOperation
+CLEAN_INSERT : DatabaseOperation
+execute(connection:IDatabaseConnection, dataSet: IDataSet)
+TRANSACTION(operation : DatabaseOperation) : DatabaseOperation
+CLOSE_CONNECTION(operation : DatabaseOperation) : DatabaseOperation
CompositeOperation
TransactionOperation
InsertIdentityOperation

File-based Data Sets

IDataSet

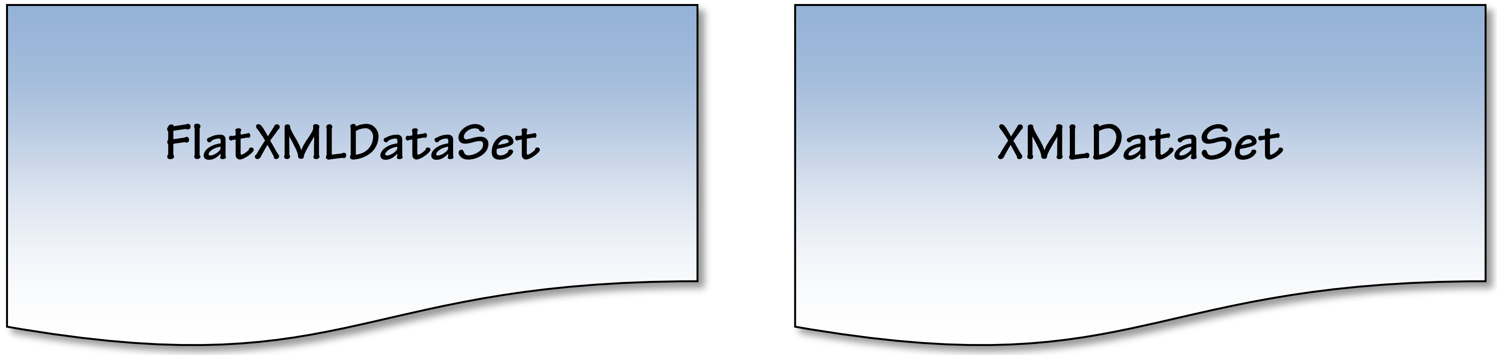
- Exposes in-memory tables of data via ITable implementations

IDataSet

- getTable(tableName:String) : ITable
- getTables() : ITable[]
- getTableNames() : String[]
- iterator : ITableIterator
- getTableMetaData : ITableMetaData

