САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Автоматизация тестирования в проекте миграции базы данных с SQL Server в Oracle

Дипломная работа студента 545 группы
Баранова Эдуарда Сергеевича

Руководитель	…….. /подпись/	ст. преп. Кириленко Я. А.
Рецензент	…….. /подпись/	к. ф.-м. н. Иванов А. Н.
«Допустить к защите» Заведующий кафедрой,	……. /подпись/	д. ф.-м. н., проф. Терехов А. Н.

Санкт-Петербург
2012

SAINT PETERSBURG STATE UNIVERSITY

Mathematics & Mechanics Faculty

Software Engineering Chair

Automation of QA in the Project of DB Migration from SQL Server into Oracle

by
Eduard Baranov
Graduate paper

Supervisor	……..	Senior Lect. I. A. Kirilenko
Reviewer	……..	Assoc. Prof. A. N. Ivanov
"Approved by" Head of Department	…….	Professor A. N. Terekhov

Saint-Petersburg
2012

Оглавление

Введение

В современном мире одним из важнейших элементов бизнеса является хранение данных и их обработка. Для этой цели и предназначены базы данных и их система управления. Требования к таким системам постоянно меняются и может возникнуть необходимость в замене такой системы. недопустимо перейти на новую чистую БД и ждать, пока будет вновь реализована необходимая функциональность. Решением этой проблемы и занимается миграция БД. Она предназначена для переноса старых данных и функциональности на новую систему.

Миграция базы данных – это достаточно сложный процесс, имеющий свои проблемы и риски. В случае миграции промышленной БД накладывается ещё больше условий и ограничений. В БД хранится огромное количество информации, которая необходима компании для продолжения бизнеса. Поэтому в результате миграции в новой БД должна содержаться абсолютно все данные. Кроме того, БД является частью большой бизнес системы, она должна взаимодействовать с другими элементами, обмениваясь с ними данными. После миграции все остальные приложения должны получать корректные данные от новой БД. Также время на проведение миграции сильно ограничено, поскольку система должна работать практически 24 часа 7 дней в неделю. Миграцию можно проводить только в течение небольшого технического перерыва. В результате необходимо сразу получить работающую систему, содержащую все данные, хранившиеся в исходной БД, чтобы по окончании процесса пользователи системы могли продолжить работу.

Цена ошибки очень велика в таком проекте. Система, частью который является БД, может обрушиться в результате некорректной миграции, или, что ещё хуже, БД может выдать неправильные значения. Чтобы минимизировать вероятность отказа необходимо уделить особое внимание обеспечению контроля качества процесса миграции. Ручное тестирование в таких проектах практически невозможно. Размеры промышленной БД огромны и для ручной проверки потребуется огромное количество человеко-часов, а время на весь процесс миграции ограничено. Кроме того, люди могут совершать ошибки, и поэтому потребуется ещё дополнительные этапы

проверки, что увеличивает стоимость, а главное время, необходимое на проведение тестирования.

Тестирование должно не только проверять корректность результата миграции, но также должно затрагивать и сам процесс миграции БД, а также подготовку к нему. Если на этапе подготовки к миграции были заложены ошибки в процесс, то вероятность получения корректного результата мала. Процесс тестирования должен охватывать все этапы миграции, начиная с самого раннего, и должен быть внедрен в процесс миграции.

В проекте миграции БД необходимо использовать автоматизированное тестирование, отлаженное на этапе подготовки к миграции. Время его выполнения не должно превышать установленных ограничений, а также обеспечивать максимально возможную степень достоверности корректности результата миграции.

Описание проекта

Основная цель проекта миграции – перевод исходной базы данных с MSSQL Server 2005 на Oracle 11gR2. К результату миграции предъявляются следующие требования.

- Должны быть перенесены все данные из исходной базы.
- Должна быть сохранена функциональность исходной базы.

Также есть дополнительное нефункциональное и, на первый взгляд, неформализуемое требование минимизации изменений в схеме данных и в сигнатурах хранимых процедур для упрощения последующей адаптации существующих клиентских приложений к новой базе данных.

В проекте можно выделить следующие особенности: объем базы данных и недостаточная документация. Размер исходной базы достаточно велик: 6 терабайт данных и 2,5 млн. строк кода хранимых процедур. Документация к исходной базе неполна, она устарела и несоответствует реальной БД. Описание функциональности, реализованной в хранимых процедурах, отсутствует, равно как и тесты. Также нужно отметить, что промышленная БД постоянно используется, в ней изменяются данные, в нее вносятся исправления. В проекте необходимо учитывать все возможные

изменения, которые могут произойти в промышленной БД во время разработки инструмента миграции.

БД используется в режиме 24/7, что накладывает ограничения на время, за которое должна проводиться миграция. Существует несколько методологий миграции таких систем, но в проекте было принято решение проводить миграцию целиком за один этап во время технологического перерыва. Поэтому процесс миграции должен завершиться за 24 часа. Такие жесткие временные рамки требуют полностью отлаженного процесса.

Весь проект миграции организован как одновременное продвижение в двух направлениях: аналитика исходной БД и развитие комплекса инструментов для автоматизации миграции. В течение проекта комплекс инструментов миграции непрерывно развивается и совершенствуется. Для этого разрабатывается комплекс средств автоматизации всех запланированных шагов миграции. Миграция разбивается на три этапа: трансформация схемы данных, миграция хранимых процедур и перегрузка данных. Одновременно с основным процессом миграции развивается комплексная система контроля качества.

Отсутствие документации, описывающей исходное функциональное поведение, а также отсутствие тестов накладывает свои ограничения на систему контроля качества, а функциональная сложность системы (2,5 млн. строк хранимого кода) и большой объем данных (6 Тб) значительно затрудняют задачу организации качественного тестирования. Кроме того в структуру и хранимый код БД вносятся изменения, что также затрудняет сопоставление функционального поведения.

Цели тестирования

В рамках проекта необходимо создать комплексную систему тестирования. Главная цель проекта миграции – получить новую БД, которая будет содержать те же данные и функционально соответствовать исходной. Поэтому тестирование сосредоточено на соответствии этим требованиям. Система тестирования должна быть автоматизирована и внедрена в проект, обеспечивая постоянный контроль над развитием инструмента миграции и

всем процессом миграции в целом.

