МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра общей информатики

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Ковешников Михаил Геннадьевич

Автоматизация загрузки больших массивов данных предметной области в промышленную БД

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Руководитель
Ковалёв С.П.
в.н.с. КТИ ВТ СО РАН, к. ф.-м. н.

……..
(подпись, дата)

Автор
Ковешников М.Г.
студент ФИТ, гр. 9205

..……..
(подпись, дата)

Новосибирск, 2013г.

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра общей информатики

УТВЕРЖДАЮ

Зав. кафедрой……
(фамилия, И., О.)

……
(подпись, дата)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студент*у (ке)* Ковешникову Михаилу Геннадьевичу

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Тема: Автоматизация загрузки больших массивов данных предметной области в промышленную БД

Исходные данные (или цель работы): автоматизировать существующий процесс загрузки данных в базу данных, который использует процедуры PL/SQL и таблицы Excel

Структурные части работы: работа включает в себя рассмотрение процесса загрузки данных в базу данных, обзор существующих методов загрузки данных и форматов хранения данных и их сравнение, определение частных задач загрузки данных, выявление требований к необходимому средству автоматизации, обзор существующих средств, разработку нового средства для автоматизации загрузки данных.

Оглавление

Введение

Энергетика является одной из самых важных отраслей в жизни, как отдельного человека, так и страны в целом. В этой отрасли активно проводятся исследования и разработки для повышения энергоэффективности и энергосбережения, предпринимаются попытки освоить новые источники энергии. Однако имеются и другие не менее важные проблемы. Согласно [1] в настоящее время объёмы данных, которые обрабатывают различные автоматизированные системы, растут. Так и в области энергетики необходимо контролировать большое количество объектов и ресурсов, получать результаты измерений со счётчиков. Чаще всего такая задача не под силу человеку напрямую, и поэтому возникает необходимость в автоматизированных средствах сбора данных.

Чтобы предоставлять пользователю системы собранную информацию её необходимо где-то хранить. Таким образом, в данной предметной области существует необходимость хранить большие объемы данных, как, например, НСИ, реестры оборудования, результаты оперативных измерений. Эти данные хранятся либо в файлах для представления табличной информации, либо в базах данных.

Также возникает необходимость иметь первичное наполнение базы данных сразу после развёртывания системы или для тестирования системы. В первичное наполнение включается нормативно-справочная информация (НСИ) и некоторые заранее известные объекты, ресурсы или точки измерения. Эта информация требует редкой актуализации или вовсе не требует её. Под НСИ понимаются справочники, словари, классификаторы, иерархии ресурсов.

Чтобы загрузить описанную выше информацию в базу данных необходимо приложить усилия, чтобы избежать таких проблем, как дублирование информации, нарушение целостности. Также иногда возникает необходимость актуализировать данные, хранящиеся в базе. В результате, чтобы минимизировать риск ошибок, процесс загрузки необходимо автоматизировать. Эта задача усложняется тем, что загрузка больших объёмов данных сопряжена со значительными тратами времени и о том, что данные содержат ошибку, можно узнать только через значительное время после начала загрузки, и поэтому автоматизация этого процесса выгодна.

Данные, которые могут ссылаться на другие значения, попадают под одно из определений большого массива данных. Согласно [2] большие данные - это такие данные, размер которых вызывает проблемы. В данном случае, проблемой является то, что операторам становится трудно управлять загрузкой данных в виду многих перекрестных ссылок.

Таким образом, можно сформулировать цель работы: автоматизировать загрузку данных в базу данных частично или полностью. Чтобы решить эту задачу необходимо рассмотреть способы загрузки, выявить среди них наиболее удобный для автоматизации, затем рассмотреть существующие решения, проанализировать их, выбрать подходящее или разработать новое. Также полезно рассмотреть форматы для хранения данных и более частные задачи, возникающие при загрузке данных - это дает больший объем информации для выявления требований к необходимому проектному решению.

В ходе работы были рассмотрены различные форматы для хранения данных, способы загрузки в реляционную базу данных, проведено их сравнение. Был подобран метод загрузки, который достаточно гибок для автоматизации и удобен для использования. На основе этого метода был разработан редактор, автоматизирующий загрузку данных, осуществляющий проверку целостности до проведения загрузки, и снижающий зависимость от формата хранения данных.

Результаты работы были доложены на 51-ой Международной научной студенческой конференции «Студент и научно-технический прогресс»[3].

1. Процесс загрузки данных

Чтобы рассмотреть способы загрузки данных, необходимо разобрать процесс перехода информации из некоторого источника в базу данных.

1.1 Сбор данных

На первом этапе оператор получает данные из источника. Сбор информации является очень важным этапом в процессе наполнения базы данных, так как, ошибки, допущенные на этом этапе, почти невозможно исправить. Поэтому чаще всего сбор точной информации, такой как показания счётчиков и информации о состоянии системы происходит автоматически. Однако не все источники данных могут быть поставлены на автоматический сбор, либо это сопряжено с большими трудозатратами. Для них осуществляется ручной сбор. Такими источниками информации являются, например, карта города, сеть Интернет. Наиболее часто использующимися источниками информации являются файлы, показания приборов, БД, которые могут использоваться для миграции данных в другую БД.

1.2 Преобразование данных

Преобразование данных может состоять из нескольких процедур, изменяющих формат данных. Формат устанавливается в спецификации, в которой описываются допустимые символы и некоторые ограничения – правила, которыми следует руководствоваться для того, чтобы при загрузке не возникло как синтаксических, так и семантических проблем. Кроме изменения формата системой может предполагаться кодирование загружаемых сущностей, эти коды могут, как получатся из внешних систем и справочников, так и генерироваться. Коды могут быть как порядковыми, когда каждой сущности присваивается целое число, полученное в результате инкремента предыдущего кода, так и последовательными. В последнем случае коды состоят из нескольких целых чисел, разделённых чаще всего точкой и отражающих вышестоящую иерархическую структуру. Кодирование в некоторых случаях предполагает верификацию в виде контрольных сумм. Контрольные числа используются в случае, если код состоит из нескольких цифр. Несоответствие вычисленной суммы и контрольного числа сигнализирует о возможной ошибке. Кроме кодирования, чтобы привести данные в необходимый формат, некоторые значения могут конструироваться из других, например, наименование сущности может конкатенироваться из диспетчерского имени и типа объекта. Правила, налагаемые на данные при загрузке, могут быть разнообразные, от указания числа символов, которое должно быть в значении, до указания справочника, в котором это значение должно быть.

Получив данные из источника, оператор преобразует, если это необходимо, их в приемлемую форму, при этом могут использоваться автоматические средства для приведения к корректному формату. Преобразование заключается в приведении в установленный формат для данного типа данных. Кроме этого в процессе преобразования можно получить метаданные для дальнейшей обработки или для более аккуратного сбора информации, осуществляемого на первом этапе, или для более удобной загрузки данных. Таким образом, в результате преобразования должны получиться данные, которые недвусмысленны, полны, правильны и согласованы между собой. Согласно [4] эти термины можно объяснить следующим образом: данные являются правильными, если значения и описания отражают связанный с ними объект безошибочно и точно. Недвусмысленность данных подразумевает то, что если информацию можно растолковать несколькими способами, то в результате её обработки необходимо добавить некоторый отличительный признак, чтобы разрешить неоднозначность. Данные находятся в согласованном состоянии, если они удовлетворяют единой схеме обозначений, формату. Данные отвечают требованию полноты, если, во-первых, все значения данных определены для каждого объекта, возможно, за исключением информации, верное значение которой неизвестно на данный момент, а, во-вторых, общее число записей в данных совпадает с необходимым количеством и ни одна запись не потеряна.

1.2.1 Средства для преобразования данных

Для такой задачи, как поиск и исправление орфографических ошибок, существует несколько подходов. Эти подходы схожи по функциональности, а различия между ними состоят в том, где происходит обработка запроса и где расположены данные-словари с корректными словами и их формами.

Первый подход заключается в использовании сторонних библиотек для проверки орфографии, таких как Jortho, Jaspell, Jazzy, Hunspell и других. Такие библиотеки используют файлы особого формата – словари. Такой файл обычно состоит из слов, разделенных символом перевода строки. Далее библиотека имеет некоторую структуру данных, которая позволяет искать по словарю совпадения и таким образом проверять правильность введенного слова. Также во многих из них имеется возможность получить список наиболее близких слов к слову, которое не было найдено в словаре. Как логика, так и словари, по которым ведется поиск, среди таких видов решений находятся на стороне клиента, встроенные в приложение.

Второй подход заключается в использовании веб-сервисов Яндекс.Спеллер, google-spelling-api, также имеются сервисы у Bing, Yahoo. Популярные поисковые системы

реализуют подсказки при введении части слова или исправления запроса пользователя на более вероятный вариант. Таким образом, они имеют некоторые алгоритмы, которые позволяют это делать и открывают их для разработчиков в виде сервисов. Для их использования необходимо послать запрос к серверу, с которого доступен сервис. Часть сервисов требуют предварительной регистрации и включения в запрос идентификатора разработчика или приложения и ограничивают использование своих сервисов, а некоторые позволяют обращаться к ним анонимно. При использовании такого рода услуг пользователю не надо заботиться об актуальности словаря, по которому ведется поиск, однако с другой стороны у него нет возможности добавить новое слово.

Название	Поддержка языка программирования	Поддерживаемые языки словарей	Способ взаимодействия	Комментарий
JOrtho	Java	Словари из Wikitionary (В том числе русский).	Использование локальных словарей	Возможность генерировать и обновлять словари.
JaSpell	Java	Английский; Португальский.	Использование локальных словарей	Для поиска используются тернарные деревья.
Jazzy	Java	Английский.	Использование локальных словарей	
Hunspell	C++, Java и многие другие	Большое число различных словарей (наличие русского словаря).	Использование локальных словарей	Существуют портированные версии для многих языков программирования, включая Java (через JNA,JNI).
google-spelling-api	Java	Языки, поддерживаемые сервисом Google	Запрос к сервису через сеть Интернет	Данный сервис больше не поддерживается
Яндекс.Спеллер	JavaScript, любой	Английски; Русский; Украинский.	Запрос к сервису через	Есть возможность

	язык, с помощью которого можно отправить HTTP запрос		интернет	проверять фразу на дублирование слов, ошибки капитализации слов.

Таблица 1. Сравнение библиотек для проверки орфографии

В таблице 1 приведены некоторые средства, которые умеют исправлять ошибки в текстах, и их характеристики. Для использования в целях исправления орфографических ошибок наиболее подходит библиотека JOrtho, так как она написана на Java, то соответственно проста в подключении в Java .приложениях. Библиотека JOrtho поддерживает многие языки, включая русский, который используется в собираемых данных на территории России. К тому же, эта библиотека не требует подключения к сети Интернет, что облегчает её использование.

Задачи, связанные с форматированием данных по необходимой маске, могут решаться стандартными средствами Java для форматирования строк. Устранение дубликатов реализуется при помощи структуры данных – множества. Более сложные и специфические преобразования необходимо проделывать при помощи кода, например, по коду ссылки на запись справочника определить наименование этой записи.

1.3 Загрузка данных

Завершающим этапом является загрузка обработанных данных в файл или базу данных. Загрузка данных может происходить, как в пустую базу данных, составляя первичное наполнение, так и в базу данных с исходным наполнением для её обновления. Обновление может происходить при изменении информации в источнике данных с разной частотой по необходимости. Обновление данных является более трудной задачей, чем наполнение первичными данными, так как могут возникнуть различные конфликты между данными, наличие дубликатов или отсутствие ключевой информации. Изменение наполнения и схемы может повлечь серьезные проблемы, как например, случайное удаление существенных данных или таблиц. Однако СУБД в большинстве случаев предоставляет механизмы, которые позволяют не допускать случайных ошибок. Тем не менее, загрузка должна быть автоматизированной и желательно должны существовать механизмы восстановления данных в случае сбоев.

При загрузке используются механизмы, которые предоставляет СУБД. Чаще всего они оптимизированы для работы с конкретной СУБД и имеют разные варианты использования в зависимости от нужд пользователя и разную скорость выполнения.

Загрузка огромного массива данных в БД является узким местом во всём процессе перемещения данных по сравнению с остальными этапами. Поэтому важно учитывать производительность процесса. Так, например, во время загрузки данных возникают накладные расходы на проверку целостности содержимого, запуск триггеров. Чтобы повысить производительность загрузки, можно использовать различные методики, такие как: использование групповых вставок, вынесение проверок целостности на этап выше, использование параллелизма. Кроме того загрузка больших объёмов данных может оказаться весьма трудной задачей ещё и из-за того, что ошибки чаще всего можно обнаружить только непосредственно во время загрузки. А некоторые из них можно и вовсе не найти и получить, например, дублирующиеся записи в таблицах (такое может произойти не только, когда на таблицу не наложены ограничения, но и когда, например, значение суррогатного первичного ключа генерируется автоматически), что ведёт к потенциальным проблемам с взаимодействием с базой данных.