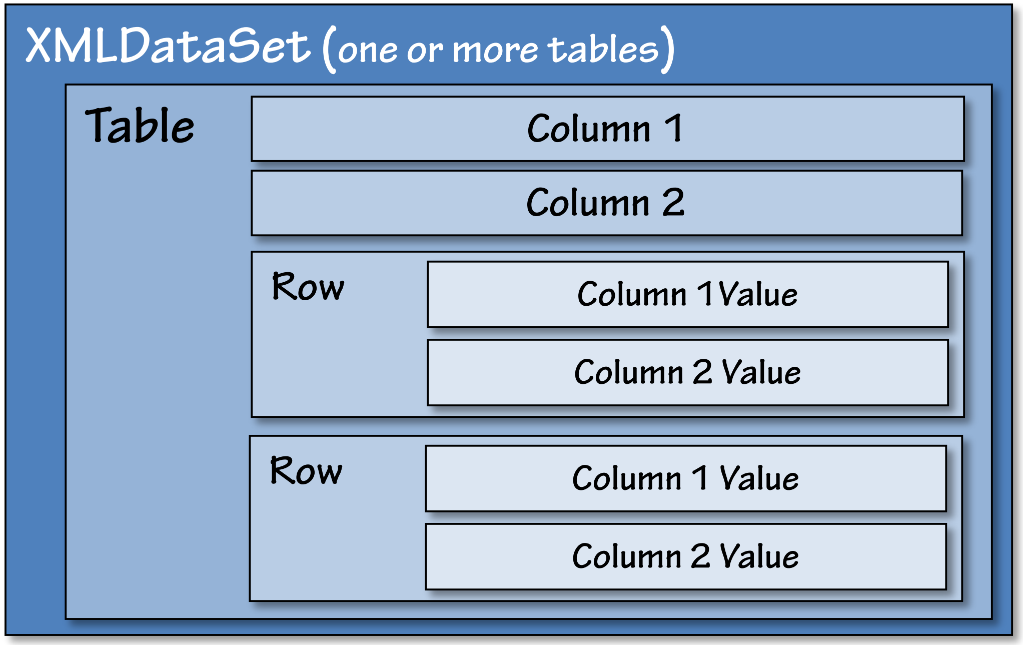
XML Data Sets

- **Manage test data within XML-based files**



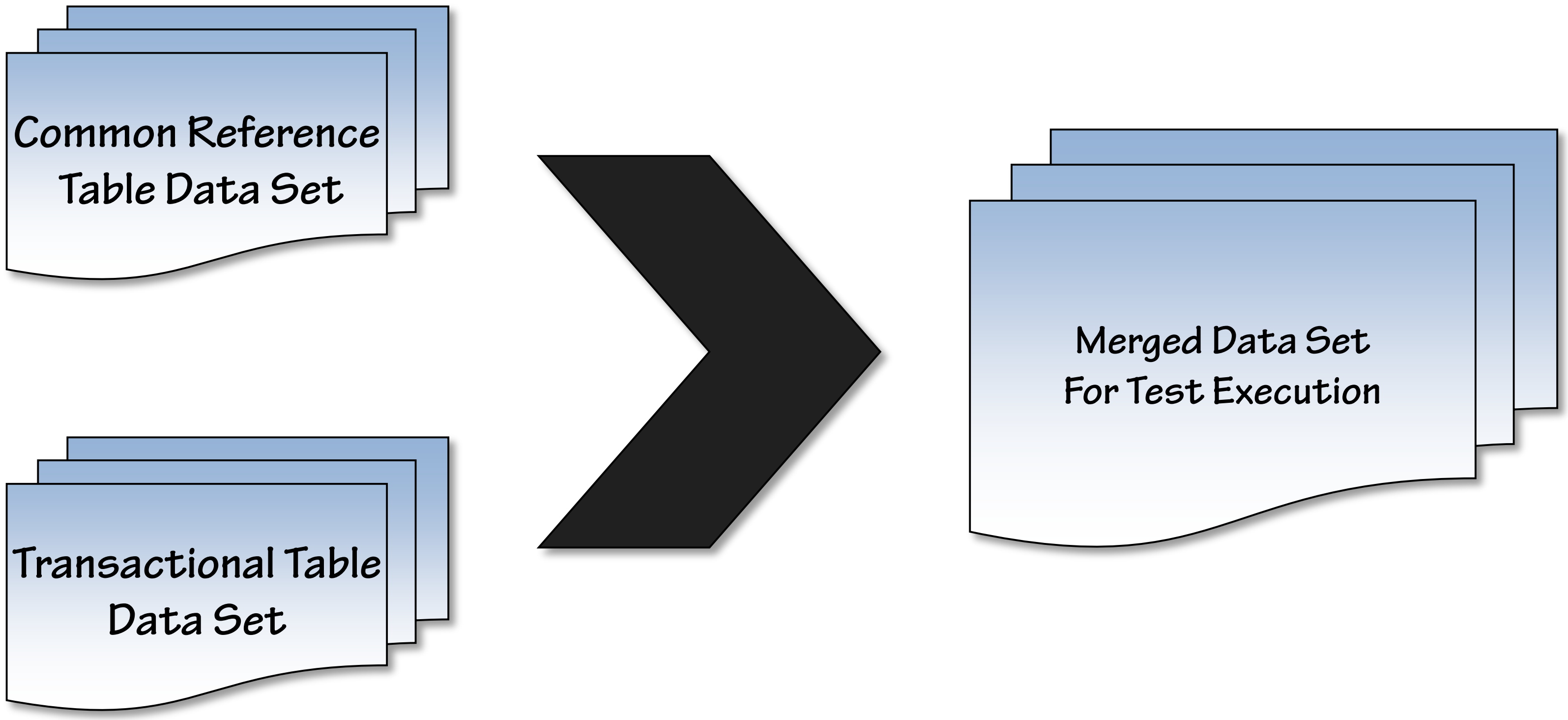
XMLDataSet

- Must conform to a strict DTD structure



XMLDataSet

- Data may become redundant
- Composite Data Set


Common Reference Table Data Set
Transactional Table Data Set
Merged Data Set For Test Execution