В данной работе рассматривается только функциональное тестирование, и предлагаемая методология основана только на применении функционального тестирования. Безусловно, применение остальных видов тестирования необходимо в проекте, в особенности тестирование производительности, но их методики и реализации остались за рамками данной работы.

Эталоном качества для мигрированной БД по постановке задачи принята оригинальная БД. Поведение исходной БД признается корректным и мигрированная БД должна быть функционально эквивалентна ей с точностью до документированных изменений, в которые входят переименования и изменения в структуре БД, а также согласованные изменения в семантике процедур.

Для функционального тестирования задаются следующие цели:

- верификация полноты миграции данных;
- функциональное соответствие исходной и мигрированной БД.

Также необходим контроль над процессом разработки комплекса инструментов миграции и постоянная проверка его работы. В связи с этим можно выделить следующие объекты тестирования:

- инструмент миграции кода;
- процедура миграции данных;
- код БД для СУБД Oracle.

Функциональное тестирование организуется для следующих задач.

- Обеспечение постоянного контроля соответствия генерируемого кода Oracle PL/SQL заложенным в инструмент миграции проекциям.
- Обеспечение постоянного контроля регресса в ходе разработки инструмента миграции.
- Обеспечение проверки соответствия функциональности исходной (SQL Server) и мигрированной (Oracle) версий БД.
- Обеспечение текущего контроля полноты миграции кода.
- Обеспечение контроля корректности миграции данных.

Обзор

Существует множество статей, в которых описана методология миграции баз данных. В качестве примера можно привести статью [4], в которой автор разбивает весь процесс на несколько фаз и описывает цели каждой их них. Процесс миграции между базами с различными моделями данных рассматриваются в работах [1] и [5]. Методология последовательной миграции БД – бабочка – представлена в статье [10] на примере применения этой методологии для миграции устаревших систем.

Проекты миграции имеют свои особенности и риски. В статье [3] рассматривается проблема управления. Автор фокусируется на трех основных направлениях развития проекта, представляя их в виде треугольника. Одним из главных направлений является оценка качества, для которого описываются специфика целей и рисков в проектах миграции. Более подробно тестирование рассмотрено в статье [7]. Авторы предлагают проводить тестирование в 2-х направлениях: валидация данных и тестирование процесса миграции. Процесс тестирования и минимизация затрат на него на протяжении всего цикла миграции БД рассмотрен в работе [9].

Разностное тестирование впервые предложено в статье [8]. Автор определяет его, как специальный вид случайного тестирования для нахождения различий между разными реализациями одной функциональности. Идея получила дальнейшее развитие. В работе [2] обсуждается регрессионное тестирование, где предлагается использовать юнит тесты для нахождения различий между разными версиями одного кода. О валидации данных написано множество статей. В работе [6] описывается процесс с акцентом на автоматизацию для улучшения качества и безопасности процесса валидации данных.

Опыт тестирования результатов миграции устаревшей системы, переносимой на другую архитектуру, изложен в статье [11]. Идея выборочного регрессионного тестирования, сравнивающая результаты работы старой и новой системы, показала свою результативность.

Таким образом, важность системы контроля качества для проекта миграции не представляет сомнения. Среди успешных методик можно

выделить процесс валидации, используемый для обеспечения качества миграции данных. Для определения функциональной эквивалентности успешно применялось регрессионное тестирование, позволяющее проверить результаты работы после перехода на новую систему.

Методология тестирования

Тестирование выполняется в двух основных направлениях по числу основных требований к проекту: тестирование сгенерированного кода и тестирование процесса миграции данных.

Тестирование мигрированного кода

Главной задачей данного этапа тестирования является контроль соответствия исходной и мигрированной БД. Основу всего процесса функционального тестирования составляет синхронное проигрывание подготовленных трасс в двух БД, исходной и мигрированной. Трасса – это последовательность запросов к БД, которая формализует взаимодействие клиентского приложения с БД. Основным требованием к набору трасс является достаточно полное покрытие функциональности. Тестирование проводится методом «черного ящика», когда проверяются только внешние эффекты при работе тестируемых объектов. В качестве внешних эффектов, возникающих при воспроизведении трасс, рассматриваются коды возврата, выходные параметры, возвращаемые наборы данных, а также изменения внутри БД.

Первые трассы были созданы на основе тестовых сценариев, которые были предоставлены вместе с исходной БД. Тестовый сценарий – это последовательность действий пользователя при работе с клиентским приложением, записанная на естественном языке. Запросы к БД собираются во время их воспроизведения и преобразуются в трассу. Также было собрано несколько трасс с промышленных серверов для сбора более точной информации об используемой функциональности. Это позволило уточнить наболее приоритеные пути расширения функционального тестирования для покрытия наиболее важных частей БД. Кроме того создано несколько синтетических трасс, позволяющих задействовать редко используемую функциональность.

Воспроизведение трасс, собранных с промышленного сервера, предоставляется наиболее полезным, так как будет задействована реально используемая функциональность. Но их сбор сопряжен с дополнительными

трудностями. Наиболее очевидной является проблема обезличивания. В запросы, добавленные в трассы, может попасть конфиденциальная информация, передавать которую в третьи руки категорически запрещено. Для возможности снятия трасс с промышленного сервера требуется предоставить инструмент, который позволит удалить всю попавшую конфиденциальную информацию, либо не собирать её вовсе. Но в то же время такое изменение трасс не должно повлиять на возможность воспроизведения созданной трассы. Поскольку реализация метода, позволяющая совсем не собирать персонифицированную информацию, кажется более сложной, то рассматривалась идея автоматического удаления такой информации на этапе создания трассы.

Полный процесс миграции БД занимает несколько часов. Поскольку развитие системы контроля качества происходит одновременно с разработкой инструмента миграции, то использование только тестирования, основанного на проигрывании трасс, не является достаточно удобным для разработчиков. Поэтому в задачу тестирования также входит обеспечение возможности нахождения ошибок на более ранних стадиях.