Отдельно можно рассмотреть процесс загрузки первичного наполнения в БД. Данные для первичного наполнения берутся из условно-постоянной информации. Таким образом, один и тот же источник данных можно использовать в дальнейшем, при условии, что эта информация где-то хранится.

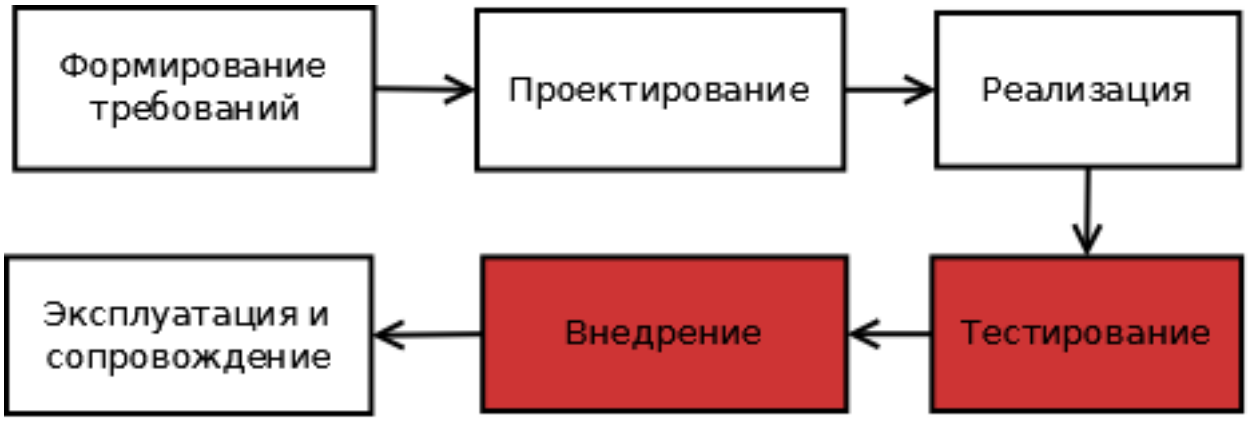

Рисунок 1. Жизненный цикл программной системы

На рисунке 1 показан возможный жизненный цикл программной системы, где этап помечен красным при условии, что на нём есть зависимость от первичного наполнения. Основная необходимость в начальных данных возникает при введении системы в эксплуатацию для корректной работы. Согласно [5] одной из обязательных процедур, которую необходимо провести во время пусконаладочных работ на стадии ввода в действие автоматизированной системы, являются загрузка информации в базу данных и проверка её ведения. Кроме того первоначальные данные необходимы и для тестирования программной системы. Хотя в большинстве случаев модульные тесты используют данные,

добавленные только на время теста, т.е. не зависят от наполнения, но тестирование пользовательского интерфейса чаще всего производится вручную и поэтому требует наличия данных для проверки функциональности элементов интерфейса.

Подводя итог, следует отметить, что процесс загрузки данных состоит из важных этапов, каждый из которых не обладает достаточной информацией, чтобы корректно исправить ошибки выполнения предыдущего этапа и полагается на его безошибочность. В результате, возникает желание повысить стабильность процесса загрузки информации в БД и минимизировать затраты на проверку ошибок. Данный процесс находит применение, как при первичном наполнении БД некоторой программной системы, так и при инкрементальном. Таким образом, возникает необходимость автоматизировать загрузку данных в базу.

2. Форматы хранения данных

2.1 XML

XML – язык разметки, который определяет набор правил для представления данных в виде XML документа. При его разработке согласно [6] одними из целей, которые ставились перед создателями, являются следующие идеи:

- XML должен быть с легкостью используемым в сети Интернет;
- XML должно поддерживать большое число приложений;
- XML документы должны быть понятными человеку;
- Размер XML документов имеет наименьший приоритет.

XML документы структурированы в виде леса так, что их можно читать без дополнительных приложений при понимании концепции вложенности тэгов. Схема XML помогает убедиться в целостности документа, проверить ограничения, налагаемые на данные, и задать возможные связи между сущностями. Также, так как все данные в XML документе имеют текстовый формат, то возможна коллективная работа над данными. Возможно использование утилит для сравнения текстовых файлов, проведения инспекций изменений между версиями одного файла.

Существует много средств для работы с XML, таких как DOM- анализаторы, SAX-анализаторы, языков запросов XPath, XQuery. Анализаторы позволяют выполнить разбор XML документа, подтвердить корректность документа или выдать сообщение о неправильной структуре файла. Существует два вида анализаторов, которые отличаются по представлению считанных данных в памяти. Древовидные анализаторы, например DOM-анализаторы, полностью считывают документ и представляют его в виде дерева или леса, доступ к которому предоставляется разработчику. В то время как потоковые анализаторы, например SAX-анализаторы, считывают документ постепенно и генерируют события по мере того, как встречаются тэги XML. Для обеспечения проверки документа либо в него включается описание, либо создается его схема. Согласно [6] документы XML можно разделить на правильно построенные и корректные. Правильно построенные документы отвечают требованиям к синтаксису XML, а корректные удовлетворяют схеме или описанию документа.

Технология XSLT применяется для преобразования XML в другие форматы. На вход XSLT процессору подаются XML документ и таблица стилей, которая определяет, как представлять те или иные данные из входного документа в результирующем файле.

Встроенные в СУБД Oracle функции для работы с XML документами позволяют форматировать данные в виде XML, агрегировать записи в виде XML дерева, конкатенировать значения.

Благодаря широкому распространению языка XML во многих языках программирования существуют, как встроенные библиотеки для работы с XML документами, так и сторонние решения. Документы XML хорошо подходят для представления и передачи структурированной информации, что делает его удобным способом для хранения данных.

2.2 CSV

Альтернативой XML для представления табличной информации может послужить CSV формат, который также является текстовым. Этот формат получил распространение за счёт широкой поддержки табличными редакторами. Основными функциями, которые работают с CSV форматом, являются импорт и экспорт, что позволяет удобно конвертировать файлы разных форматов. И как следствие он может использоваться для передачи данных между различными системами, как единый вид представления табличной информации.

CSV формат хорошо подходит для хранения последовательностей записей с одинаковым числом полей. Однако он не приспособлен к хранению иерархической информации. Тем не менее, если необходимо, можно преобразовать иерархию в плоскую структуру. Этот формат хранит только данные и небольшое количество символов для поддержки структуры и экранирования служебных символов. Соответственно, в одном CSV файле могут храниться только записи одного типа, что ограничивает возможности по представлению данных в этом формате.

Реляционная СУБД Oracle предоставляет возможность для импорта и экспорта данных в формат CSV. Для импорта данных предпочтительным способом является утилита SQL*Loader, которая позволяет гибко задавать формат файла. Также можно использовать технологию внешней таблицы для загрузки данных. В этом случае создаётся таблица, которая использует данные из плоского файла, затем происходит запроса вида INSERT INTO … SELECT, который выполняет загрузку в обычную таблицу. Для экспорта данных применяется спулинг, когда результат запроса выводится в файл, также можно производить выгрузку при помощи PL/SQL процедур. Для упрощения загрузки и выгрузки данных имеются заготовки в веб-форме.

Как и в случае с XML, для работы с данными в формате CSV существует множество библиотек для Java, которые реализуют основные функции такие, как чтение и запись (SuperCSV, OpenCSV, JavaCSV и т.д.).

2.3 XLS

Другим форматом, подходящим для хранения данных, является формат XLS. В отличие от XML и CSV он является бинарным. Также существуют и другие форматы, которые хранят данные в бинарном виде, например, ODS, XLSX. Два последних формата являются сжатыми архивами, которые содержат описание документа и данные преимущественно в формате XML. XLS формат поддерживают многие табличные процессоры. Однако встроенными средствами СУБД чаще всего не удаётся осуществить загрузку и выгрузку, используя этот формат. Поэтому необходимо первоначально произвести промежуточные преобразования. Кроме того, в связи с тем, что этот формат является бинарным, файлы в этом формате необходимо блокировать в потенциально многопользовательском окружении перед работой.

2.4 Сравнение форматов

Хранение информации в том или ином формате, даёт свои преимущества и для того, чтобы выбрать подходящий, необходимо привести сравнение по некоторым критериям.

Одним из немаловажных качеств формата является размер документа. Так бинарные форматы, чаще всего применяют сжатие для того, чтобы размер файла был меньше, однако помимо данных в файлах хранятся и метаданные, которые определяют стили и прочие дополнительные свойства данных, что оказывает существенное влияние на размер файла. Текстовые форматы хранят данные в виде текста, что увеличивает размер файла относительно заархивированного формата. Тем не менее, такие форматы как CSV, которые не содержат метаданных, при средних и малых объемах данных занимают меньше дискового пространства, чем аналоги в форматах XLS или XML.

Другим немаловажным фактором при использовании файлов с табличной структурой является возможность совместной работы. Для бинарных файлов по умолчанию такая возможность отсутствует, так как обычные средства сравнения файлов не могут определить эффективную разницу между ними. Тем не менее, существуют сторонние средства и дополнения к системам контроля версий для сравнения таблиц в одинаковом формате, например xdocdiff.

Кроме того следует учесть особенности работы с файлами в форматах. Так, если речь идёт о бинарных файлах, то для работы с ними необходим табличный процессор и к тому же системы управления базами данных не предоставляют функций для обработки

таких форматов. Поэтому для загрузки данных в базу и автоматической обработки данных может потребовать различный объём усилий. К примеру, чтобы загрузить данные из сложного документа XML, может потребоваться произвести преобразование данных в SQL запросы.

Формат	CSV	XML[1]	XLS	XLSX	ODS
Размер среднего файла[2]	2.1 Мбайт	19.1 Мбайт	6.5 Мбайт	1.7 Мбайт	1.3 Мбайт
Размер небольшого файла[3]	49 Кбайт	541 Кбайт	103 Кбайта	39 Кбайт	70Кбайт
Коллективная работа	+	+	+[4]	+4	+4
Зависимость от дополнительного программного обеспечения	нет	нет	Табличный процессор с поддержкой данного формата	Табличный процессор с поддержкой данного формата	Табличный процессор с поддержкой данного формата
Использование для импорта данных	+	+[5]	-[6]	-6	-6
Использование для экспорта данных	+	+5	-6	-6	-6

Таблица 2. Сравнение форматов хранения данных

В результате можно сказать, что использование формата CSV подходит для хранения и преобразования табличных данных, которые не имеют иерархической структуры, между различными системами. Если же необходимо структурировать данные в виде иерархии, то целесообразно использовать XML, так как этот формат поддерживает вложенность объектов.

[1] Размеры XML документов в данной таблице представлены для формата «XML таблица», поддерживаемого Excel
[2] Таблица из 360 тысяч уникальных значений, каждое размером от 1 до 6 символов, в 10 тысячах строках
[3] Таблица из 10 тысяч уникальных значений, каждое размером от 1 до 5 символов, в 1 тысяче строк
[4] При установке дополнения к системе контроля версий
[5] Если XML файл имеет жестко заданную структуру, в противном случае необходимо использовать сторонние средства
[6] Возможно только при использовании специализированных программ

3. Подходы к загрузке данных

3.1 SQL-запросы

Один из простейших вариантов загрузки данных - это формулирование интерактивных SQL-запросов к базе данных с использованием INSERT и UPDATE выражений. Данный подход предполагает знание не только языка SQL, но и схемы базы данных. Поэтому чаще всего непосредственно формулированием запросов занимается разработчик, а оператор предоставляет данные. Таким образом, возникает существенная вероятность возникновения ошибки, так как контролировать передачу информации от источника к базе данных довольно трудно. Кроме того, учитывая, что схема базы данных в большинстве случаев нормализована, то запись в понимании оператора может быть представлена группой записей в нескольких таблицах, имеющих ссылки между собой для проверки целостности. При этом вставки должны быть совершены в определённом порядке: сначала записи, которые не содержат внешних ключей, затем те, которые ссылаются на уже вставленные данные, и так далее. При загрузке могут возникнуть проблемы с циклическими ссылками между записями таблиц, наличие таких связей чаще всего сигнализирует об ошибке при составлении схемы данных, но, тем не менее, возникновение такой ситуации вполне возможно. В этом случае необходимо выполнять загрузку с отключенными проверками целостности и включать их после выполнения вставок. Поэтому для того чтобы вставить в базу данных сущности, надо обладать компетенцией для работы с базой данных. Более того ситуация усугубляется тем, что формулированием запросов, сбором и обработкой данных занимаются люди, поэтому на качество загружаемых данных влияет и человеческий фактор. Причем чем больше происходит передач информации от человека к человеку, тем выше риск неправильной трактовки данных и возникновения ошибок. Данный процесс представлен на рисунке 2, где стрелка помечена красным цветом, если существует вероятность ошибки при выполнении указанного действия.

