Сравнение фреймворка SDUnit для юнит-тестирования БД с другими аналогами

Ниже будет представлено сравнение нескольких фреймворков с SDUnit, созданным в Сбербанк Технологии.

Первым на очереди будет фреймворк под названием Database Rider. Вот основные различия:

* В Database Rider осуществляется правило JUnit для интеграции с DBUnit с помощью аннотаций в Java коде, когда в SDUnit интеграции через аннотации в Java коде нет;
* В Database Rider поддерживаются JSON, YAML, XML, XLS и CSV, а в SDUnit фреймворке поддерживается XML, SQL, XLS (XLSX);
* В Database Rider настройка DataSet выполняется с помощью аннотации @DataSet на уровне класса или метода. В SDUnit настройка юнит-тестов выполняется либо через XML, либо через интерфейс приложения;
* Database Rider поддерживает все базы данных, какие и JDBC. SDUnit предназначен для конкретной работы с Teradata и Oracle;
* Database Rider также работает с Groovy и JavaScript в наборе данных, когда в SDUnit данный функционал не поддерживается;
* SDUnit поддерживает достаточно крупный набор Assert-методов как для проверки на правильность или ложь, так и для проверки, что указанная таблица (или представление) пустая. В свою очередь Database Rider поддерживает лишь Assert-методы для проверки DataSet с выше представленными расширениями;
* SDUnit позвалят пользователю создать различные заглушки, которые в некоторых случаях бывают незаменимыми. Database Rider предоставить выше описанный функционал не может;
* SDUnit содержит атрибут **rollback\_after\_execution**, который указывает, как следует фреймворку выполнять юнит-тест: весь тест в одной транзакции или после каждого statement выполнить явной commit. В Database Rider правило EntityManagerProvider предоставляет транзакции менеджера сущностей, поэтому есть возможность вставлять / удалять сущности в тестах.

Database Rider не удовлетворяет требованиям программистов Сбербанк Технологии по тестированию некоторых ключевых объектов БД, так как не содержит необходимого функционала.

Вторым к сравнению идёт фреймворк под названием JdbcSli. Ниже будут представлено сравнение данного фреймворка и SDUnit:

* В JdbcSli настройка конфигурации осуществляется с помощью файла конфигурации или с помощью вики-странички внутри прибора Define Properties, а в уже представленном SDUnit либо через XML, либо через интерфейс приложения;
* Результаты проведённого тестирования в JdbcSli представляется в том же файле конфигурации, когда в SDUnit они попадают в БД Oracle;
* Фреймворк SDUnit работает с Teradata и Oracle, потому что он писался целенаправленно для этой цели, а JdbcSli являтся фреймворком с открытым исходным кодом. Так как пользователей много, а сколько пользователей, столько вариаций применения, следовательно он должен поддерживать все распространённые БД: Oracle, SQL Server, PostgreSQL, MySQL и др;
* Рассматриваемый фреймворк не может предоставить механизм создания заглушек;
* В JdbcSli механизм работы с транзакциями представлен немного иначе, чем в SDUnit. Для каждого отдельного statement пользователь может отдельно установить, будет оно выполняться в транзакции или нет;
* В JdbcSli набор объектов, которые подлежать тестированию невелик: select запросы и процедуры, когда SDUnit поддерживает тестирование всех возможных объектов БД;
* Как известно, SDUnit фреймворк предоставляет большой список Assert-методов. В JdbcSli такого механизма нет.

При создании JdbcSli акцент ставился не на качество и полноту функционала, а на больший охват пользователей.