FlatXMLDataSet

- **Does not conform to DTD**
- **Be careful when omitting columns**
  - Database insert built on first occurrence of each table

FlatXMLDataSet

```
<dataset>
    …
    <OrderItem  id=“12345”
        quantity=“1”
        sellingPrice=“25.99”
    />
    <OrderItem  id=“12346”
        quantity=“2”
        sellingPrice=“3.99”
        backorderDate=“2014-01-01”
    />
    …
</dataset>
```

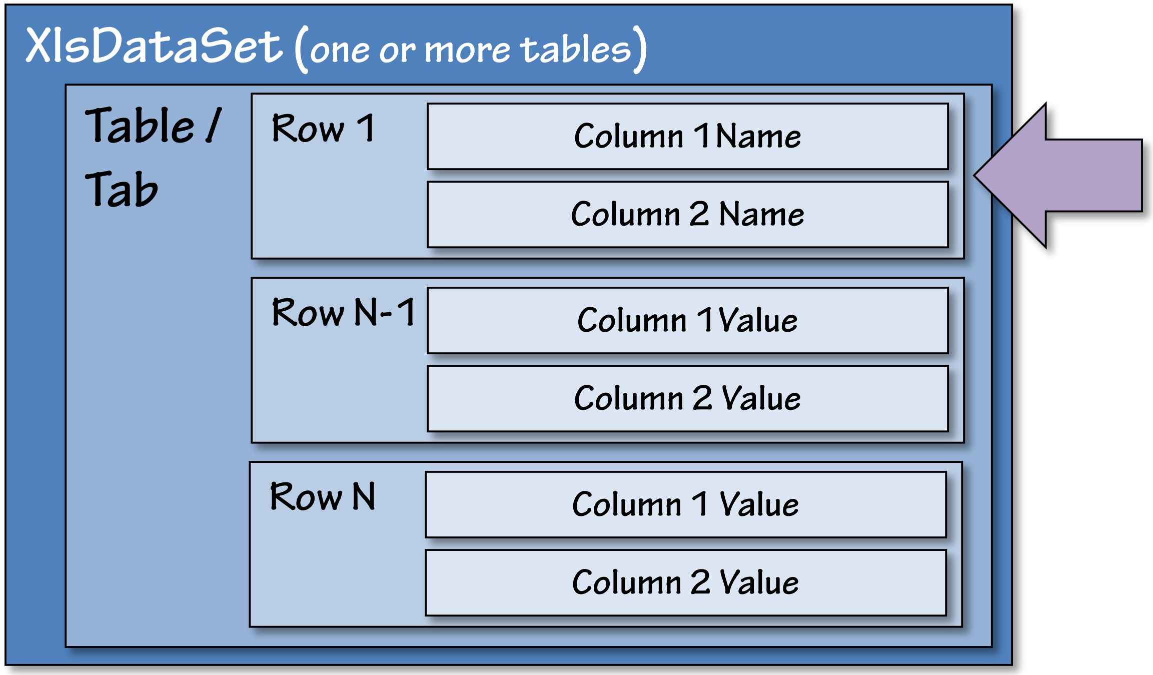
FlatXMLDataSet

- **Does not conform to DTD**

- **Be careful when omitting columns**
  - Database insert built on first occurrence of each table

- **Consider using the FlatXMLDataSetBuilder to create instances**
  - *columnSensing* field set to true will scan all elements of the same name first
  - Because of this scan, there is a slight performance impact on the test execution