Данный способ загрузки имеет низкую скорость, так как на каждый запрос тратятся ресурсы СУБД: выполняется синтаксический анализ, составляется плана запроса, и происходит его выполнение. Более того если таблица индексируется по одному или нескольким столбцам, то могут возникнуть накладные расходы на реорганизацию индекса. Поэтому целесообразно включать индексирование после загрузки данных, и хотя это займет, возможно, значительное время, но оно не превзойдет суммарные затраты на работу во время загрузки. Также на скорость загрузки могут влиять триггеры, которые выполняются при каждой вставке или, если задано условие, при некотором ограничении

на входные данные. При выполнении загрузки большого объёма данных следует сделать это условие максимально строгим, чтобы не вызывать ложных срабатываний триггеров. Однако при загрузке 5 тысяч записей с использованием триггера, у которого было указано WHEN условие срабатывания, не отличилось по скорости выполнения от триггера, где та же проверка была внутри тела.

Использование запросов SQL для загрузки в базу данных переносимо между разными СУБД, так как синтаксис INSERT запросов, не использующих специфических функций, совпадает. Запросы удобно хранить в файлах для последующего использования. Эти текстовые файлы удобно хранить в системах контроля версий, так как все модификации сохраняются в виде истории и возможна одновременная работа нескольких пользователей над одним и тем же файлом. Различные изменения одно и того же текстового файла разными пользователями по умолчанию поддаются слиянию воедино. Хранение исполненных запросов для наполнения на жестком диске позволяет передавать на другие компьютеры для получения в точности такого же наполнения. Целесообразно логически разделять файл с запросами на последовательность файлов - обновлений базы данных. Такое разделение может пригодиться не только для более быстрого поиска синтаксических ошибок, но и для внесений дополнительных специфических данных для заказчика. Кроме того, обновление базы данных через запуск нескольких файлов с наполнением становится более надежным, так как при ошибке достаточно отменить изменения до последнего успешно установленного обновления. Способ внесения данных с помощью запросов удобен, когда необходимо либо внести небольшой объём данных в базу немедленно, либо когда необходимо сделать обновление некоторых данных в базе. Оба этих варианта использования базы могут диктоваться срочными требованиями заказчика к первичному наполнению или иными событиями, влекущими изменения общего первичного наполнения системы. Однако, по мере роста объёма данных для обновления, растёт и время, затрачиваемое на этот процесс. Также возрастает трудность поддержки и сопровождения, так как в случае ошибок приходится делать все новые и новые обновления. Причиной тому, что исправить ошибку наполнения проще, сделав новое обновление, является факт, что источник проблемы довольно трудно найти и могут существовать данные, которые уже зависят от этого наполнения. Данные обновления помимо всего прочего надо хранить и упорядочивать, чтобы процесс установки происходил в правильной последовательности. Для этого необходимы метаданные об обновлениях, позволяющие однозначно определить порядок их установки. При большом количестве таких обновлений и появлении ответвлений в дереве зависимостей возникает

необходимость в автоматическом разборе этих метаданных для упорядочивания обновлений для установки.

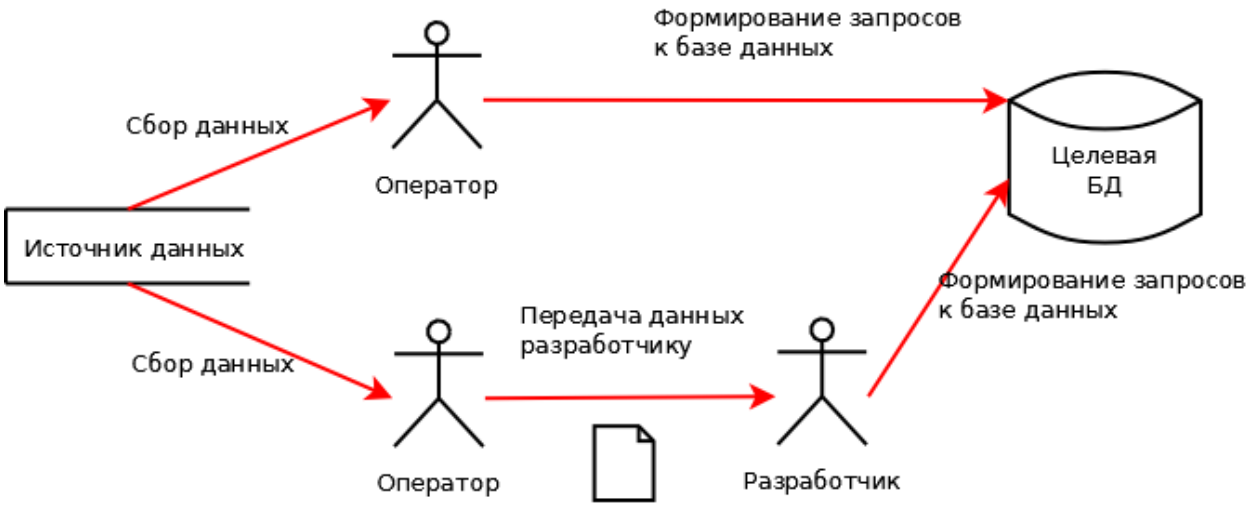

Рисунок 2. Процесс загрузки данных в БД с использованием интерактивных SQL запросов

Недостатки подхода:

- необходимо знание схемы базы данных и языка запросов к БД (SQL);
- легко допустить ошибку, как в самих данных, так и в генерации SQL-запроса, если процесс генерации не автоматизирован;
- трудоёмкость использования и освоения;
- для многократного использования необходимо сохранение SQL-запросов в файлах. Объём файлов растет по мере увеличения количества информации. Файл содержит операторы и конструкции языка запросов, что может негативно отразится на занимаемом объёме памяти;
- необходимо упорядочивать исполнение запросов при наполнении из существующих файлов;
- смена схемы базы данных влечёт исправление всех запросов, связанных с ней;
- затруднительна обработка и модификация данных, которые сформулированы в виде SQL-запросов;
- низкая скорость, обусловленная низкой скоростью генерации запросов и их выполнением с проверками целостности;

Преимущества:

- переносимость между СУБД;
- ошибки сразу видны при выполнении одиночного запроса;
- возможность дополнения базы данных в виде дополнительных SQL-запросов;

- различные версии текстового файла с запросами можно сравнивать стандартными средствами систем контроля версий и соответственно выполнять слияние;

В итоге данный метод предполагает, что либо необходимо обучать операторов, которые осуществляют сбор данных языку SQL и посвящать в устройство баз данных, либо тратить время разработчиков или администратора на наполнение базы данных. Оба варианта приносят большие финансовые потери. Следует отметить, что использование построителей запросов частично решает проблемы данного метода, но как бы, то, ни было, этот метод является совершенно неприемлемым для больших массивов данных. Данное решение, тем не менее, является переносимым между разными СУБД и позволяет производить обновление данных в базе, что удобно для небольших обновлений.

3.2 Загрузка данных при помощи утилиты Import для СУБД Oracle

Существуют и иные способы загрузки данных в базу, отличные от описанного выше метода. Они используют другие механизмы, которые позволяют производить загрузку данных быстрее, нежели вставка каждой записи в таблице в отдельном запросе. Тем не менее, за счет увеличения скорости загрузки появляются ограничения на формат и тип загружаемых данных, а для некоторых способов ещё и наличие дополнительных метаданных для разбора файла.

Один из таких способов – загрузка данных в некотором бинарном формате. Данный способ использует заранее заготовленный файл – дамп. Этот файл генерируется утилитой Export и содержит данные в бинарном виде. В заголовок файла помещаются данные о сеансе выгрузки с помощью утилиты Export: имя пользователя совершившего выгрузку данных, время, версию СУБД, имя файла. Затем идут данные о схеме таблиц и типах, сопровождаемые дополнительной информацией, и перечисляются данные. Ближе к концу файла описываются ограничения. Таким образом, для загрузки в базу данных утилита Import обладает достаточным набором информации внесённой в дамп-файл.

Из-за того что файл хранится в бинарном формате система контроля версий, если она используется, не может достоверно определить, какая порция данных изменилась при редактировании файла.

Следует отметить, что версия утилиты Import должна быть не ниже версии утилиты Export, которая использовалась для создания файла - дампа.

Использование утилиты Import предполагает, что дамп был сгенерирован утилитой Export, поэтому загрузка данных из файлов в бинарных форматах генерируемых другими СУБД не представляется возможной. Однако у многих СУБД есть свои утилиты или встроенные команды для экспорта базы данных в виде файла, который содержит схему

базы и имеющиеся в ней данные. В последствие при нарушении работы СУБД и потере текущей версии базы можно загрузить данные из этого файла, потеряв только изменения, которые произошли с момента его создания. Для СУБД MS SQL Server аналогами Import и Export являются команды BACKUP и RESTORE.

Данный способ загрузки не подходит, если необходимо какое-либо вмешательство. Любые изменения могут повлечь нарушение формата и проблемы при восстановлении данных из файла. В связи с этим, хранение дампа в системах контроля версий для данного способа не приносит пользы, так как файл не подлежит одновременному изменению несколькими пользователям. Перед изменением один из них должен заблокировать файл в системе контроля версии, чтобы изменения не были утеряны. Загрузка из файла работает для конкретной СУБД, утилиты которой были использованы для генерации дампа, так как бинарные форматы таких файлов различных СУБД закрыты и не совместимы между собой. В связи с этим отсутствует переносимость между разными СУБД.

Так как утилита Import загружает данные из дампа в определённом порядке, то этот процесс протекает довольно быстро. Загрузка происходит в том порядке, в котором данные и метаданные помещены в файл дампа. Так в самом начале последовательности создаются таблицы и типы, затем загружаются данные и индексы, потом импортируются триггеры и включаются ограничения целостности. Такая последовательность помогает избежать повторных срабатываний триггеров и проблем с ограничениями при загрузке внешних ключей. То есть, в ходе загрузки ограничения целостности и триггеры включаются только тогда, когда данные уже расположены по таблицам.

Утилита Import позволяет загружать данные из дампа в уже созданные таблицы, однако при этом возникают дополнительные трудности: необходимо вручную отключать ограничения, наложенные на данную таблицу, таблицы должны иметь совместимые схемы. Кроме того можно вручную управлять процессом импорта, постепенно загружая данные из дампа. Также необходимо позаботиться о ручном создании системных триггеров, если таковые имелись в базе, из которой происходил экспорт, так как они не попадают в дамп.

Утилита Import имеет четыре варианта работы:

- полная загрузка;
- загрузка пространства таблиц;
- загрузка объектов определенного пользователя;
- загрузка определенных таблиц и секций.

Во время полной загрузки перемещаются все объекты базы данных, однако чтобы провести импорт в этом режиме необходимо обладать привилегией `IMP_FULL_DATABASE`.

Режим пространства таблиц позволяет загрузить определённые группы таблиц. Также необходимо обладать привилегией указанной выше, чтобы осуществить данный вид загрузки. Режим загрузки объектов некоторого пользователя позволяет загрузить таблицы и другие объекты базы данных. Каждый пользователь может загружать все объекты, принадлежащие ему, однако для загрузки чужих объектов необходимо быть привилегированным пользователем. Режим загрузки таблиц загружает таблицы и секции текущего пользователя, привилегированный пользователь может загружать любые таблицы.

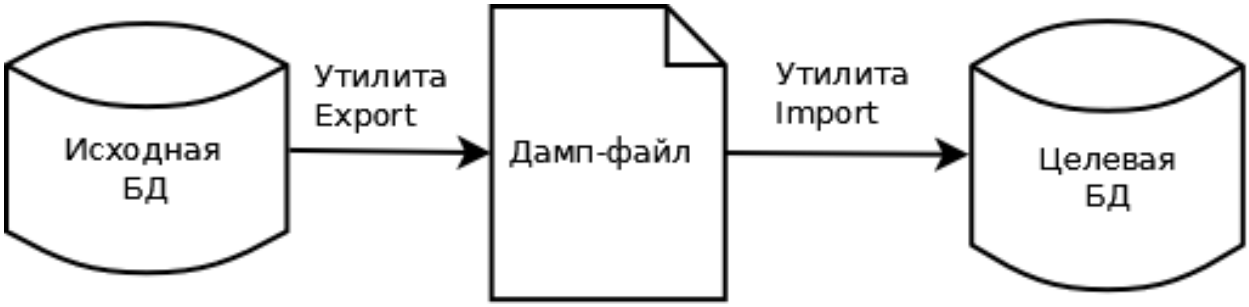

Рисунок 3. Загрузка данных с помощью утилиты Import

Плюсы:

- простота загрузки данных из дампа;
- дамп занимает небольшое количество дискового пространства;
- относительно высокая скорость загрузки;
- вероятность совершить ошибку при загрузке почти отсутствует.