Раннее определение недостатков

Для первичного тестирования инструмента миграции используются небольшие модульные тесты, покрывающие основную функциональность. Эти тесты предназначены для раннего обнаружения регресса. Например, среди них содержатся примеры проекций на все конструкции. Запуск тестов производится автоматически после каждого изменения в системе контроля версий, что позволяет быстро определить некорректную трансформацию конструкций. Тестовый набор постоянно пополняется по мере развития инструмента миграции и при поддержке трансформации новых конструкций.

Ежедневно выполняется автоматическая загрузка и компиляция в Oracle процедур, переведенных свежей версией инструмента. Это позволяет следить за числом синтаксически корректно переведенных процедур и выявлять ошибки, возникшие в транслированных процедурах. Такой тип тестирования достаточно неполон с синтаксической точки зрения для конкретной мигрируемой БД, поскольку много критически важных процедур активно используют динамически формируемые запросы. Тем не менее, этот

этап тестирования достаточно полезен, так как позволяет контролировать возможный регресс, возникающий в ходе развития инструмента миграции.

Данный этап позволяет обеспечить постоянный контроль над соответствием генерируемого кода Oracle PL/SQL заложенным в инструмент миграции проекциям и контроль регресса в ходе разработки инструмента миграции.

Проигрывание трасс и сравнение результатов

Проигрывание серии трасс всегда начинается с одного и того же состояния, определяемое сохранённым образом исходной БД. Мигрированная БД – это результат применения разрабатываемого процесса миграции к сохраненному состоянию исходной БД. Таким образом, проигрывание трасс всегда начинается с двух БД, находящихся в эквивалентных состояниях. Для функционального тестирования трассы записываются на исходной БД в однопользовательском режиме на отдельной виртуальной машине. Трассы проигрываются синхронно, также в однопользовательском режиме, что позволяет обеспечивать детерминированность порядка выполнения хранимых процедур и регулярную воспроизводимость результата при проигрывании трасс. Расхождения в результатах могут наблюдаться только в наборах данных, возвращаемых неупорядоченными выборками.

При синхронном выполнении трасс производится контроль следующих внешних эффектов:

- коды возврата и значения выходных параметров процедур;
- возвращаемые наборы данных;
- изменения в БД, произошедшие на каждом шаге.

Получаемые на каждом шаге наборы данных сравниваются как разность множеств A-B и B-A. Если обе разности представляют пустое множество, то результат считается положительным (т.е. транслированный код считается функционально тождественным исходному T-SQL коду). В противном случае, если разностные множества не пустые, производится попытка сравнения множеств A-B и B-A по строкам, а затем по столбцам для локализации ошибки, и результаты сравнения записываются в отчет

(отдельно для каждой трассы). Также в отчете фиксируются все ошибки выполнения переведенного и исходного кода.

Другим этапом контроля эквивалентности исходной и мигрированной БД является контроль изменений внутри БД, произошедших во время проигрывания трасс. Для этого в исходной и мигрированной БД для каждой таблицы добавлены триггера на события INSERT/UPDATE/DELETE. Они записывают в таблицу аудита информацию обо всех изменениях, произошедших во время проигрывания трассы.

Оценка тестового покрытия

Поскольку целью является эквивалентное поведение исходной и мигрированной БД, то одной из важнейших характеристик набора трасс является оценка тестового покрытия. Оценка вычисляется на исходной БД как отношение различных выполненных операторов в ходе воспроизведения трасс к общему числу операторов в БД (в том числе и мертвый код).

Предполагается, что предоставленные тестовые сценарии, по которым записываются трассы, достаточно полно покрывают функционал системы. В требованиях к проекту утверждено, что тесты должны покрывать не менее 20% операторов.

Для оценки качества (полноты) тестового покрытия на тестовых серверах в начало каждого линейного участка процедуры, те есть участка кода без разветвлений и слияний, добавляется команда вставки в специальную таблицу о прохождении данной точки. Это позволяет с приемлемой точностью отследить порядок выполнения операторов. Во время выполнения трасс эта таблица заполняется, последовательно записывая информацию о прохождении точек. В результате после проигрывания трасс из этой таблицы можно достать путь, по которому выполнялась трасса. Зная все точки и их местоположение, по этим данным можно отследить ход выполнения трассы и оценить покрытие кода тестами.

Тестирование процесса миграции данных

Вторым основным аспектом тестирования является контроль процесса миграции данных, а также полнота и корректность результата миграции. Для

этого необходимо удостовериться, что все данные мигрировали и мигрировали корректно. Поэтому в основе механизма контроля качества процесса перегрузки данных лежат анализ кодов возврата и логов всех задействованных в процессе перегрузки утилит, контроль целостности средствами БД и дополнительный контроль результата перегрузки – валидация.

Валидация позволяет удостовериться, что все данные успешно перегружены в новую БД, претерпев необходимые документированные изменения. Актуальность валидации при миграции БД такого объема сложно недооценить. Во-первых, в схеме более 2 тысяч таблиц и почти 10000 колонок, а некоторые таблицы содержат десятки миллионов записей. Во-вторых, внешние ключи, как средства контроля целостности, в исходной БД практически не используются.

Процесс валидации должен затрагивать не только содержимое объектов, но и всю схему данных, верифицируя наличие и состояние объектов. Полная валидация для установления идентичности двух баз таких размеров требует затраты больших ресурсов, в особенности времени. Поэтому для процесса необходим подход, требующий значительно меньшего количества ресурсов и в то же время обладающий высокой степенью достоверности результата.

Для первичного тестирования использовалась валидация по числу строк. Критерием успешности для данного метода является совпадение числа строк в таблицах исходной и мигрированной БД. Преимуществами этого метода является высокое быстродействие, особенно на таблицах имеющих первичные или уникальные ключи, когда расчет количества строк проходит по индексу, и не требуется полное сканирование таблиц. Даже такой способ помог выявить неудачно мигрировавшие объекты. Тем не менее достоверность результатов у этого метода достаточно низкая, так как он не позволяет верифицировать значения объектов, что является большим недостатком, особенно для миграции, сопровождающейся трансформацией типов данных.

