

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по выполнению лабораторных работ по дисциплине
«ОСНОВЫ ПОСТРОЕНИЯ ЗАЩИЩЕННЫХ БАЗ ДАННЫХ»
для студентов специальности 10.05.01 «Компьютерная безопасность»
специализации «Информационно-аналитическая и техническая
экспертиза компьютерных систем»

Ставрополь,
2026

СОДЕРЖАНИЕ

Лабораторная работа № 1. Исследование рисков информационной безопасности в базах данных. 4 часа.....	3
Лабораторная работа №2. Разграничение доступа к базе данных. 4 часа.....	11
Лабораторная работа №3. Использование и защита транзакций в базах данных. 4 часа.....	21
Лабораторная работа №4. Архитектура реляционных баз данных. 4 часа.....	26

Лабораторная работа № 1. Исследование рисков информационной безопасности в базах данных. 4 часа

Цель работы: приобретение студентами навыков по исследованию существующих или проектируемых баз данных на предмет наличия рисков информационной безопасности в реализации или проекте.

Теоретическая часть

При проектировании информационных систем различного назначения для хранения больших и сверхбольших объемов информации проектировщики обычно делают выбор в пользу реляционной СУБД. Такова сложившаяся практика. На последующих стадиях проектирования и разработки обеспечение безопасности базы данных (ядра всей системы) обычно сводится к выделению классов пользователей, их информационных потребностей и привилегий (эти и еще несколько этапов входят в формирование политики безопасности), проектированию системы разграничения доступа.

Для назначения/отмены привилегий используется язык SQL, включающий операторы GRANT, REVOKE и т.п. Большинство современных реляционных СУБД поддерживает дискреционную (DAC) и мандатную (MAC) модели разграничения доступа, а также дополнительные средства обеспечения безопасности.

Под угрозой обычно понимают потенциально возможное событие, действие (воздействие), процесс или явление, которое может привести к нанесению ущерба чьим-либо интересам.

Угрозой информационной безопасности автоматизированной информационной системе (АИС) назовем возможность воздействия на информацию, обрабатываемую в системе, приводящего к искажению, уничтожению, копированию, блокированию доступа к информации, а также возможность воздействия на компоненты информационной системы, приводящего к утрате, уничтожению или сбою функционирования носителя информации или средства управления программно-аппаратным комплексом системы.

Источники угроз информации баз данных

Разработка системы информационной безопасности должна базироваться на определенном перечне потенциальных угроз безопасности и установлении возможных источников их возникновения. Проектирование конкретной системы безопасности для любого объекта, в том числе и для систем баз данных, предполагает выявление и научную классификацию перечня источников угроз безопасности.

Сформулируем перечень внешних и внутренних угроз информационной безопасности баз данных.

Внешними дестабилизирующими факторами, создающими угрозы безопасности функционированию систем баз данных и СУБД, являются:

умышленные, деструктивные действия лиц с целью искажения,

уничтожения или хищения программ, данных и документов системы, причиной которых являются нарушения информационной безопасности защищаемого объекта;

искажения в каналах передачи информации, поступающей от внешних источников, циркулирующих в системе и передаваемой потребителям, а также недопустимые значения и изменения характеристик потоков информации из внешней среды и внутри системы;

сбои и отказы в аппаратуре вычислительных средств;

вирусы и иные деструктивные программные элементы, распространяемые с использованием систем телекоммуникаций, обеспечивающих связь с внешней средой или внутренние коммуникации распределенной системы баз данных;

изменения состава и конфигурации комплекса взаимодействующей аппаратуры системы за пределы, проверенные при тестировании или сертификации системы.

Внутренними источниками угроз безопасности баз данных и СУБД являются:

системные ошибки при постановке целей и задач проектирования автоматизированных информационных систем и их компонент, допущенные при формулировке требований к функциям и характеристикам средств обеспечения безопасности системы;

ошибки при определении условий и параметров функционирования внешней среды, в которой предстоит использовать информационную систему и, в частности, программно-аппаратные средства защиты данных;

ошибки проектирования при разработке и реализации алгоритмов обеспечения безопасности аппаратуры, программных средств и баз данных;

ошибки и несанкционированные действия пользователей, административного и обслуживающего персонала в процессе эксплуатации системы;

недостаточная эффективность используемых методов и средств обеспечения информационной безопасности в штатных или особых условиях эксплуатации системы.

Полное устранение всех потенциальных угроз информационной безопасности баз данных принципиально невозможно. Реальная задача состоит в снижении вероятности реализации потенциальных угроз до приемлемого для конкретной системы уровня. Приемлемость соответствующего уровня угроз может определяться областью применения, выделенным бюджетом или положениями действующего законодательства. Как правило, не удастся построить дерево угроз со строгой иерархией. Поэтому совокупный риск является достаточно сложной функцией уязвимости компонентов системы. Различные негативные воздействия также достаточно сложным образом влияют на основные характеристики качества и безопасности систем баз данных.