Минусы:

- необходимость знания схемы базы данных при обновлении;
- модификация и обработка файла дампа невозможна;
- схема целевой базы данных и импортируемой базы данных должны быть совместимы или совпадать;
- дамп не переносим между различными СУБД;
- нельзя определить по двум файлам, какие данные в них схожи, а какие различаются;
- данные из различных файлов нельзя слить воедино без загрузки в базу;

Данный способ также не подходит, так как он совершенно не гибок и не позволяет никаким способом вносить изменения до загрузки.

3.3 Использование SQL*Loader

Утилита SQL*Loader позволяет загружать данные в базу из текстовых файлов. Этот инструмент позволяет загружать данные из нескольких файлов с заполнением нескольких таблиц за одну сессию загрузки. Файлы с данными могут иметь разнообразный формат,

так как SQL*Loader позволяет задавать разделители. В приложении А представлен пример конфигурационного файла, который загружает записи в зависимости от значения в первом столбце в разные таблицы базы данных. При загрузке имеется возможность использовать последовательности для генерации первичных ключей. Однако проставление внешних ключей затруднено, так как загрузка выполняется последовательно для каждой таблицы, а не по мере встречаемости записей в файле для загрузки и таким образом не удаётся получить промежуточное значение последовательности, на которое ссылается внешний ключ. Утилита позволяет производить селективную загрузку и применять к загружаемым данным SQL функции.

Кроме этого, при загрузке можно использовать два режима: обычный и прямой. При обычном способе из данных формируются вставки SQL для загрузки в базу, в то время как при прямом, SQL не используется, а блоки данных форматируются непосредственно. Также SQL*Loader может выполнять загрузку параллельно в нескольких потоках.

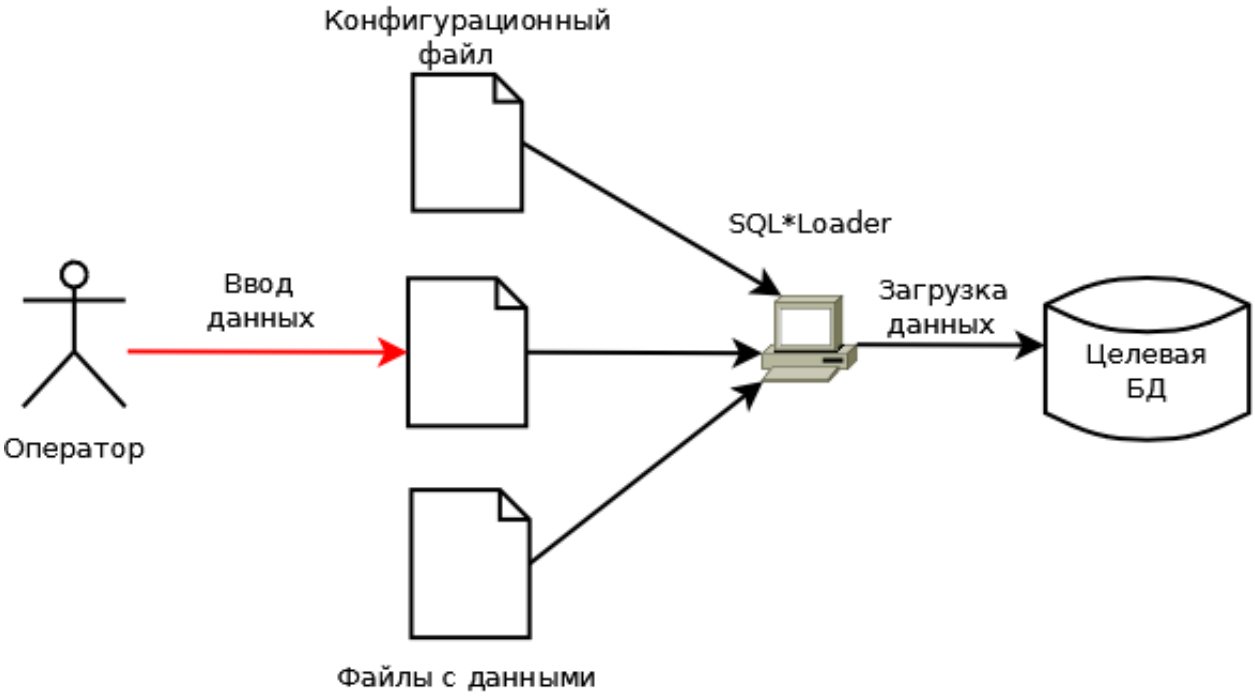

Рисунок 4. Загрузка с использованием SQL*Loader

Тем не менее, чтобы произвести загрузку данных, необходимо знать схему базы данных. Это может существенно осложнить написание конфигурационного файла, который используется для управления процессом загрузки. Текстовые файлы, которые используются для загрузки данных, могут быть изменены любым текстовым редактором, однако вероятность ошибок при таком наполнении довольно большая, так как большие блоки данных могут быть разной длины, при этом визуально разрушая форматирование. Так как данные и конфигурационный файл имеют текстовый формат, то возможна

одновременная работа над этими файлами и последующее слияние изменений. Также при использовании систем контроля версий можно получить историю изменений.

Утилита SQL*Loader используется исключительно для загрузки в СУБД Oracle, поэтому данный способ является не переносимым между СУБД. Однако существуют аналогичные утилиты у других СУБД, позволяющие загружать данные из плоских файлов, имеющих заданный формат, например для MS SQL Server это – «bcp». Разница между ними в том, что они могут поддерживать разные режимы загрузки и допускать разные форматы данных. Кроме того надо убедится, что утилита загрузки для конкретной СУБД умеет загружать в несколько таблиц и работать с несколькими файлами, если эта функция необходима. Таким образом, если схема базы данных довольно проста, то использование формата CSV(Comma-Separated Values) будет переносимым для загрузки на различных СУБД в большинстве случаев.

Плюсы:

- высокая скорость загрузки;
- гибкость формата файлов с данными;
- возможность многократного использования файлов с данными и конфигурационного файла при неизменной схеме;
- возможность одновременной работы над данными и конфигурационным файлом;

Минусы:

- необходимо знание схемы БД;
- трудно редактировать файлы наполнения;
- высока вероятность допустить ошибку или опечатку при вводе или редактировании наполнения;
- трудно написать конфигурационный файл в случае с большим количеством таблиц в схеме базы данных;
- изменение схемы приводит к изменению конфигурационных файлов и всех файлов с наполнением;
- данный способ частично переносим между СУБД.

Учитывая условие, что как схема БД, так и сами данные могут измениться, данный метод не подходит в данном случае. К тому же редактирование форматированных текстовых файлов, которые предназначены для загрузки, сопряжено с большими рисками возникновения ошибок, так как человек непосвященный в тонкости формата может, например, использовать разделитель как часть значения и тем самым, нарушив всю процедуру загрузки.

3.4 Использование Excel + PL/SQL

Чтобы облегчить загрузку больших массивов данных, необходимо иметь возможность не только хранить файлы, но и модифицировать их. Это позволит использовать данные неоднократно и обновлять их по мере необходимости. Также появится возможность исправления ошибок наполнения, если они будут обнаружены при загрузке данных. А затем, используя наполнение этих файлов, сгенерировать файлы с вызовами процедур или SQL-запросами для загрузки в БД, которые впоследствии можно удалить, чтобы освободить место.

В качестве формата файлов можно использовать любой удобный формат представления табличной информации, для которого существует возможность удобного редактирования. Наиболее используемый формат - Excel-файлы(.xls). Также аргументом в пользу этого формата является поддержка макросов. Учитывая ограничения, имеющиеся на количество записей в файле, данные могут быть распределены по нескольким файлам. Однако, большинство форматов файлов, в которых хранятся табличные данные, имеют бинарный вид, что препятствует как одновременной работе с ними, так и определению различий и сходств данных в файле потому, что системы контроля версий по умолчанию не могут сравнить файлы в бинарном формате. Тем не менее, существуют различные дополнения к программам сравнения файлов и программы для нахождения сходств, различий и слияния изменений некоторых наиболее популярных форматов.

Кроме того существуют API для работы с xls файлами, например, для языка Java это Apache POI и JExcel API. Это позволяет писать сложные приложения для работы с Excel файлами.

С помощью макросов или программы можно сгенерировать вызовы PL/SQL процедур для корректных вставок в базу данных. Процедуры позволяют производить сложные вставки, когда наполнение происходит сразу в несколько таблиц со связями между ними. Однако макрос для генерации файла наполнения и сами PL/SQL процедуры, осуществляющие загрузку, необходимо написать. Так как макрос генерации можно сделать довольно гибким, то его можно написать единожды. Это верно и для PL/SQL процедур, их изменение необходимо только в случае коренной смены схемы БД.

Данный метод частично позволяет менять СУБД с тем ограничением, что для этой СУБД должна проводиться генерация файлов для наполнения специфичным образом. Это необходимо, так как синтаксис создания процедур выполняющих загрузку данных может различаться при переходе на другую СУБД. Таким образом, для каждой СУБД, на которую требуется иметь возможность загрузить данные, необходимо создать свой VBA-макрос для генерации вызовов процедур и написать эти процедуры.

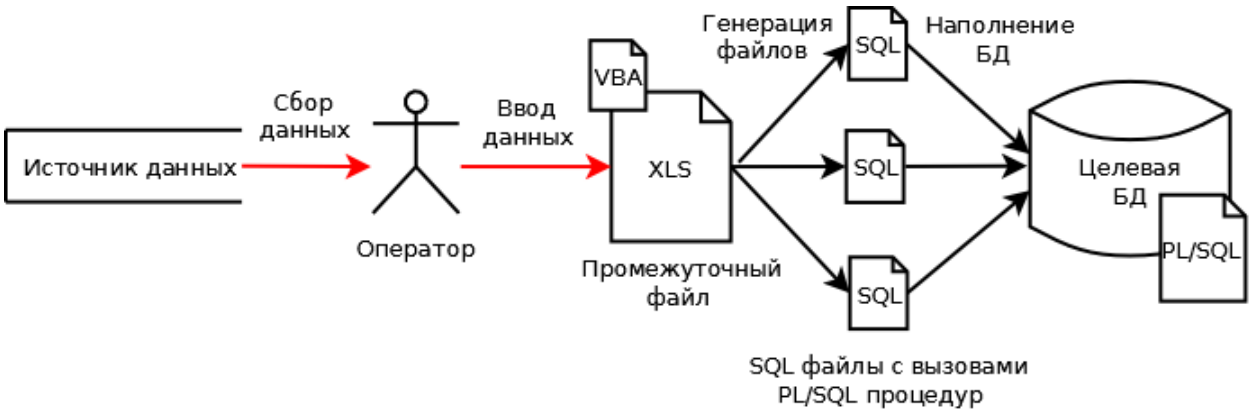

Рисунок 5. Загрузка данных с использованием Excel и PL/SQL.

Для выполнения такого рода загрузки оператору достаточно уметь работать с табличным редактором, в котором осуществляется редактирование данных. Однако остается не разрешенной проблема человеческого фактора: возможны ошибки и опечатки во время наполнения и модификации файла.

Кроме того, данный метод обладает более высокой скоростью загрузки по сравнению с простыми SQL вставками. Для того чтобы вставить 35 тысяч записей в несколько таблиц баз данных со связями при использовании SQL запросов потребовалось 475 секунд, в то время как при использовании процедур PL/SQL на этот процесс потребовалось 121 секунда. Это объясняется тем, что процедура компилируется один раз и запрос в ней разбирается единожды, в то время как для каждой SQL вставки приходится проводить разбор.

Плюсы:

- простота обработки и модификации данных;
- возможность многократного использования;
- нет необходимости знать схему БД при наполнении файла данными;
- файл хранит только необходимую информацию, операторы для вставки генерируются автоматически и после загрузки их можно удалить;
- простота использования и освоения;
- гибкость: при смене схемы БД необходимо переписать только процедуры для вставки;

Минусы:

- низкая скорость загрузки больших массивов данных
- возможны ошибки во время наполнения или модификации файла, обнаружение которых отложено на момент загрузки;
- дополнительные трудозатраты для возможности загрузки в базы под управлением дополнительных СУБД;

- необходимо написать макрос генерации вызовов процедур и сами процедуры.

- Excel файлы по умолчанию нельзя сравнивать и находить отличия между несколькими версиями одного файла.

3.5 Сравнение методов загрузки данных

В таблице 3 приведено сравнение вышеописанных методов загрузки данных.