Для повышения достоверности результата валидации используется подход, в основе которого лежит сравнение хэш-ключей. В обеих БД для каждой колонки в каждой таблице считаются хэш-ключи, а результаты подсчета сохраняются в отдельной таблице. Процедура подсчета хэш-

функции должна быть выбрана с учетом того, что значения хэш-ключей не должны зависеть от порядка обхода строк в таблице, поскольку в общем случае порядок записей в выборке не определён, а упорядочивание может крайне негативно сказаться на скорости подсчета. Далее значения в этих таблицах сравниваются и, в случае расхождения, идентифицируется таблицы и колонки, которые мигрировали неправильно.

Реализация

Общая архитектура

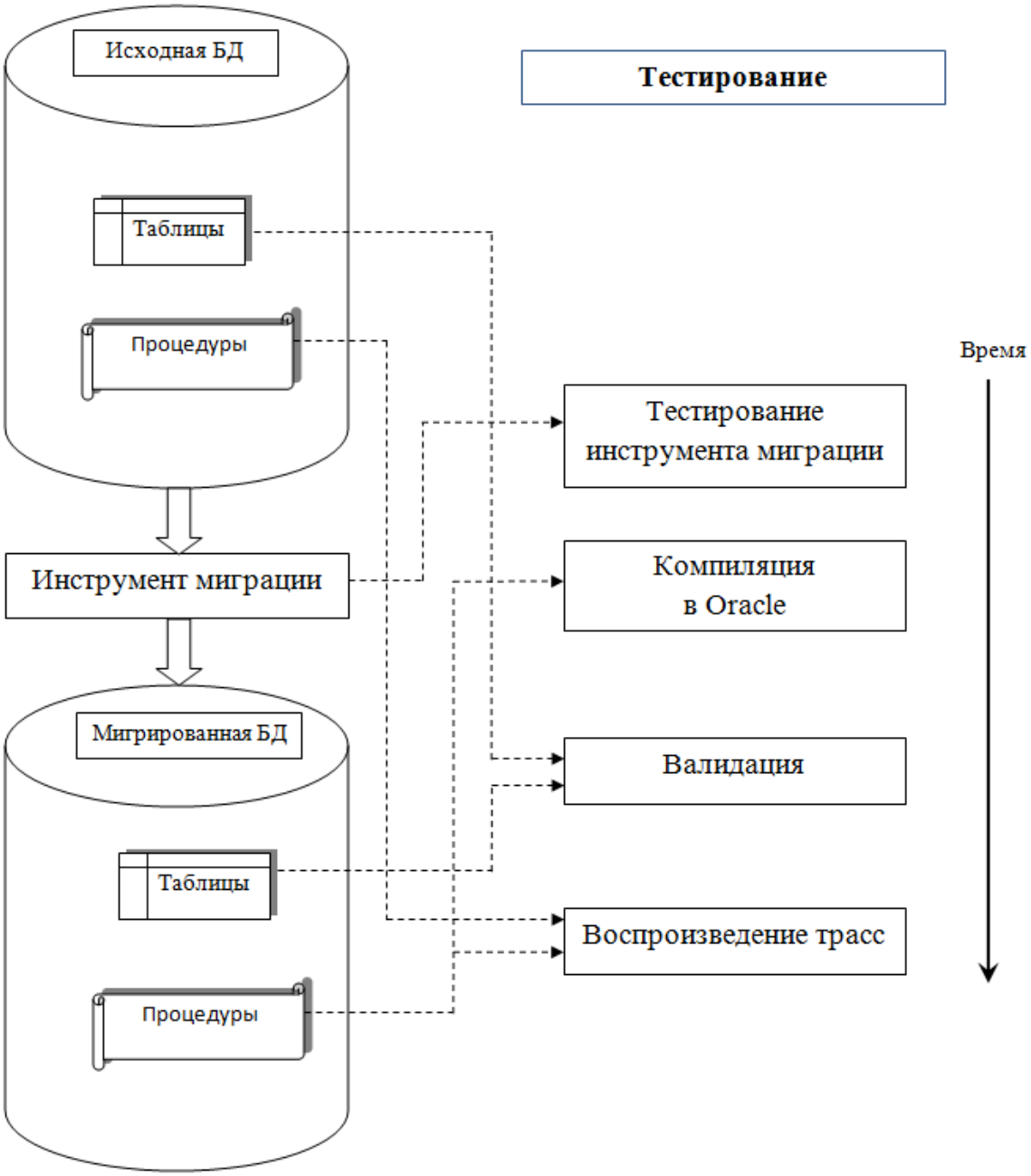

Рис. 1 Последовательность этапов тестирования.

На основе описанной методологии можно выделить следующие основные этапы тестирования:

- тестирование инструмента миграции;
- компиляция процедур в Oracle;
- валидация;
- сравнение внешних эффектов при воспроизведении трасс.

Для тестирования инструмента миграции внутри него создана дополнительные сборки, которые содержит набор модульных тестов. Их запуск производится инструментом NUnit. Компиляция процедур в Oracle проводится во время их загрузки в мигрированную БД. Процесс валидации запускается сразу после перегрузки данных, подробнее последовательность действий, выполняемая внутри процесса, описана ниже. Последняя часть тестирования разбивается на два основных этапа: подготовка трасс для воспроизведения и, собственно, воспроизведение, во время которого сраниваются поведения двух БД.

Создание трассы

Первым этапом тестирования, основанного на сравнении результатов работы на исходной и мигрированной БД, является подготовка трасс. Основные модули, используемые для создания, и последовательность их применения изображено на рисунке 2.