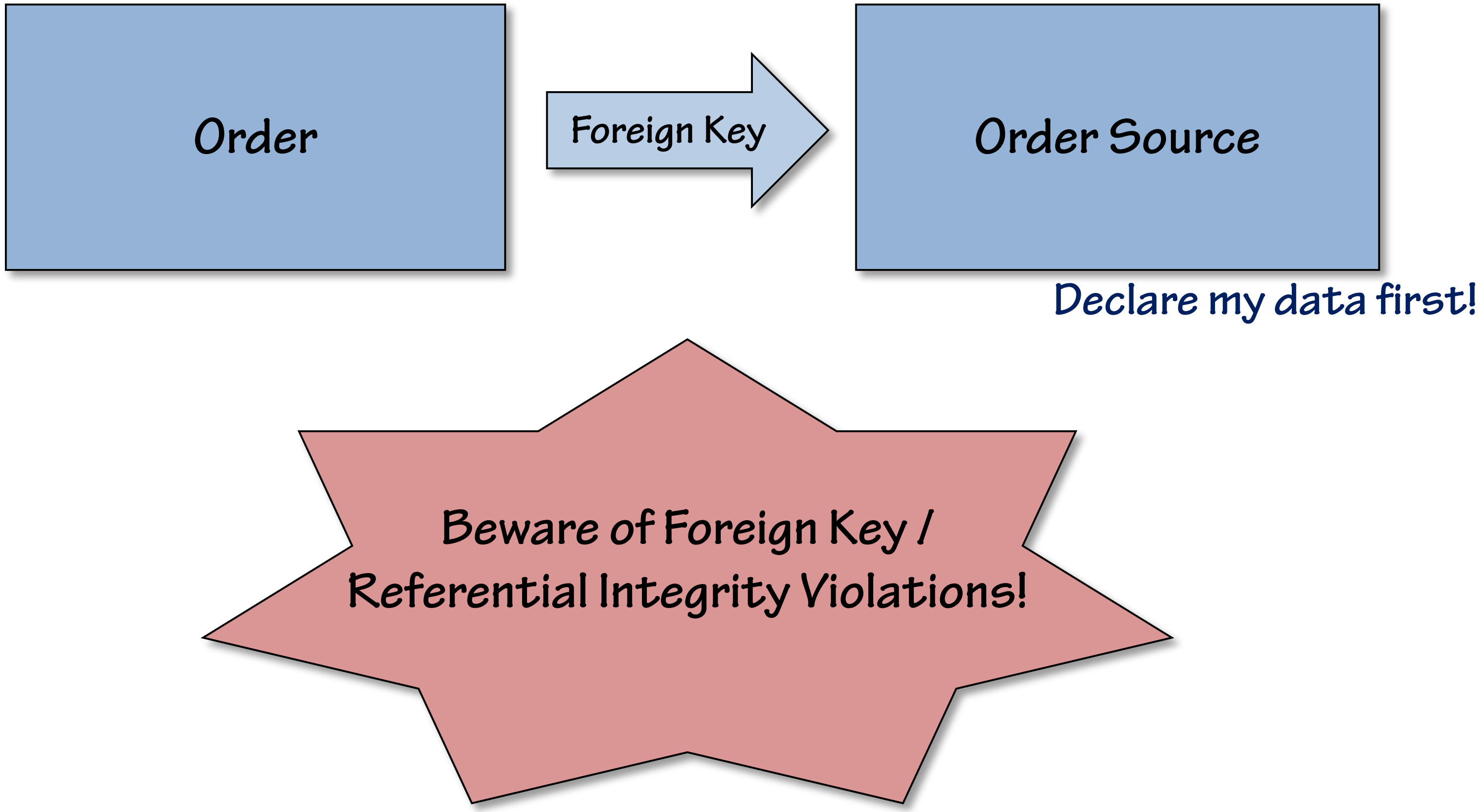
Excel Data Sets



First row used as basis for creating the SQL statements

Table Ordering

- The order the table specified is very important!