Критерий	SQL вставки	Import	SQL*Loader	Excel + PL/SQL
Возможность модификации данных	Да	Нет	Да	Да
Простота модификации данных	Модификация затруднительна	Невозможно, модификация только в базе данных	Модификация затруднительна	Модификация проста
Знания о схеме базы данных	Необходимы	Необходимы	Необходимы	Не требуются (Инкапсулируются в процедурах)
Наличие избыточной информации в файле	Конструкции языка SQL	Нет избыточной информации (бинарный формат)	Нет избыточной информации	Нет
Простота использования	Требует специальных знаний - знания SQL	Необходима работа с командной строкой и утилитой Import	Необходима работа с командной строкой и утилитой SQL*Loader	Необходима работа с табличным редактором, консолью
Действия при смене схемы БД	Переписать все SQL запросы	Заново генерировать дамп файл	Переписать конфигурационный файл	Переписать процедуры
Допускается ли автоматизация	Возможна автоматизация генерации запросов и т.д.	Автоматизация запуска утилит	Возможна генерация данных для файла	Возможна автоматизация на этапах наполнения и загрузки
Скорость загрузки	Низкая	Наивысшая	Высокая	Средняя
Возможность допустить ошибку при загрузке	Высокая вероятность на многих этапах загрузки	Отсутствует	Высокая вероятность на этапе наполнения файлов для загрузки	Средняя вероятность на этапе наполнения файлов для загрузки (Из-за более удобного редактирования)
Этап обнаружения ошибки	При загрузке	При загрузке	При загрузке	При загрузке
Переносимость между СУБД	Переносимо	Непереносимо	Частично переносимо (При условии,	Частично переносимо (Необходимо

			что формат данных для загрузки совпадает)	написать процедуры для других СУБД)
Дополнительные трудозатраты помимо наполнения данными	Нет	Нет	Написание конфигурационного файла	Написание процедур, макроса или программы генерации их вызовов
Возможность параллельной модификации	Да	Нет	Да	Да (При использовании дополнений для слияния Excel файлов)
Возможность инкрементальной загрузки	Да	Да	Да	Нет

Таблица 3. Сравнение методов загрузки

Таким образом, метод с использованием Excel файлов для представления табличной информации и PL/SQL процедур для наполнения базы данных является наиболее удобным и подходящим для автоматизации способом загрузки.

4. Виды загрузки

В зависимости от задачи можно разделить загрузку данных на несколько видов.

4.1 Частичная загрузка

Задачу частичной загрузки можно сформулировать следующим образом. Имеется набор данных, в кортежах которого могут быть заполнены не все атрибуты. Предполагается, что незаполненные атрибуты могут быть заполнены в дальнейшем или если информации не хватает, оставлены в таком виде, в котором они были на момент загрузки. Необходимо осуществить загрузку имеющихся данных в базу.

Способы решения данной задачи зависят от того, как отображается кортеж из массива данных на схему БД, в которую ведется загрузка данных.

В первом случае, если незаполненный атрибут определён в той же таблице, что и загружаемый кортеж, то при вставке кортежа из массива данных можно явно загрузить значение NULL (при условии, что нет ограничения NOT NULL) или некоторое значение по умолчанию. Кроме того, если в самой схеме отношения для незаполненного атрибута указано значение по умолчанию, то при вставке можно пропустить значение атрибута.

Во втором случае, если значения атрибутов представлены отдельной таблицей, связанной с отношением, где хранятся основные данные загружаемого кортежа, внешним ключом или другим отношением, то необходимо иметь возможность настраивать поведение загрузки. Такую гибкость могут обеспечить процедуры, которые обрабатывают входные данные и учитывают схему базы данных для расположения значений атрибутов и проставления связей между объектом и его атрибутами. В результате частичная загрузка данных таким способом сводится к первому случаю с тем отличием, что отсутствие значения или значение по умолчанию, указывается в принимаемом параметре процедуры.

Средство, которое позволит выполнять данную задачу должно иметь возможность исключения как определённых атрибутов из загружаемых кортежей, так и определённых значений атрибутов. Это реализуется, например, так, что для каждой колонки и для каждой ячейки добавляется переключатель, который показывает, будет ли участвовать этот параметр или конкретно это значение при генерации вызовов процедур. Однако, необходимо продумать вопрос с хранением состояния переключателей, если оно должно сохраняться между запусками. В качестве альтернативы, можно заменять имеющиеся значения, отключенного параметра или значения, константой NULL или некоторым значением по умолчанию, что позволит подстроиться под интерфейс процедуры.

4.2 Инкрементная загрузка

Задача инкрементной загрузки заключается в том, чтобы выполнить загрузку в базу данных, в которой уже есть наполнение. Для того чтобы корректно решить эту задачу необходимо контролировать, чтобы данные, которые уже имеются в базе не повторялись и чтобы те данные, которых на данный момент нет в базе, успешно загрузились.

Учёт дубликатов при инкрементной загрузке можно производить на разных уровнях.

Контроль, осуществляемый на уровне БД, заключается в том, что проверка того, имеется ли сущность в базе, выполняется при помощи процедур или других встроенных механизмов СУБД. Например, загрузка может выполняться через процедуру, которая проверяет, имеется ли объект, идентифицируемый ключом кортежа, в базе и, если нет, то выполняет загрузку нового объекта, иначе объект считается уже загруженным. В результате пользователь указывает лишь состояние, к которому он хочет привести данные в базе, а процедуры выполняют необходимые проверки и вставки. Также данный способ удобен тем, что процедура инкапсулирует в себе знание о схеме базы данных.

Также, контроль можно проводить на уровне средства загрузки данных. Можно привести несколько способов реализации контроля над корректностью инкрементной загрузки.

Первый способ заключается в хранении сессий загрузки в качестве метаданных для каждого кортежа. То есть каждой строке приписывается номер сессии, когда она была загружена в базу. Средство, догружающее данные должно предоставлять выбор: загружать все данные со всех сессий, выполнить загрузку определённой сессии, выполнить загрузку ещё не загруженных данных. Однако чтобы реализовать такой способ загрузки необходимо решить, где хранить метаданные для каждого загружаемого кортежа. В результате ответственность за контролем над данными возлагается на пользователя, что может привести к ошибкам загрузки.

Второй способ подразумевает соединение с базой данных, куда осуществляется загрузка, и возможность проверки, имеются ли данные с таким ключом. Этот метод является аналогом подхода, когда проверки производятся на уровне БД, описанном выше, с тем лишь отличием, что контроль над корректностью загрузки производится из средства загрузки. Одним из недостатков такой конструкции является то, что приложению необходимо знать схему базы данных.

Третий способ возлагает ответственность за контролем над корректностью инкрементной загрузки пользователю. То есть пользователь сам выбирает, какие кортежи он хочет загрузить в базу данных. Такой подход может привести к ошибкам, если он применяется в чистом виде, однако он наиболее прост в реализации.

Для того чтобы найти компромисс между удобством, скоростью и надежностью, можно объединить методы контроля с разных уровней. То есть, например, использовать подход с выбором пользователя в совокупности с методом контроля, осуществляемого процедурой в базе данных. Пользователь указывает, какие кортежи он хочет, чтобы были в базе данных. Система загрузки передает эти данные процедурам загрузки и они, осуществляя необходимые проверки, производят вставку кортежей, которые не были найдены, в базу.

4.3 Корректирующая загрузка

Задача корректирующей загрузки заключается в том, чтобы иметь возможность обновлять или удалять данные из базы данных. Целью корректирующей загрузки является изменение содержимого базы данных таким образом, чтобы оно стало корректно на текущий момент.

Эта задача похожа на предыдущие две: как и в задаче частичной загрузки, должна быть возможность выбирать, какие атрибуты необходимо обновить или удалить, и также как и в задаче инкрементной загрузки, необходимо модифицировать только указанные строки, не изменяя остального содержимого. Соответственно решение этой задачи во многом похоже на предлагаемые варианты для вышеописанных задач. Это может использоваться для реализации средства загрузки, которое будет одновременно справляться со всеми задачами. Аналогично, логика загрузки данных может располагаться на разных уровнях.

При выполнении непосредственных запросов на изменение и удаление данных на уровне базы необходимо иметь три механизма, которые отвечают за вставку, изменение и удаление данных. Вставка данных реализуется процедурой, которая проверяет, нет ли кортежа с таким первичным ключом в базе данных, и выполняет запрос на вставку в случае его отсутствия. Процедура удаления выполняет аналогичные действия: поиск и выполнение запроса на удаление, если такой объект имеется. Однако удаление из базы данных со сложной схемой может потребовать более сложной логики. Например, если на удаляемый кортеж ссылаются кортежи других отношений, то необходимо выполнить каскадное удаление или изменения ссылок на NULL или некоторое другое значение. Процедура обновления также выполняет поиск кортежа, проверяет введённые значение и обновляет изменившийся кортеж.

В качестве более удобной альтернативы можно предложить использовать одну процедуру, которая выполняет все вышеперечисленные функции в зависимости от опционального параметра, значение по умолчанию которого, установлено на вставку или обновление. Данная процедура выполняет проверку, имеется ли кортеж в базе данных,

если да, то выполняется модификация, если нет, то вставка. Если же указан режим удаления, то выполняется поиск и удаление кортежа.

Для выполнения корректирующей загрузки на уровне редактора необходимо следить за историей создания записей в загружаемых данных. При создании записей добавлять им метаданные, показывающие, что этот кортеж новый и его необходимо вставить. При генерации событий загрузки, например, вызовов процедур для вставки, все данные помечаются, как загруженные. При модификации записей, которые уже были загружены, в метаданные вносится флаг того, что запись была изменена и её необходимо обновить в БД. Кроме того, необходимо учесть возможность помечать запись данных для удаления из базы. При загрузке все метаданные учитываются, и в зависимости от их значений происходит вызов процедур или методов вставки, обновления или удаления.

4.4 Трассируемая загрузка

Задача трассируемой загрузки возникает тогда, когда необходимо отследить, как данный кортеж попал в базу данных. Если организовать загрузку так, что остается возможность определить источник каждого вставленного, измененного или удалённого кортежа, то можно определить позицию ошибки и устранить её.

В зависимости от способов загрузки, можно идентифицировать источник данных по-разному. Например, данные могут попадать в базу через приложения, из средств загрузки данных или через непосредственные вставки пользователями. В любом случае, можно организовать хранение информации несколькими способами.

Один из способов хранения источника данных в виде журнала, куда вносится информация о событиях, но связи между записями журнала и вставляемыми сущностями не подразумеваются.

Другой способ заключается в создании дополнительных отношений в базе данных, которые содержат записи об источниках данных и связывают их с данными. Связь между таблицей данных и таблицей источников можно установить, как события в базе данных, которые хранят действие, дату, первичный ключ вставленного кортежа, первичный ключ источника данных. Контроль над наполнением данных в эти таблицы можно возложить на триггеры, которые будут вносить записи по событиям, или процедуры.

Если загрузка выполняется из табличного документа, то можно добавить в источник данных наименование файла, имя листа, номер колонки и строки. В случае ручной вставки можно записывать имя пользователя базы данных, выполнившего операции над данными. Аналогично и для приложения сохраняется его название, имя пользователя, и, возможно, другие отличительные черты данного способа вставки.

Преимущество использования процедур заключается в том, что можно разграничить доступ посредством предоставления процедур, которые выполняют необходимые действия и заодно отмечают события в отношениях-журналах.

В результате по событиям можно отследить историю изменений от создания кортежа до удаления. При больших объемах данных может возрасти объем отношения, хранящего историю событий, однако при таком же количестве данных трудоемкость поиска ошибки может существенно превысить расходы на дополнительную память.

5. Выгрузка данных

Задача выгрузки данных заключается в том, чтобы получить необходимую информацию из базы данных. Дополнительно на неё налагается требование определённого формата. Этот формат чаще всего указывается непосредственно при выгрузке данных. Кроме того, в случае, если необходимый формат не поддерживается по умолчанию, то требуется использовать некоторый промежуточный вариант, который можно преобразовать в необходимый.

Выгрузка может осуществляться различными способами в зависимости от необходимого формата. По умолчанию веб-форма СУБД Oracle позволяет использовать CSV, и XML как выходные форматы, однако этот сервис позволяет выгрузить только одну таблицу за сеанс. Другими стандартными средствами для выгрузки данных являются утилиты Export и Data Pump Export. Существенным ограничением в их использовании является то, что они выгружают данные в бинарных форматах, которые можно обработать только утилитами Import и Data Pump Import соответственно. Для выгрузки данных в текстовом виде можно применять спулинг, когда результаты запроса SQL выводятся не только на экран, но и в файл. В результате формат выходного файла, полученного таким образом, ограничен лишь возможностями SQL. Также можно использовать PL/SQL процедуры для вывода форматированных данных из базы в файл.