Основным источником трасс были тестовые сценарии, предоставленные вместе с БД. Тестовые сценарии написаны на естественном языке, поэтому полностью автоматизировать процесс создания трасс из них в рамках проекта не представляется возможным. Методика записи трасс из сценариев основана на использовании SQL Server Profiler, встроенного в MS SQL Server 2005. Перед записью трассы создаётся трассировка в SQL Server Profiler и выбираются данные или события, за которыми будет вестись наблюдение, и информация, которая будет собираться о них. Для создания трассы производится наблюдение за следующими событиями: удаленные вызовы процедур, работа с курсорами, выполнение инструкций T-SQL, выполнение пакета запросов и.т.д. Во время работы пользователя с БД через клиентское приложение собираются данные о каждом запросе и после этого сохраняются в файл трассы. Для автоматизации работы с SQL Server Profiler была написана утилита, которая позволяет задавать только необходимый

набор опций для трассировки, такие как параметры подключения к БД и имя файла трассы, а все остальные, в том числе набор столбцов для трассировки, добавляет автоматически.

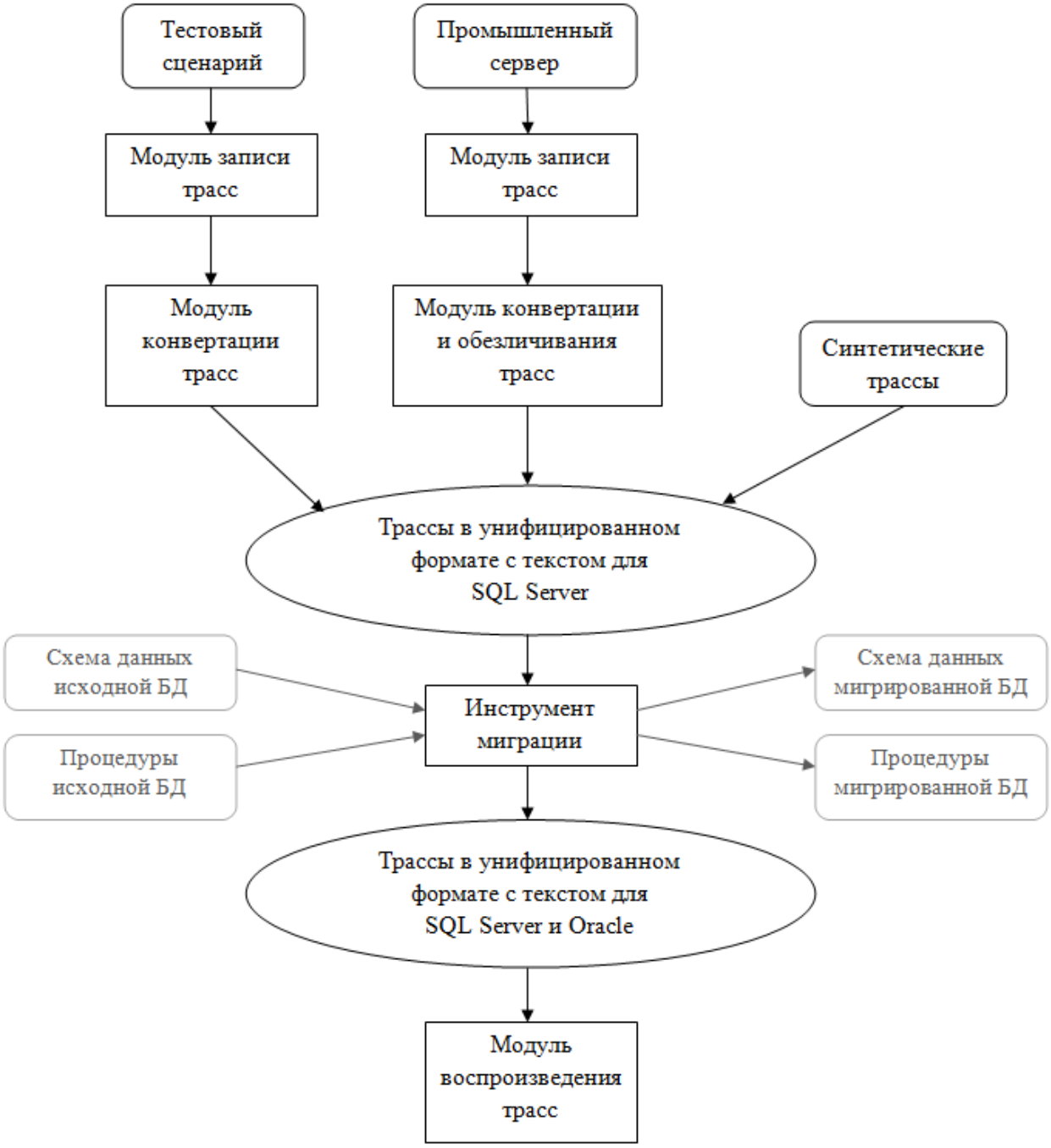

Рис. 2 Модули инструмента для создания и воспроизведения трасс.

После записи трассы из формата SQL Server Profiler конвертируются в унифицированное представление на основе XML. Унифицированное представление отражает структуру оригинальной трассы: состоит из пакетов запросов, разделённых на запросы. Для каждого запроса сохраняется его тип и исходный текст, который выполнялся на SQL Server. Встроенные средства SQL Server Profiler позволяют прочитать необходимую информацию из файла, и по ней создается XML файл. Синтетические трассы сразу создавались в унифицированном формате. По окончании этого этапа трасса сохраняется, и все последующие запуски тестирования начинаются со следующего этапа.

Следующим этапом является конвертация полученной трассы для возможности работы с Oracle. Преобразование проводится автоматически во время трансляции хранимых процедур по тем же правилам, что и обычные процедуры. Это позволяет учесть все переименования и изменения в сигнатурах процедур. Для этого перед трансляцией с помощью скрипта из кодов запросов из трассы создаются процедуры специального вида на T-SQL и подаются на вход к инструменту трансляции вместе с хранимыми процедурами исходной БД. После трансляции сгенеренный код на PL-SQL конвертируется в аналогичное представление на основе XML и добавляется к исходной трассе. Трасса после этапа зависит от текущего состояния инструмента миграции, который постоянно совершенствуется, и для использования самой корректной на текущий момент версии трасс запросы для Oracle должны пересоздаваться перед каждым запуском тестирования.

В результате в каждой трассе для каждого этапа трассы хранится ее тип, который зависит от исходного и нового запроса, текст для SQL Server и сгенерированный текст, который будет выполняться в Oracle. Способ хранения запросов зависит от их типа, что сделано для более удобного воспроизведения подготовленной трассы.