Основным руководящим принципом создания систем защиты является принцип равнопрочности. Следует распределять доступные ресурсы,

обеспечивающие информационную безопасность системы, таким образом, чтобы минимизировать некоторый обобщенный показатель риска при любых негативных внешних и внутренних воздействиях на систему. Наличие в системе угроз, для защиты от которых в системе не предусмотрено каких-либо мер противодействия, приводит к тому, что все усилия, затраченные на возведение эффективных барьеров для иных способов деструктивного воздействия на систему, к ожидаемому результату не приведут. Отсюда следует важный практический вывод: учет угроз должен быть всесторонний и для каждой из возможных угроз должен быть реализован соответствующий угрозе метод защиты.

Угрозы конфиденциальности информации

К угрозам такого типа можно отнести:

1. *Инъекция SQL.* Во многих приложениях используется динамический SQL - формирование SQL-предложений кодом программы путем конкатенации строк и значений параметров. Зная структуру базы данных, злоумышленник может либо выполнить хранимую программу в запросе, либо закомментировать «легальные» фрагменты SQL-кода, внедрив, например, конструкцию UNION, запрос которой возвращает конфиденциальные данные. В последнее время даже появились специальные программы, автоматизирующие процесс реализации подобных угроз.

2. *Логический вывод на основе функциональных зависимостей.* Пусть дана схема отношения: $R(A_1, \dots, A_n)$. Пусть $i = \{A_1, \dots, A_n\}$, X, Y - подмножества из U . Говорят, что X функционально определяет Y , если в любом отношении r со схемой $R(A_1, \dots, A_n)$ не могут содержаться два кортежа с одинаковыми значениями атрибутов из X и с различными из Y . В этом случае имеет место функциональная зависимость, обозначаемая $X \rightarrow Y$. Пример функциональной зависимости для схемы отношения (фамилия, имя, отчество, должность, зарплата): если должность = менеджер, то зарплата = 1200. В реальных базах данных при наличии сведений о функциональных зависимостях злоумышленник может вывести конфиденциальную информацию при наличии доступа только к части отношений, составляющих декомпозированное отношение.

3. *Логический вывод на основе ограничений целостности.* Для кортежей отношений в реляционной модели данных (РМД) можно задать ограничения целостности - логические условия, которым должны удовлетворять атрибуты кортежей.

4. *Использование оператора UPDATE для получения конфиденциальной информации.* В некоторых стандартах SQL пользователь, не обладая привилегией на выполнение оператора SELECT, мог выполнить оператор UPDATE со сколь угодно сложным логическим условием. Так как после выполнения оператора UPDATE сообщается, сколько строк он обработал, фактически пользователь мог узнать, существуют ли данные, удовлетворяющие этому условию.

Специфичными для систем управления базами данных угрозами доступности являются:

1. Использование свойств первичных и внешних ключей. В первую очередь сюда относится свойство уникальности первичных ключей и наличие ссылочной целостности. В том случае, если используются натуральные, а не генерируемые системой значения первичных ключей, можно создать такую ситуацию, когда в таблицу невозможно будет вставить новые записи, так как там уже будут записи с такими же значениями первичных ключей. Если в базе данных поддерживается ссылочная целостность, можно организовать невозможность удаления родительских записей, умышленно создав подчиненные записи. Важной особенностью реализации ссылочной целостности является вопрос об индексировании внешнего ключа.

2. Блокировка записей при изменении. Заблокировав записи или всю таблицу, злоумышленник может на значительное время сделать ее недоступной для обновления.

3. Загрузка системы бессмысленной работой. Простейший пример - выполнение запроса, содержащего декартово произведение двух больших отношений. В реляционных СУБД возможны реализации и других классических угроз, например, атаки типа «тройанский конь» - запуска пользователями программ, содержащих выполняющий определенные действия код, внедренный туда злоумышленником.

Практически все современные СУБД имеют встроенный процедурный язык программирования (PL/SQL, Transact SQL и т.п.) Программы на этом языке хранятся внутри базы данных и выполняются исполняющей подсистемой сервера баз данных, наличие и широкое использование производителями прикладного программного обеспечения механизма скрытия исходных текстов хранимых программ затрудняют его обнаружение. Также возможны многочисленные скрытые каналы, обусловленные семантикой данных и необходимостью обеспечения работы в условиях многопользовательского доступа.

На основании проведенного анализа можно сделать вывод, что в современных реализациях реляционных СУБД имеется значительное число уязвимостей, которые могут быть использованы для различных атак на информационные системы, построенные на их базе. Эта проблема отчасти может быть решена использованием специализированных программных средств анализа уязвимостей и защиты от угроз различных типов.