Если стоит задача выгрузить данные в XLS формате, то для её выполнения необходимо задать отображение между таблицами базы данных и листами Excel файла. По умолчанию можно принять тождественное отображение, то есть каждой выгружаемой таблице соответствует лист Excel файла. В качестве временного решения можно использовать формат таблицы XML, которая хранит данные в виде XML, но позволяет Excel работать с ним также как с XLS файлом. Тем не менее, чтобы сгенерировать XML таблицу необходимо использовать PL/SQL. Также Excel поддерживает формат CSV, что позволяет проводить дальнейшую работу с данными, выгруженными в этом формате.

В качестве альтернативы PL/SQL, можно использовать язык программирования Java в совокупности с драйвером используемой СУБД, для соединения и получения данных, и библиотекой для генерации XLS, например Apache POI, что позволяет генерировать сразу файл в XLS формате.

6. Возможности улучшения метода загрузки с использованием Excel + PL/SQL

Учитывая недостатки метода загрузки данных с использованием Excel + PL/SQL можно предложить следующие улучшения:

- добавление средства для проверки файлов;
- упрощение ввода и редактирования данных;
- использование плоских файлов для хранения данных;
- исследование нагрузочных показателей.

6.1 Средство для проверки файлов

Также как и остальные методы загрузки, способ с использованием Excel файлов и PL/SQL процедур подвержен ошибкам при модификации файла наполнения базы данных. Тот факт, что данные хранятся в табличном файле, частично упрощает ситуацию, потому что данный формат удобно читается и не требует больших усилий для освоения основных операций, доступных табличному редактору. Поэтому возникает необходимость минимизировать риск возникновения ошибки на этапе загрузки, то есть ошибки должны быть обнаружены ещё до того, как данные поступят на загрузку. Благодаря специальному формату файлов поверх табличной структуры, включающему источник данных для столбцов, можно следить, находится ли значение в определённой ячейке таблицы среди допустимых и таким образом сигнализировать о потенциальной ошибке.

Соответственно, проверка файлов позволит находить ошибки ещё до загрузки в базу данных и исправлять их. Кроме проверки всего файла должна быть возможность проверять данные по мере их ввода и редактирования, что уменьшает вероятность допустить ошибку или опечатку. Также проверка должна быть возможна, если допустимые значения распределены по разным файлам вследствие того, что один файл может физически не может вместить необходимое количество записей или необходимо логическое разделение данных по файлам.

6.2 Упрощение ввода и модификации данных

В связи с тем, что объемы данных могут быть велики, ввод новых значений должен быть простым. Если имеются некоторые ограничения на значения данных, то это должно отображаться по мере ввода. Чаще всего допустимые значения представляют в виде выпадающего списка с возможностью фильтрации по мере набора текста и выбора корректного значения. Фильтрация необходима, так как столбец, из которого берутся допустимые значения, может содержать тысячи значений. Более того значения могут браться из нескольких столбцов, что может существенно затруднить выбор значения.

При вводе данных, оператор не должен отвлекаться на системную информацию столбцов, которая располагается непосредственно над данными. Чтобы сократить количество ошибок, связанных с жестко заданным форматом системной информации, необходимо представить редакторы этих значений отдельными компонентами, которые считывают и записывают данные в определённые ячейки Excel файла.

Кроме того, необходимо предоставить возможность выбирать, какие столбцы и строки данных должны присутствовать в наполнении. Соответственно желательно выделение цветом этих столбцов и строк, которые не используются при генерации файла с вызовами процедур или которые имеют фиксированные значения. Это позволит оператору проще ориентироваться по большому количеству данных.

6.3 Использование плоских файлов для хранения данных

Одним из недостатков метода загрузки Excel + PL/SQL является то, что параллельная работа над одним и тем же набором данных затруднена, так как Excel файл является бинарным. Данную проблему можно решить, используя дополнения для систем контроля версий для выполнения слияния двух Excel файлов, либо храня данные в виде плоских файлов, например в формате CSV.

Хранение данных в виде CSV имеет также преимущество по размеру над стандартным форматом таблиц XLS. Существенным недостатком работы с CSV файлами является то, что в каждом файле может храниться только один лист данных, в то время как XLS позволяет хранить книги с несколькими листами.

Следовательно, инструмент, который будет использоваться для работы с файлами наполнения, должен предоставлять однообразный интерфейс независимо от формата данных в файлах и предоставлять механизм агрегации CSV листов в книги.

6.4 Нагрузочные показатели

Для того чтобы оценить, возможны ли улучшения в процессе загрузки по скорости необходимо оценить загрузочную способность метода с использованием Excel + PL/SQL. Для проверки скорости загрузки было создано несколько таблиц со связями между собой. Были написаны типичные процедуры, которые осуществляют проверку на тот факт, есть ли кортеж с таким ключом в базе или нет и затем в случае отсутствия, вставляют этот кортеж в базу. Также было сгенерировано разное количество вызовов этих процедур с разными параметрами, которые вставляли суммарно от 8 тысяч записей в таблицы до 280 тысяч записей. Была проведена серия испытаний с изменением количества загружаемых записей и в результате, как можно увидеть на левом графике Рисунка 6, наблюдается квадратичная зависимость.

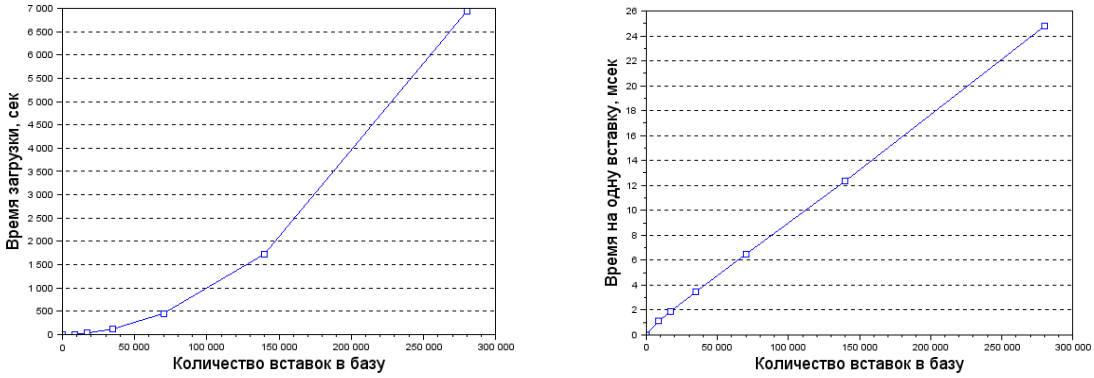

Рисунок 6. Зависимость общего времени загрузки и времени загрузки одной записи с использованием PL/SQL от количества вставок в базу

В этом случае, также можно рассмотреть время, затрачиваемое на загрузку одной записи от общего числа загружаемых записей, график зависимости, представленный на правом графике Рисунка 6, показывает, что зависимость является линейно возрастающей. Таким образом, при возрастании объема данных, загружаемого с помощью PL/SQL процедур, общее время загрузки растёт пропорционально квадрату числа записей, а время загрузки одной записи растёт линейно.

Процесс загрузки 35 тысяч записей в базу данных, которые были выполнены 20 тысячами процедурами, занял примерно 121 секунду. Из этого показателя можно получить, что в среднем одна запись при таком объёме данных загружается за 3,4 мсек. В качестве желаемого времени загрузки данного объема в 35 тысяч записей можно выбрать показатель в полторы минуты. Можно заметить, что данный метод почти удовлетворяет данному требованию. Для того чтобы достичь лучшего результата можно предложить использование групповых вставок с использованием оператора FORALL PL/SQL. Для использования этого метода необходимо изменение реализации процедур, но интерфейс остаётся тем же. При вызове процедуры вставки, кортеж помещается в буфер и как только буфер заполняется, или поступает событие о необходимости принудительного сброса в базу данных, данные загружаются в базу. Согласно [7] применение этого метода оптимизации загрузки заключается в том, что переключение контекста при выполнении вставок в базу данных между ядрами PL/SQL и SQL происходит реже и соответственно эти переключения вызывают меньше накладных расходов. С использованием данного метода оптимизации для примера в 35 тысяч записей средняя скорость загрузки данных составила 327 записей в секунду. Всего было затрачено 106 секунд, со средним временем загрузки одной записи в 3,1 мсек. Из этого можно сделать вывод, что при использовании метода загрузки с использованием Excel + PL/SQL переключения контекстов между

ядрами PL/SQL и SQL не столь сильно оказывают влияние на общее время выполнения загрузки.

Кроме возможных технологических улучшений необходимо учитывать альтернативные факторы, влияющие на загрузку данных. Например, если есть процедура, которая выполняет некоторые операции (к примеру, вычисление наименования записи по остальным полям записи) над только что вставленными данными, то важно, чтобы эта процедура выполнялась единожды для каждой записи. Иначе, если её вызывать после каждой вставки, то будет проделано много лишней работы, и временная оценка будет пропорциональна квадрату от количества записей. В то время как, если вызывать эту процедуру для всех вставленных записей в конце загрузки, то временная оценка снизится до линейной от количества записей.

Подводя итог, следует заметить, что метод оптимизации с использованием групповых инструкций вставки не подходит для данного метода загрузки, так как при увеличении объёмов данных затраты на изменение процедур превышают ускорение получаемое в результате оптимизации. Таким образом, если ключевым фактором является скорость загрузки, то необходимо предусмотреть возможность конвертации данных в другой формат с использованием другого метода загрузки, возможно с использованием утилиты SQL*Loader.

7. Предлагаемый вариант решения

Необходимо найти или разработать редактор Excel файлов, который бы удовлетворял следующим требованиям:

- возможность проверки файлов на наличие ошибок;
- фильтрация допустимых значений по мере ввода;
- возможность выбора из возможных значений;
- простота в использовании и освоении;
- предоставление графического интерфейса для взаимодействия с пользователем;
- возможность ссылок на другие файлы;
- гибкость при изменении формата данных внутри файла;
- независимость от операционной системы;
- работа с CSV файлами, как с Excel книгой;
- загрузка данных из редактора.

Был проведён поиск в сети Интернет по имеющимся табличным процессорам, которые поддерживают импорт данных из Excel файлов или позволяли бы напрямую редактировать данные в этих файлах. Многие из существующих редакторов не отличаются по функциональности от Excel и не поддерживают средства для задания связей между столбцами. В большинстве случаев в них есть механизм для выбора среди заданных значений, однако все допустимые значения необходимо либо задавать вручную путём ввода в область редактирования, что весьма неудобно для большого количества данных, либо указывать диапазон значений на листе. Последний подход для задания допустимых значений гораздо удобней, чем первый, однако при выходе данных за указанный ранее диапазон, можно не учесть нужных значений. Таким образом, необходимо, чтобы диапазон значений автоматически расширялся по мере ввода данных и изменения отражались в выпадающих списках для ввода данных.

Табличный процессор	Excel	OpenOffice Calc	Gnumeric	KSpread	Lotus 1-2-3	LibreOffice
Возможность проверки файлов на наличие ошибок	+[1]	+1	-	+1	+1	+1
Фильтрация допустимых	-	-	-	-	-	-

[1] При использовании макросов

значений по мере ввода						
Возможность выбора из возможных значений	+	+	+	+	+[1]	+
Простота в использовании и освоении	+	+	+	+	+	+
Предоставление графического интерфейса	+	+	+	+	+	+
Возможность ссылок на другие файлы	+[2]	+2	-	-	-	+2
Возможность работать с другими форматами файлов	+2	+2	-	-	-	+2
Независимость от операционной системы	-	+	+	+	-	+
Работа с CSV файлами, как с Excel книгой	-	-	-	-	-	-
Загрузка данных из редактора	+2	+2	-	+2	+2	+2

Таблица 4. Сравнение существующих табличных процессоров

Учитывая то, что объём данных может быть очень большим, и тот факт, что человек ответственный за наполнение должен иметь возможность выбрать допустимое значение из списка, фильтрация значений по мере ввода является одним из важнейших требований. Однако ни один из рассмотренных редакторов не поддерживает такой функции.

[1] При установке дополнения для проверки
[2] При использовании макросов

Следует отметить, что особую ценность табличным процессорам приносит функциональность для запуска макросов, так как иначе невозможно было бы выполнять большинство требуемых функций. Так, например, благодаря макросам можно осуществить обработку файлов в произвольных форматах, осуществить проверку целостности всего файла и некоторым образом проверять значения, входящие в другие файлы.

Большинство табличных редакторов позволяют импортировать файлы в формате CSV, однако, ни один из них не позволяет группировать логически CSV файлы для удобной работы с ними. Открытие каждого CSV файла приводит к открытию нового окна, а не новой вкладки, что вызывает неудобство работы.

Все решения, которые входят в обзор обладают примерно одинаковым графическим интерфейсом, что делает их весьма лёгкими для освоения, так как в основном люди, занимающиеся наполнением базы данных, знакомы в той или иной степени с интерфейсом и методами работы с табличными процессорами. Даже если навыки использования таблиц отсутствуют, существуют книги, документация и ресурсы в сети Интернет, содержащие основные приёмы использования.