Запросы подразделяются на следующие типы:

- Login;
- Logout;
- Batch;
- Block;
- Procedure.

Первые два не возвращают никаких результатов, поэтому в дальнейшей работе инструмента они не участвуют. Login может содержать несколько инструкций set, которые инструмент выполняет. Также дополнительно во время выполнения этих запросов производится подсчет числа текущих соединений с БД. Принадлежность запросов к этим типам определяется на этапе конвертации из формата SQL Server Profiler в унифицированный формат.

Подразделение запросов на следующие типы происходит во время добавления созданного запроса для Oracle. Различие между Batch и Block в том, что во втором в запросе для Oracle находится более одной инструкции, при этом среди них есть инструкция открытия курсора. Для таких запросов требуется добавить output параметр, имеющий тип RefCursor, чтобы была возможность получить его значение после выполнения запроса. Кроме того в таких случаях требуется модифицировать инструкцию открытия курсора в запросе, добавляя “:” перед именем курсора для связывания его с созданным параметром вызова запроса.

Тип процедура предназначен для вызова хранимых процедур и функций. Для них в XML хранится элемент, содержащий имя хранимой процедуры или функции, и список параметров, с которыми они должны вызываться. Для корректного вызова в Oracle этот список должен содержать все параметры, которые являются out или inout. Поэтому во время создания трассы в унифицированном формате необходимо указать все такие параметры, и, если они не вызываются с каким-нибудь значением, то необходимо указать их значение по умолчанию. Также для возвращаемых параметров необходимо указывать размер для типов Char и VarChar2, а для типа Decimal в MS SQL Server надо указывать точность и масштаб.

Набор созданных трасс в унифицированном формате передается модулю воспроизведения трасс. Запросы для MS SQL Server и Oracle воспроизводятся синхронно. Для каждого элемента трассы создается две команды, которые исполняются в БД. После выполнения собираются результаты их выполнения. От типа запроса зависит, какие результаты будут рассматриваться, а также для разных типов запросов могут добавляться различные дополнительные параметры в команды. После выполнения запроса собираются возвращаемые наборы данных (а также их схема), коды возврата и значения выходных параметров для процедур, а также изменения

внутри БД. Для записи изменений внутри БД перед воспроизведением трасс создаются таблицы аудита и триггера для всех остальных таблиц на события INSERT/UPDATE/DELETE, которые будут записывать факт операции с данными. Сохраняется информация о таблице, в которой произошло изменение, событие, число затронутых строк. Перед каждым запросом из трассы эти таблицы очищаются, а по окончании его выполнения данные из выгружаются для последующего сравнения.

Обезличивание промышленных трасс

Обезличивание промышленных трасс проводится на этапе преобразования трассы из формата SQL Server Profiler в унифицированный формат. При реализации используется предположение, что все персонифицированные данные в запросах содержатся только в строковых литералах. Во время сбора трасс с промышленного сервера пришлось столкнуться с дополнительной проблемой: невозможно снять трассы в однопользовательском режиме. Множество пользователей взимодействуют с БД одновременно и в результате собираются перемешанные запросы от разных пользователей. Это означает, что информация о пользователе, из под логина которого был запущен запрос, является важной для воспроизведения и должна сохраняться в трассе.

В инструмент добавлена возможность автоматического изменения персонифицированной информации. В тексте запроса ищутся все литералы, и информация внутри них заменяется символом ‘x’. Вся информация о текущем пользователе системы и имя БД обезличивается с помощью алгоритма MD5 hash.

Безусловно данный алгоритм нарушает структуру трассы, и в результате воспроизведение таких трасс невозможно. Тем не менее, даже такая реализация достаточно полезна. Она позволила собрать несколько трасс с промышленного сервера, а на их основе были созданы синтетические трассы, покрывающие схожую функциональность, что позволило добавить в тестирование наиболее актуальные сценарии использования БД и улучшить покрытие кода трассами. Доработка алгоритма обезличивания и запуск трасс,

собранных с промышленного сервера, остается для дальнейшего развития системы контроля качества.

Сравнение внешних эффектов

Сравнение внешних эффектов проводится после каждого шага трассы, то есть после выполнения каждой пары запросов. Сравнение проводится для возвращаемых значений, выходных параметров, возвращаемых наборов данных и изменений внутри БД. Для возвращаемых значений и выходных параметров сравниваются их наличие (совпадение их числа и их имен с учетом переименований в ходе миграции для выходных параметров), типы и значения.

На этапе сравнения возвращаемых наборов возникает проблема: они могут содержать столбцы, значения в которых не совпадают из-за того, что они зависят от текущего состояния окружения. Например в таких столбцах могут содержаться время и дата, или это может быть столбец идентификаторов. Если в возвращаемый набор данных записывается текущее время, то добиться идентичности даже при синхронном проигрывании на двух БД практически невозможно. Чтобы избежать возможных расхождений, добавлена возможность игнорировать столбцы такого типа. Для этого кроме самих возвращаемых наборов данных сохраняются также метаданные для столбцов, что позволяет легко определять колонки, которые не должны участвовать в сравнении. Если указано в параметрах запуска инструмента, что необходимо игнорировать эти колонки, то они удаляются из набора данных перед сравнением.

Возвращаемые наборы данных рассматриваются как множество строк со значениями, среди них проводится поиск строк, которые присутствуют только в одном наборе. Для этого вычисляются разности множеств A-B и B-A. Метод вычисления разности множеств (вместо построчного сравнения) позволяет избавиться от проблемы несовпадения порядка записей в наборе, которые могут возникнуть, например, при неупорядоченной выборке. Отдельного внимания заслуживает сравнение строк в наборах. Строки сравниваются поэлементно, то есть строки равны, если значения во всех соответствующих столбцах эквивалентны. В MS SQL Server пустая строка и null – два различных значения, тогда как в Oracle это не всегда верно.

Поэтому создана опция в инструменте, при включении которой пустые строки преобразуются в null перед сравнением. Если получившиеся разности не пусты, то все различия записываются в журнал.