Other Data Sets

Composite Data Sets

- **Tables across multiple, varying data sets, are merged together**

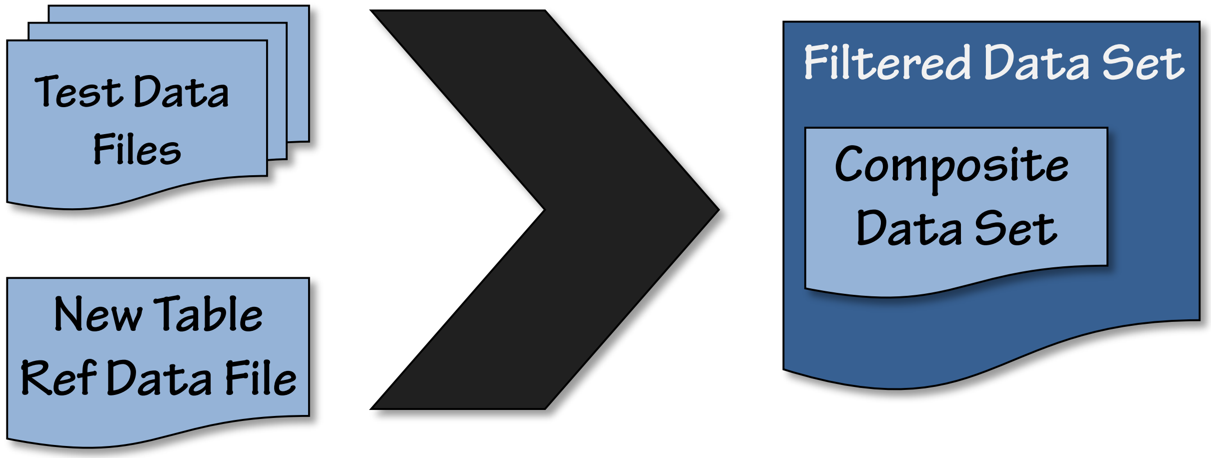
```
…
IDataSet referenceTableDataSet = …
IDataSet transactionTableDataSet = …

IDataSet[] sourceDataSets = new IDataSet[] {
    referenceTableDataSet,
    transactionTableDataSet
};

IDataSet mergedDataSet = new CompositeDataSet(sourceDataSets);
…
```

Filter Data Set

- Allows you to store a lot of data in a single data set and narrow down when loaded
    - Specify tables to include
    - Specify tables to exclude
- Re-sequence tables in the data set



Replacement Data Set

- Specify value replacements broadly in a data set

```
<dataset>
    …
    <OrderItem  id=“12345”
        quantity=“1”
        sellingPrice=“25.99”
        itemMessage=“xx-null-xx”
/>
    <OrderItem  id=“12346”
        quantity=“2”
        sellingPrice=“3.99”
        itemMessage=“Must have 2 items delivered”
    />
    …
</dataset>
```

Replacement Data Set

- Specify value replacements broadly in a data set

```
IDataSet sourceDataSet = … // A FlatXmlDataSet

IDataSet replacementSourceDataSet = new ReplacementDataSet(sourceDataSet);

replacementSourceDataSet.addReplacementSubstring(“xx-null-xx”, null);
```

Replacement Data Set

- **Specify value replacements broadly in a data set**

```
IDataSet sourceDataSet = … // A FlatXmlDataSet

IDataSet replacementSourceDataSet = new ReplacementDataSet(sourceDataSet);

replacementSourceDataSet.setStrictReplacement(true);
replacementSourceDataSet.addReplacementSubstring(“xx-null-xx”, null);
```

Databases As A Source Data Set

DatabaseDataSet

- Allows access to all data in the database

QueryDataSet

- Only allows access to data in the specified tables

Databases As A Source Data Set

DatabaseDataSet

- Allows access to all data in the database
- All data in a table is lazy-loaded on demand

QueryDataSet

- Only allows access to data in the specified tables

Databases As A Source Data Set

DatabaseDataSet

- Allows access to all data in the database
- All data in a table is lazy-loaded on demand

QueryDataSet

- Only allows access to data in the specified tables
- Limit tables, rows (via criteria), and columns fetched

Databases As A Source Data Set

DatabaseDataSet

- Allows access to all data in the database
- All data in a table is lazy-loaded on demand

QueryDataSet

- Only allows access to data in the specified tables
- Limit tables, rows (via criteria), and columns fetched
- Create logical tables based on result of a complex query