Тем не менее, решения подходящего по всем параметрам найдено не было, поэтому возникает необходимость разработки собственного решения.

8. Редактор и спецификация файлов

Для разработки редактора использовался язык объектно-ориентированного программирования Java с использованием библиотеки Swing для создания графического интерфейса. Swing предоставляет компоненты для представления табличной информации и выпадающих списков, которые представляют наибольшую ценность для редактора.

В качестве средства для программируемого взаимодействия с Excel-файлами была выбрана библиотека Apache POI.

8.1 Редактор

Редактор предоставляет пользователю интерфейс для редактирования, проверки и загрузки данных из файлов. Данное приложение применяется на этапе ввода данных оператором в Excel файл и при загрузке. Редактор автоматизирует трудоемкую проверку, как существующих данных, так и вводимых данных и устраняет промежуточные этапы при загрузке, что автоматизирует процесс загрузки в целом. При этом минимизируется риск возникновения ошибок и опечаток.

При открытии файла выводится список ошибок, которые содержатся в файле. Ошибки могут быть следующими:

- значение не входит в диапазон допустимых значений;
- ссылка не может быть разрешена (не найдены допустимые значения);

Также имеются кнопки в меню для проверки, как всего файла, так и выбранного листа. Существует возможность сохранения измененного файла. Ведётся история недавно открытых файлов, для более быстрого доступа к ним. Для удобства имеется возможность автоматического расширения колонок, чтобы были видны значения в клетках таблицы. Реализовано создание новых файлов, листов, столбцов и строк. Редактор позволяет работать с XLS книгами и CSV файлами, агрегированными в книги для поддержания схожей структуры. В результате имеется возможность конвертировать файлы из XLS формата в CSV формат. Интерфейс пользователя одинаков при работе, как с XLS файлами, так и с CSV. Изменение соответствующих файлов происходит прозрачно для оператора.

Редактор предоставляет возможность редактирования в боковой панели системной информации:

- файла (название книги, имя файла при сохранении, ссылки на другие файлы);
- листов (название процедуры, признак включения в генерацию, название листа, имя генерируемого файла);

- столбцов (имя столбца, название параметра в процедуре, комментарий к столбцу, ссылки на допустимые значения, генерируется значение ли по столбцу, используется ли NULL или какое-то определённое значение).

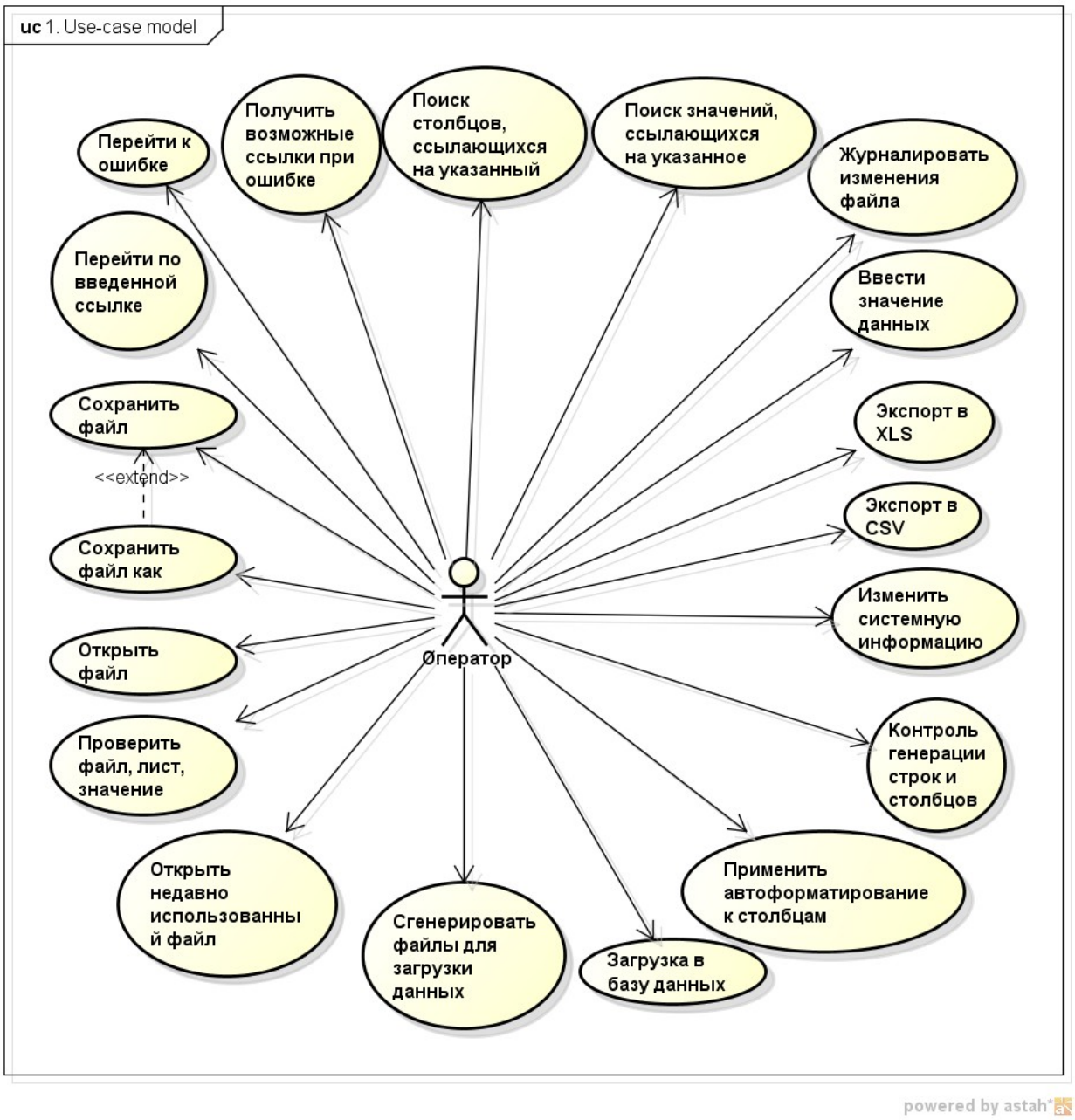

Рисунок 7. Основные варианты использования редактора.

Отдельные поля для ввода позволяют снизить зависимость от заданного формата файла и облегчить ввод и редактирование системной информации. Таким образом, при наполнении файла с использованием редактора нет необходимости знать спецификацию формата файлов, что облегчает освоение данного метода загрузки. Все изменения учитываются в журнале.

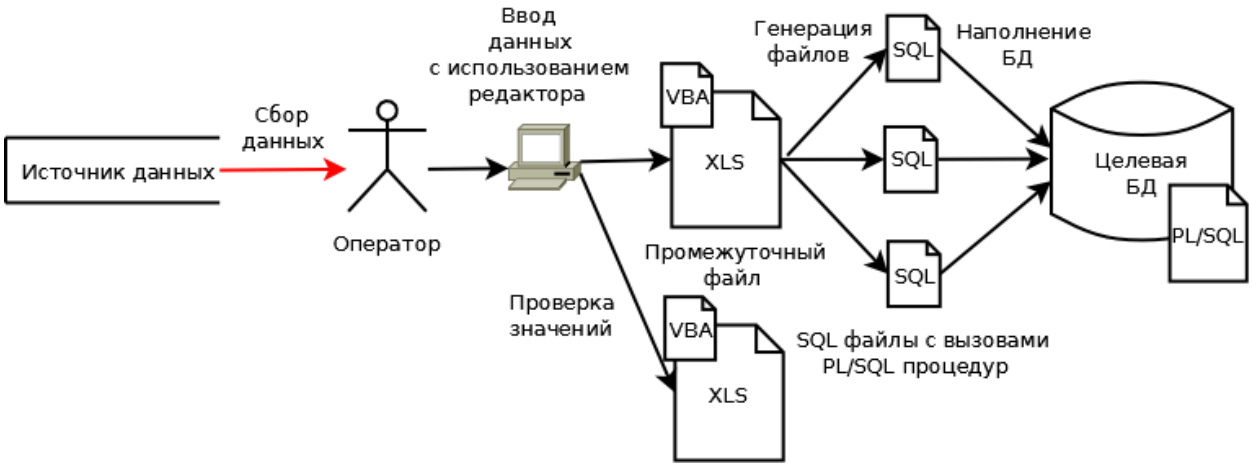

Рисунок 8. Загрузка данных с использованием редактора

8.2 Спецификация файлов

Для возможности проверки значений между файлами необходимо, чтобы проверяемые файлы имели некоторую дополнительную информацию о расположении на жёстком диске других файлов, в которых находятся допустимые значения. Для этого необходимо каждому файлу сопоставить название, которое бы указывалось в ссылке и однозначно определяло этот файл

Учитывая требования, на первом листе расположена кнопка для запуска макроса генерации и имя книги. Также если внутри файла есть ссылки на другие файлы, то должен существовать лист с названием «References», на котором указан список пар: название книги, путь до файла. Остальные листы содержат данные и системную информацию для генерации вызовов процедур и проверки целостности: название процедуры, признак включения в генерацию, название листа, имя генерируемого файла, имена столбцов, названия параметров в процедуре, комментарии, ссылки на столбцы с допустимыми значениями. Один столбец может ссылаться на несколько других, поэтому в ячейке могут указываться несколько ссылок разделённых символом “точка с запятой”.

Ссылки на столбцы имеют особый формат: <название_книги>::<название_файла>(<название_параметра>) – где <название_книги> и <название_параметра> опциональны и, по умолчанию, имеют значения текущей книги и первого параметра в файле соответственно.

Для того чтобы CSV файлы объединялись в книги, используется дополнительный CSV файл, который указывает, сколько листов содержит данная книга, их название и путь до них. Также в нем указывается количество и путь до книг, на которые ссылаются значения из текущей книги. По аналогии с обычной Excel книгой хранится название книги. Формат ссылок используется в Excel и CSV книгах одинаковый, что позволяет иметь зависимые значения в книгах другого формата.

8.3 Пример применения редактора для загрузки данных

Пусть, например, необходимо загрузить в базу данных типы счётчиков, которые используются на объектах. Оператор или иной человек, ответственный за наполнение БД, вводит информацию в редактор. Если файл с наполнением уже существует и необходимо лишь добавить записи, то вводить метаданные заново не нужно, остается ввести значения. Однако если требуется создать новый файл, то можно как создать его в редакторе, так и создать файл в Excel и затем открыть его в редакторе. При открытии файла, если на столбцы были заданы ссылочные ограничения и если они нарушаются, то появляется окно с ошибками. Кликнув дважды по ошибке можно перейти к её месту и исправить. Кроме этого существует механизм устранения ссылочных ошибок, который применяется, когда ссылка задана, но она либо не указывает на нужные значения, либо имеет неправильный формат. Чтобы использовать подсказки необходимо щелкнуть правой кнопкой мыши по ошибке и выбрать пункт «Показать возможные варианты». В этом случае появляется окно для поиска и замены указанной ссылки. При редактировании, если указаны метаданные ссылок, то при введении значения появляется выпадающий список с допустимыми значениями, который фильтруется по мере набора текста. Перед загрузкой желательно проверить содержимое файла на наличие ссылочных ошибок, выбрав одну из опций для проверки в меню на закладке «Редактировать». Убедившись, что отсутствуют ссылочные ошибки, для нужных листов установлен флаг, подтверждающий генерацию файла, введено название PL/SQL процедуры и названия её параметров для соответствующих столбцов, можно приступать к генерации SQL файлов с вызовами процедур или непосредственно загрузке данных.

Генерация может происходить двумя способами: либо, выбрав пункт «Генерировать SQL файлы» в меню на закладке «Редактировать», либо, открыв файл в Excel и нажав кнопку «Генерировать» на первом листе для запуска макроса генерации файлов.