Для сравнения изменений внутри БД данные из таблиц аудита сохраняются в формате таблицы. Это позволяет использовать тот же механизм поиска расхождений, что и для возвращаемых наборов данных. Все выявленные несоответствия сохраняются в журнал.

Требования к инструменту

Для корректной работы инструмента для записи и воспроизведения трасс есть требование к системе: на машине должен быть установлен MS SQL Server (не Express версия). Он необходим, так как используются библиотеки для подключения к БД и для чтения файла трассы из формата SQL Server Profiler. Кроме того, эти библиотеки имеют несколько зависимостей от других, входящих в MS SQL Server, причем они не определяются на этапе компиляции инструмента, но при отсутствии необходимых библиотек инструмент падает во время работы. К сожалению, часть из зависимых библиотек скомпилирована только под .Net 2.0, а для разработки инструмента используется .Net 4.0. Было найдено следующее решение данной проблемы: в конфигурационный файл в элемент startup, который указывает общие сведения о среде CLR, добавляется атрибут useLegacyV2RuntimeActivationPolicy со значением true и с дочерним элементом supportedRuntime, версия которого устанавливается в 4.0. Эта возможность была добавлена в .Net с версии 4.0 и позволяет запускать компоненты, разработанные в предыдущей версии платформы. Каждый компонент при этом запускается в своей версии платформы .Net параллельно в одном процессе.

Оценка тестового покрытия

Для оценки тестового покрытия внутри БД создается таблица, в которую заносятся данные обо всех линейных участках кода, а именно: идентификатор участка, его местоположение, название процедуры, в которой он находится, а также число операторов в нем. Одновременно в процедуры

добавляются инструкции перед каждым линейным участком кода, которые будут записывать информацию о прохождении данного участка в дополнительную таблицу журналирования трасс. Эти инструкции будем называть контрольными точками, а их местоположение можно определить по таблице, содержащей информацию о линейных участках кода.

При выполнении команды записи в таблицу о прохождении линейного участка может измениться контекст исполнения, в частности системные переменные. Анализ исходной БД показал, что изменение переменной @@IDENTITY может привести к некорректному результату выполнения хранимой процедуры. Для сохранения значения переменной @@IDENTITY создана специальная хранимая процедура, которая вызывается при записи информации о прохождении контрольной точки. Вначале сохраняется значение системной переменной, а только после этого выполняется команда вставки идентификатора контрольной точки в таблицу журналирования трасс. Для восстановления переменной создается динамический запрос. Внутри него создается временная таблица со столбцом идентификаторов, причем значение первой строки должно совпадать с сохраненным значением @@IDENTITY. После этого делается вставка во временную таблицу (в этот момент значение системной переменной восстанавливается) и таблица удаляется. Значения других перемкенных не восстанавливается, поскольку они не влияют на результат выполнения процедуры.

Для загрузки в БД измененных процедур используется программа sqlcmd. Хранимые процедуры могут содержать идентификаторы, которые совпадают с управляющей командой sqlcmd. Для предотвращения таких ситуаций перед загрузкой процедуры проверяются на наличие таких идентификаторов и, в случае совпадения, они изменяются (добавляются [] вокруг идентификатора). В исходной базе были найдены идентификаторы “ed”, которые совпадают с управляющей командой sqlcmd. Такие идентификаторы ищутся с помощью регулярных выражений, и замены проводятся автоматически при запуске специально написанного скрипта.

После выполнения инструментированных трасс в БД остается заполненная таблица журналирования. Для подсчета результатов из нее выбираются все различные пройденные контрольные точки и суммируется длина линейных участков, следующих за ними.

Процесс валидации

Первоначально был реализован метод, основанный на сравнении числа строк в таблицах. Был написан скрипт, который подсчитывает число строк в каждой таблице, а после окончания подсчетов значения для соответствующих таблиц сравниваются.

В основе второй реализации валидации лежит сравнение хэш-ключей для столбцов таблиц. Для подсчета хэш-ключей используется коммутативная операция XOR, встроенная SQL. Коммутативность операции гарантирует независимость полученного хэш-ключа от порядка выборки строк из таблицы. Для работы с char и varchar перед использованием операции XOR данные конвертируются с помощью алгоритма MD5, который также встроен в SQL Server. Слишком большие decimal и numeric предварительно конвертируются в varchar, поэтому для них также используется алгоритм MD5. Дополнительно вычисляется хэш-ключ для всей таблицы.

В БД создается дополнительная таблица валидации, в которой сохраняются все необходимые данные. В ней 5 столбцов: имя БД, имя таблицы, имя колонки, тип хэш-функции, значение хэш-ключа. Тип хэш-функции указывает на тип (размер хеш-ключа) и использовался ли алгоритм MD5 перед вычислением ключа. Инструмент валидации создает скрипт на T-SQL, который постепенно вычисляет хэш-ключи для всех колонок и для таблицы целиком, а по окончании добавляет их в таблицу валидации. Вычисление производится внутри Select, а результирующие значения накапливаются в локальных переменных. Имена таблиц и колонок преобразуются перед записью в таблицу с помощью модуля трансформации имен из основного инструмента миграции. Это позволяет на этапе сравнения результатов легко находить соответствующие таблицы и столбцы. По завершению работы скрипта таблица выгружается из БД.

Аналогичная таблица создается и для Oracle, после чего в неё загружаются значения подсчитанные в MS SQL Server. Хэш-функция для Oracle реализована на Java. За один вызов она вычисляет значение хэш-ключей для всех колонок в одной таблице. После этого все вычисленные ключи сохраняются в таблице.

Последним этапом является сравнение полученных значений. Для него написан скрипт, который сначала проверяет совпадение таблиц и колонок,

потом совпадение числа строк и в конце совпадение хэш-ключей. В реализации используются достаточно простые операции Select. Все выявленные несоответствия выдаются в качестве отрицательного результата.

Для валидации создан инструмент, который автоматически генерирует набор необходимых скриптов для подсчета хэш-ключей для всех таблиц и столбцов в обеих БД.