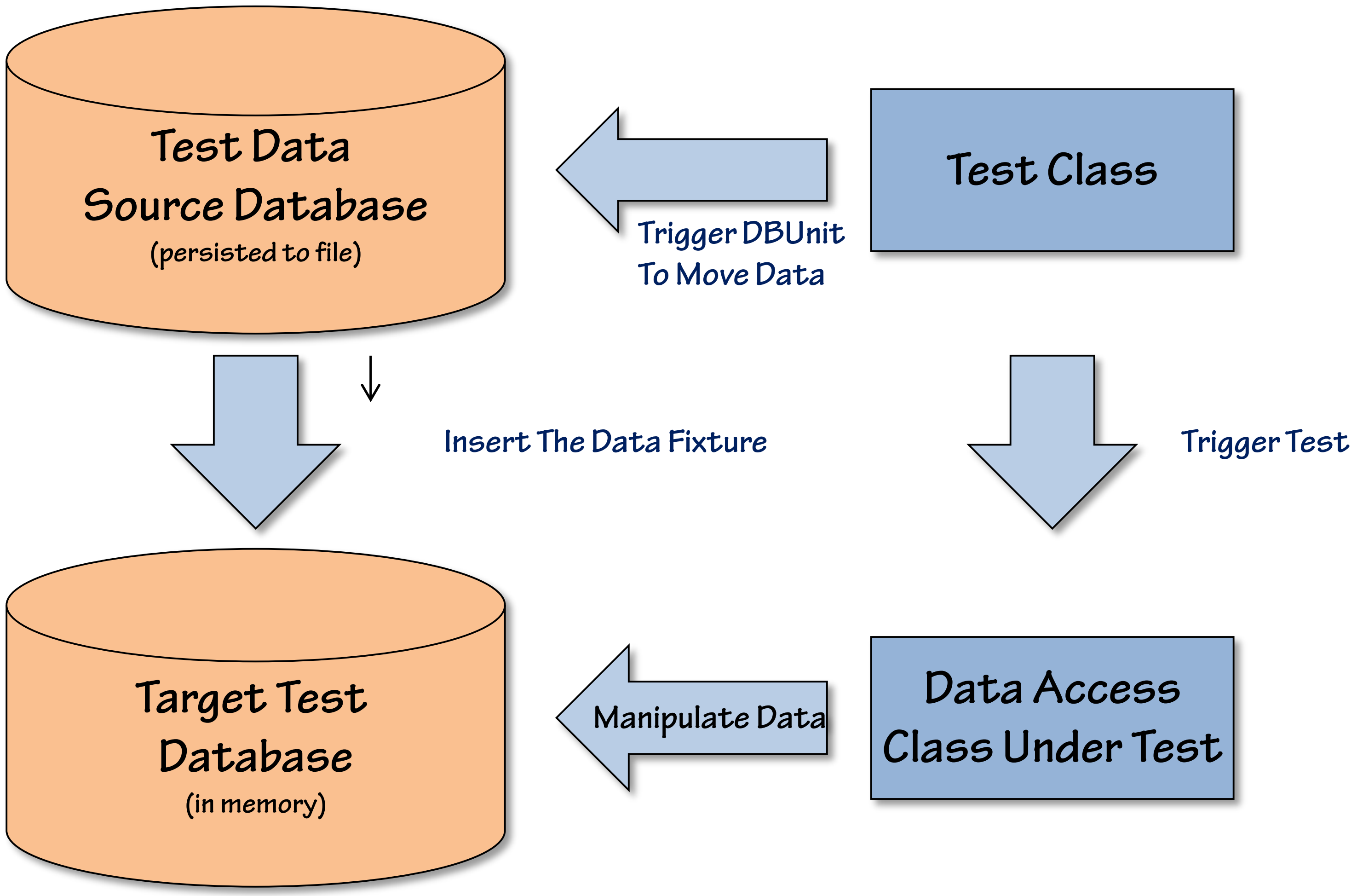
Comparisons

- **The Assertion class can compare data sets and tables**

```
IDataSet expectedResultDataSet = … // Load from one of the file-based data sets
IDataSet actualResultDataSet = new QueryDataSet();

// Add the tables you want to verify in the query data set
Assertion.assertEquals(expectedResultDataSet, actualResultDataSet);
```

Pattern For Using a Source Database



Summary

- **Fixture Management**
- **Data Access Testing**
- **DBUnit**
  - Database Operations
  - Data Sets