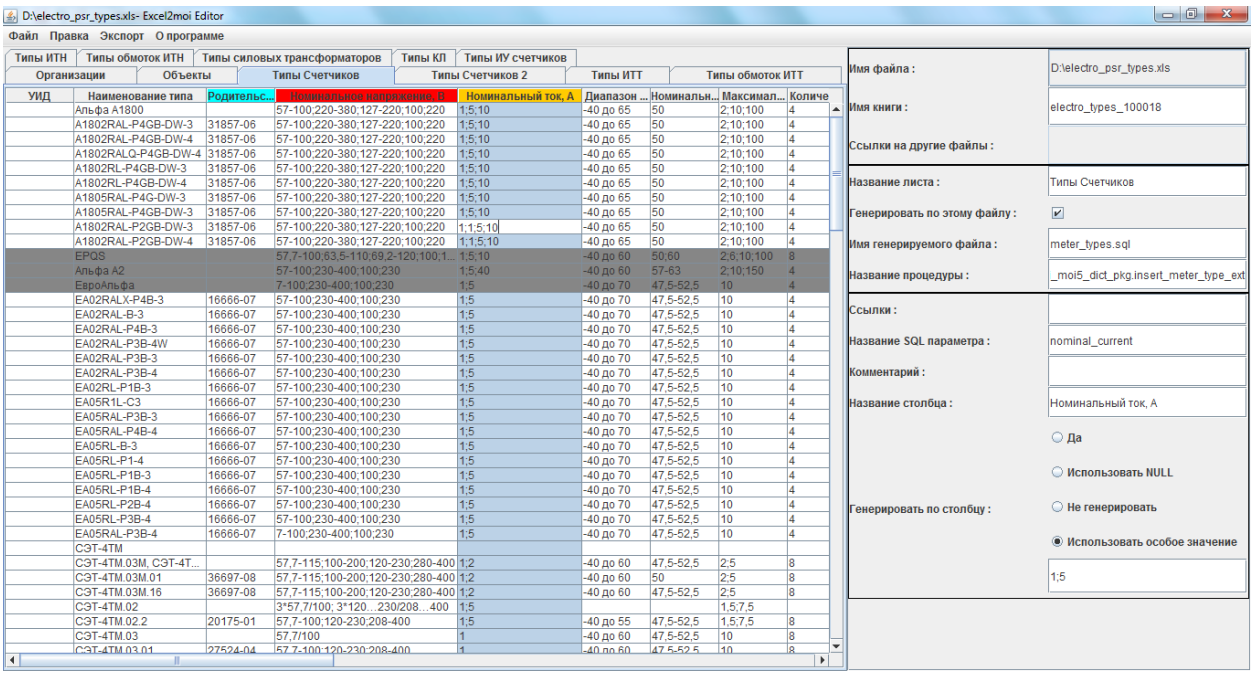

Рисунок 9. Вид данных заполненных в редакторе

После выполнения процедуры генерации файлов можно приступать к их загрузке в базу данных. На рисунке 10 показано, как выглядят данные в виде готовом для исполнения загрузки. В результате генерации получаем не только файлы с наполнением, но и файл с названием «index.sql» для запуска загрузки в определенной последовательности и файл «log.txt» с отчетом о генерации файлов.

Рисунок 10. Вид данных в виде вызовов процедур

Далее, запуская в SQL*Plus или в SQL клиенте файл «index.sql», происходит наполнение базы данных. Следует учитывать, что схема базы данных должна быть создана ещё до загрузки.

Кроме того, редактор позволяет выполнять загрузку, минуя генерацию файлов и их запуск в SQL*Plus. Для этого необходимо нажать в меню «Экспорт» на пункт «Загрузить в базу данных», а затем выбрать из списка листы, которые необходимо загрузить.

Использование такой загрузки существенно уменьшает количество этапов, которые оператору необходимо сделать при загрузке данных.

8.4 Компетенции необходимые для загрузки данных

Чтобы выполнить загрузку данных без использования редактора, пользователь должен открыть Excel файл, отредактировать его, сгенерировать по нему документ. Далее ему необходимо поместить сгенерированный файл в SQL developer, присоединится к базе данных, и выполнить его. Можно также выполнить запуск через SQL*Plus, однако это требует навыков для работы с консолью. Можно создать исполняемый файл, который будет автоматически выполнять присоединение к базе и загрузку данных. Таким образом, получается, что нельзя с большой уверенностью сказать, что обычный пользователь сможет выполнить загрузку.

В результате возникает необходимость осуществлять загрузку более простым способом, желательно не меняя при этом инструменты. Для решения такой задачи в редактор была реализована возможность загрузки данных, не выходя из интерфейса приложения. На рисунке 11 представлено окно загрузки, где оператор вводит данные необходимые для соединения с базой данных, указывает необходимые для загрузки листы и жмёт кнопку, чтобы выполнить загрузку.

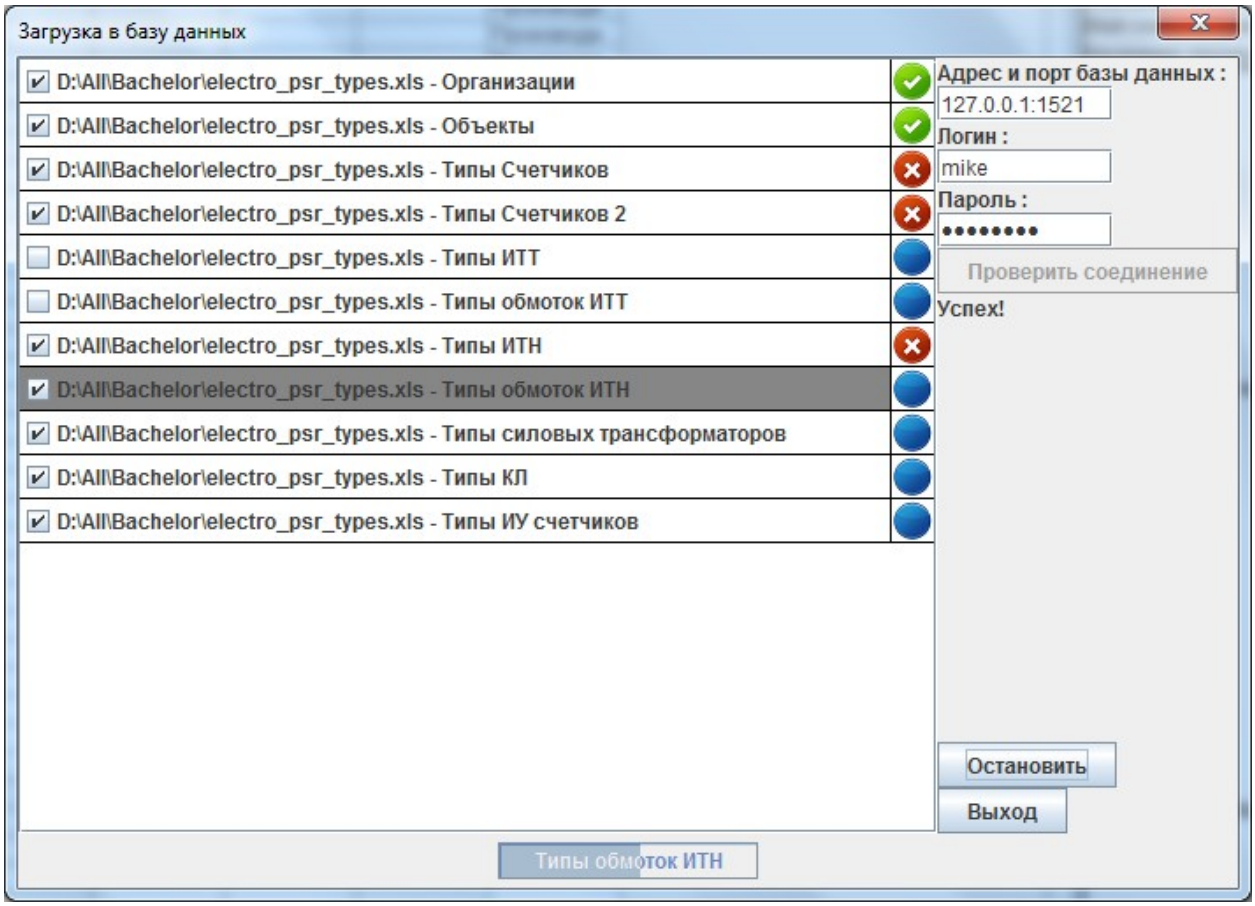

Рисунок 11. Окно загрузки данных

Загрузка выполняется в отдельном потоке, так как она может занимать длительное время и не давать пользователю взаимодействовать с графическим интерфейсом. Ход и прогресс работы отображается внизу окна. После работы возле выбранных для загрузки листов, отображается статус, который может принимать 3 значения: выполнен без ошибок, не был выбран для загрузки или ещё не выполнялся, выполнен с ошибками. Если при загрузке возникают ошибки, то по наведению на статус листа с ошибками будет отображаться ошибка во всплывающем окне.

Таким образом, оператор не выполняет переходов между инструментами и как следствие повышается надежность и автоматизация процесса загрузки. Соответственно, компетенции, которыми должен обладать оператор, упрощаются. В результате оператор для загрузки должен иметь навыки работы с табличным редактором.

Заключение

В работе рассмотрен процесс, а также способы загрузки данных в базу данных. Проведён обзор различных форматов хранения табличной информации и их различия. Проанализированы различные подходы к загрузке, среди них выявлен способ, который наиболее удовлетворяет задаче работы. Рассмотрена выгрузка данных и частные задачи, возникающие в ходе загрузки данных. С учетом поставленных требований к средству загрузки и проанализированной информации были выявлены возможности для улучшения метода. Разработано решение для автоматизации выбранного способа загрузки данных.

Разработанный редактор позволяет перенести часть накладных расходов, связанных с проверкой целостности данных, с процесса загрузки на процесс обработки / модификации данных, а также выполнять загрузку данных за меньшее количество действий со стороны оператора. Таким образом, используя полученный метод, можно добиться прироста в скорости загрузки больших массивов данных, если найти удачное средство для загрузки с возможностью отключения проверок целостности. Кроме того, за счет упрощения загрузки снижается количество этапов, на которых можно допустить ошибку. Приведён пример использования проектного решения.

Также в ходе работы была размещена информация, как о редакторе, так и о допустимом формате файлов в виде спецификации на корпоративном портале.

Используемые обозначения

Таблица 5. Аббревиатуры

Термин	Описание
БД	База данных
НСИ	Нормативно-справочная информация
СУБД	Система управления базами данных
РСУБД	Реляционная СУБД
SQL	Structured Query Language
PL/SQL	Procedural Language/Structured Query Language
API	Application Programming Interface
XML	Extensible Markup Language
DOM	Document Object Model
SAX	Simple API for XML
XLS	Excel Spreadsheet
XSLT	Extensible Stylesheet Language Transformation
CSV	Comma-Separated Values
ODS	Open Document Spreadsheet
JNA	Java Native Access
JNI	Java Native Interface
HTTP	HyperText Transfer Protocol

Литература

1. Bohlouli M., Schulz F., Angelis L., Pahor D., Brandic I., Atlan D., Tate R. Towards an Integrated Platform for Big Data Analysis. // Fathi M. Integration of Practice-Oriented Knowledge Technology: Trends and Prospectives. – Germany: Springer, 2013. - pp 47-56
2. O'Reilly Media. Big Data Now. September 2011. 128 p.
3. Ковешников М.Г. Автоматизация загрузки больших массивов данных предметной области в промышленную базу данных // Материалы 51-й Международной научной студенческой конференции «Студент и научно-технический прогресс»: Информационные технологии / Новосиб. гос. ун-т. Новосибирск, 2013. – 161 с.
4. Kimball R., Caserta J. The data warehouse ETL toolkit: practical techniques for extracting, cleaning, conforming, and delivering data. – Canada: Wiley Publishing, Inc., 2004. – 491 p.
5. ГОСТ 34.601-90. Автоматизированные системы. Стадии создания. : Межгос. стандарт. - Введ. 01.01.92 // Комплекс стандартов на автоматизированные системы.
6. W3C Recommendation. Extensible Markup Language (XML) 1.1 (Second Edition). http://www.w3.org/TR/2006/REC-xml11-20060816 .
7. Фейерштейн С., Прибыл Б. Oracle PL/SQL. Для профессионалов. 5-е изд. – СПб.: Питер, 2011. – 800 с.
8. Кайт, Томас. Oracle для профессионалов: архитектура, методики программирования и особенности версия 9i, 10g и 11g, 2-е изд. :Пер с англ.- М.: Вильямс, 2011. – 848 с.
9. Rich, Kathy. Oracle Database Utilities, 11g Release 2 (11.2). Oracle Corporation. 2011. – 728 p.
10. Хорстманн, Кей С., Корнелл, Гари. Java 2. Библиотека профессионала, том 2. Тонкости программирования, 8-е изд. :Пер. с англ. – М. : ООО «И.Д. Вильямс», 2012. – 992 с.
11. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2011. – 368 с.

Приложение А. Пример конфигурационного файла SQL*Loader загружающего данные в две таблицы

```
LOAD DATA
 INFILE *
 INTO TABLE table_a
 WHEN type = 'a'
 FIELDS TERMINATED BY ',' TRAILING NULLCOLS
  ( type  FILLER CHAR(1),
    A_ID_PK "table_a_seq.nextval",
    A_CODE,
    A_NAME
  )
 INTO TABLE table_c
 WHEN type = 'c'
 FIELDS TERMINATED BY ',' TRAILING NULLCOLS
  ( type  FILLER POSITION(1) CHAR(1),
    C_CODE_PK,
    C_NAME
  )

BEGINDATA
a,code1,name1
c,code3,name3
a,code2,name2
c,code4,name4
a,code5,name6
a,code7,name7
c,code9,name9
```