Взаимодействие всех элементов системы

Для согласованной работы всех элементов системы были написаны скрипты, которые управляют запуском всех компонент. Для постоянной работы тестирования используется система непрерывной интеграции TeamCity. Она настроена для ежедневного запуска всей системы миграции и тестирования в необходимом порядке (за исключением модульных тестов, которые выполняются после каждого изменения в системе контроля версий). Это позволяет ежедневно получать новые результаты миграции и тестирования и контролировать процесс разработки.

В большинство компонент добавлено наблюдение за временем ее выполнения. Кроме того, TeamCity предоставляет функциональность оценивания времени, необходимого для каждого шага миграции, основанное на сопоставлении времени, потребовавшегося в предыдущие запуски. Это позволяет ввести дополнительный контроль процесса миграции, проверяя, что он укладывается в отведенные временные рамки. Измерение времени выполнения запросов во время проигрывания трасс позволяет приблизительно оценить производительность мигрированной БД. Поскольку трассами старались покрывать наиболее важный функционал, то временам выполнения трассы можно оценивать скорость работы некоторых типичных сценариев работы с БД. Результаты этих оценок можно будет учитывать на дальнейшем этапе улучшения производительности.

Результаты

Предложенная методология была реализована и внедрена в проект, где показала свою работоспособность. Работа проводилась командой разработчиков, и в результате разработана методология для системы контроля качества, которая может обеспечить контроль выполнения поставленных задач, а также создан набор инструментов, основанный на методологии.

Моим вкладом в созданную систему является создание инструмента для записи, воспроизведения и сравнения трасс, модуля оценки тестового покрытия и автоматизация всего процесса тестирования.

Для системы было создано более 100 модульных тестов для контроля над разработкой инструмента миграции. Постоянный контроль над качеством трансформации конструкций позволил значительно сократить время на поиск небольших ошибок, которые, тем не менее, сильно влияют на результат миграции. Ежедневная компиляция процедур, сгенерированных свежей версией инструмента, в Oracle позволила обнаруживать регресс, вызванный изменениями в предыдущий день, что значительно упростило поиск ошибок. Каждый из этих этапов тестирования занимает считанные минуты, что практически мгновенно по сравнению со временем проведения полного процесса миграции.

Тестирование мигрированного кода выполняется более чем на 400 трассах. Весь процесс воспроизведения занимает около 6 часов. В результате исследования расхождений, найденных в ходе воспроизведения трасс, было выявлено огромное число ошибок. Среди них были те, которые обнаружили некорректную трансляцию, а также указали необходимость подкорректировать выбранные проекции. Более того, этот метод тестирования позволил обнаружить ошибки в исходной БД. Например, были найдены части кода, в которых запросы возвращали первый элемент из неупорядоченной выборки. Результат такого запроса неопределен, и, потенциально, он может привести к некорректной работе БД.

Вычисление тестового покрытия показало, что тестовые сценарии, поставляющиеся с исходной БД, покрывают только 14% операторов. Добавление дополнительных синтетических трасс позволило увеличить

покрытие до 33%. Кроме этого, во время анализа исходной БД было выявлено огромное количество мертвого кода (более 40% операторов), что также влияет на объем тестируемой функциональности. В требованиях к проекту было необходимо покрыть 20% операторов (из принципа Парето они затрагивают 80% функциональности), поэтому результирующее тестовое покрытие считается достаточным.

Первая реализация валидации на основе подсчета числа строк в таблицах была очень быстрой, и позволила обнаружить несколько потерь при миграции на ранних этапах проекта. Следующая реализация на основе хэш-ключей помогла улучшить процесс миграции данных. Она обеспечивает достаточно высокую вероятность, что все ошибки миграции данных были обнаружены. В отличие от первой реализации процесс занимает достаточно много времени (примерно 6 часов на тестовых серверах), но это значительно быстрее, чем полная валидация.

Заключение

В рамках дипломной работы была разработана комплексная методология оценки качества для тестирования проекта миграции БД. Методология была реализована и внедрена в проект. Созданный комплекс инструментов автоматизировано позволяет обнаруживать ошибки, как в результатах миграции, так и в исходной БД, а также значительно ускоряет разработку инструмента миграции и поиск ошибок. Методология была применена в реальном проекте и показала свою эффективность.

Список литературы

[1] *Chang-Yang Lin* Migrating to Relational Systems: Problems, Methods, and Strategies // Contemporary Management Research, Vol. 4, No. 4, December 2008, Pages 369-380

[2] *Elbaum S., Chin H., Dwyer M.* Carving differential unit test cases from system test cases // FSE'06, 2006.

[3] *Haller K.* Data Migration Project Management and Standard Software – Experiences in Avaloq Implementation Projects // Proceedings of the DW2008 Conference, St. Gallen, Switzerland, 2008

[4] *Hudicka J.* The Complete Data Migration Methodology // Dulcian Inc., June

[5] *Maatuk A.* Migrating Relational Databases into Object-Based and XML Databases // Doctoral thesis, Northumbria University, 2009

[6] *Manjunath T., Ravindra S.,Mohan H.* Automated Data Validation for Data Migration Security // International Journal of Computer Applications (0975 – 8887)Volume 30– No.6, September 2011

[7] *Matthes F., Schulz C., Haller K.* Testing & quality assurance in data migration projects // Software Maintenance (ICSM), 2011 27th IEEE International Conference.

[8] *McKeeman W.* Differential testing for software // Digital Technical Journal, 1998

[9] *Patil S., Royy S., Augustinez J.* Minimizing Testing Overheads in Database Migration Lifecycle // The 16th International Conference on Management of Data (COMAD), 2010

[10] *Sneed H.M.* Selective Regression Testing of a Host to DotNet Migration // Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference J. ds, 1892, pp.68–73.

[11] *Wu B., Lawless D., Bisbal J.* Legacy System Migration: A Legacy Data Migration Engine // Proccedings of the 17th International Database Conference, October, 1997. pp 129-138

[12] Microsoft Developer Network Library, http://msdn.microsoft.com/en-us/library/default.aspx

[13] Документация Oracle, http://www.oracle.com/pls/db112/homepage