Завершающим фреймворком будет NoSQLUnit. Данный фреймворк предназначен для модульного тестирования. Такие тесты должны быть быстрыми, изолированными, повторяемыми, самоутвержденными и своевременными. Тесты будут быстрыми, если они не обращаются к сети или файловой системе, а в случае персистентных систем сетевые и файловые системы являются наиболее используемыми ресурсами. В случае RDBMS (SQL) существует много баз данных Java в памяти, таких как Apache Derby, H2 или HSQLDB. Эти базы данных, как следует из их названия, встроены в вашу программу, а данные хранятся в памяти, поэтому ваши тесты все еще бывают быстрыми. Проблема с системами NoSQL из-за их неоднородности. Некоторые системы работают с использованием подхода Document (например, MongoDb), другие - Column (например, Hbase) или Graph (например, Neo4J). По этой причине поставщик должен предоставить режим оперативной памяти, а универсального решения не существует.

Для однородных систем, таких как СУБД, DBUnit существует для поддержания базы данных в известном состоянии перед каждым выполнением. Но для гетерогенных систем NoSQL подобной среды DBUnit не существует.

NoSQLUnit решает эту проблему, предоставляя расширение JUnit, которое помогает нам управлять жизненным циклом систем NoSQL, а также заботиться о поддержании баз данных в известном состоянии.

NoSQLUnit - это расширение JUnit для упрощения написания модульных и интеграционных тестов систем, в которых используется серверная часть NoSQL, и состоит из двух наборов правил и группы аннотаций.

Первый набор правил предназначен для управления жизненным циклом базы данных; есть два для каждого поддерживаемого бэкэнда.

Первый (если это возможно) - это режим в памяти. Этот режим заботится о запуске и остановке системы базы данных в режиме «в памяти». Этот режим обычно используется во время выполнения модульного тестирования.

Второй - управляемый режим. Этот режим отвечает за запуск NoSQL-сервера, но как удаленный процесс (на локальной машине) и его остановку. Обычно это используется во время выполнения интеграционного тестирования.

Можно добавить их в наборы тестов и / или классы тестов, NoSQLUnit позаботится об однократном запуске базы данных.

Второй набор Правил отвечает за поддержание базы данных в известном состоянии. У каждого поддерживаемого бэкэнда будет свой собственный, и его можно понимать как соединение с определенной базой данных, которая будет использоваться для выполнения необходимых операций для поддержания стабильности системы.

Необходимо обратить внимание, что поскольку базы данных NoSQL неоднородны, каждая система потребует своей собственной реализации.

И, наконец, предоставляются две аннотации, @UsingDataSet и @ShouldMatchDataSet.

@UsingDataSet используется для заполнения базы данных с определенным набором данных. Чтобы заполнить базу данных, используйте аннотацию @UsingDataSet, вы можете определить ее либо в самом тесте, либо на уровне класса. Если есть определение для обоих, аннотация уровня теста имеет приоритет. Эта аннотация имеет два атрибута location и loadStrategy.

С помощью атрибута местоположений вы можете указать местоположение наборов классов.

Второй атрибут обеспечивает стратегии для вставки данных. Реализованные стратегии:

INSERT Вставить определенные наборы данных перед выполнением любого метода тестирования. DELETE\_ALL Удаляет все элементы базы данных перед выполнением любого метода тестирования. CLEAN\_INSERT Это наиболее используемая стратегия. Он удаляет все элементы базы данных, а затем вставляет определенные наборы данных перед выполнением любого метода тестирования.

Данный фреймворк использует JSON файлы в качестве хранилищ данных. Транзакции поддерживаются на уровне Java кода через другие билиотеки. Настройки юнит-теста, как было описано выше, диктуются через аннотации. Механизм управления заглушками отсутствует.

Работает этот фреймворк только с нереляционными БД, так как создавался именно для этой цели. Так как данный фреймворк является расширением Junit, используются Assert - методы последнего. Поддержки работы с процедурами, функциями, триггерами и некоторыми другими объектами БД нет. NoSQLUnit может быть подключен с помощью одной из средств сборки проектов.

Таким образом, представленные фреймворки не подходят для использования, так как в каждом отсутствует часть ключевого функционала, представленного в SDUnit.