Но нельзя не заметить, что проблеме обеспечения информационной безопасности баз данных присуща определенная специфика. Источником дополнительных, специфических для баз данных, угроз являются:

- стандарт и реализация языка SQL - основного средства описания и обработки данных;
- обязательный интерфейс между СУБД и операционной системой, под управлением которой ведется обработка данных;
- протоколы сетевого взаимодействия на прикладном уровне, которые обеспечиваются средствами СУБД.

Любая система обработки данных, допускающая

многопользовательский режим работы, предусматривает идентификацию и аутентификацию пользователей и процессов как обязательное предусловие предоставления доступа к ресурсам системы. Широко распространенной технологией аутентификации является использование паролей. Естественно ожидать, что технология подбора паролей также находится на достаточно высоком уровне развития. Можно выделить следующие *методы подбора паролей пользователей*.

1. Тотальный перебор. В этом случае злоумышленник последовательно опробует все возможные варианты пароля. Для паролей длиннее шести символов во многих случаях данный метод может быть признан неэффективным.

2. Тотальный перебор, оптимизированный по статистике встречаемости символов. Разные символы встречаются в паролях пользователей с разной вероятностью. Например, вероятность того, что в пароле пользователя встретится буква «а», гораздо выше вероятности того, что в пароле присутствует символ «л». Согласно различным исследованиям, статистика встречаемости символов в алфавите паролей близка к статистике встречаемости символов в естественном языке.

При практическом применении данного метода злоумышленник вначале опробует пароли, состоящие из наиболее часто встречающихся символов, за счет чего время перебора существенно сокращается. Иногда при подборе паролей используется не только статистика встречаемости символов, но и статистика встречаемости диграмм и триграмм - комбинаций двух и трех последовательных символов соответственно.

Тотальный перебор, оптимизированный с помощью словарей

В большинстве случаев пароли пользователей представляют собой слова английского или русского языка. Поскольку пользователю гораздо легче запомнить осмысленное слово, чем бессмысленную последовательность символов, пользователи предпочитают применять в качестве паролей осмысленные слова. При этом количество возможных вариантов пароля резко сокращается. Словарь С. И. Ожегова содержит около 60000 слов, объем словаря среднестатистического пользователя заметно меньше.

При использовании данного метода подбора паролей злоумышленник вначале опробует в качестве паролей все слова из словаря, содержащего наиболее вероятные пароли. В сети Интернет представлено множество подобных словарей, адаптированных для различных стран и групп пользователей. Если подбираемый пароль отсутствует в словаре, злоумышленник, как правило, может опробовать всевозможные комбинации слов из словаря, слова из словаря с добавленными к началу или к концу одной или несколькими буквами, цифрами и знаками препинания и т. д. «Хитрости» с написанием слов в обратном порядке в иной раскладке клавиатуры также не являются чем-то новым. Обычно данный метод используется в комбинации с предыдущим.

Человек склонен использовать пароли, которые легко запоминаются.

Многие пользователи, чтобы не забыть пароль, выбирают в качестве пароля свое имя, фамилию, дату рождения, имена детей, любимых (в том числе принятые в узком кругу), номера телефонов и автомобилей и т. д. Хорошо известно, что некоторая категория мужчин склонна использовать в качестве (не отображаемого на дисплей) пароля ненормативную лексику, а определенная категория женщин часто использует уменьшительно-ласкательные имена домашних животных. В этом случае, если злоумышленник хорошо изучил пользователя, ему, как правило, достаточно провести меньше сотни опробований.

Базовые процедуры аутентификации на основе паролей в большинстве современных СУБД построены на сравнении вычисляемой значения хэш-функции от вводимого пароля с хранимым образом пароля, приписанного пользователю. Для вычисления образов паролей используются стандартные хэш-функции, поэтому их криптографические свойства не вызывают сомнения. Количество опробований (неуспешных попыток ввода) пароля устанавливается в профиле пользователя.

Наиболее слабым местом базовой процедуры аутентификации, видимо, является возможность доступа к образу пароля привилегированного пользователя. Если не используется процедура аутентификации средствами операционной системы, то образ пароля хранится в справочнике данных. В этом случае привилегированный пользователь может не только считать образ для организации подбора пароля, но и незаметно для пользователя временно подменить пароль и выполнить некоторые действия от чужого имени.

Нецелевое расходование вычислительных ресурсов сервера

Выполнение запросов и иные обращения к серверу баз данных связаны с расходованием его ресурсов, в первую очередь времени процессора, оперативной и внешней памяти. Это очевидное утверждение имеет одно неочевидное следствие. Если пользователю предоставлена возможность создавать процедуры и их выполнять, то может возникнуть проблема нарушения доступности в связи с нецелевым расходованием ресурсов.

Ясно, что запуск процедуры приведет к «замиранию всего живого» в системе. Эффект от подобной атаки может быть уменьшен назначением пользователю профиля, ограничивающего максимальное выделяемое время центрального процессора.

Напомним, что система контроля ограничений на ресурсы должна быть приведена в активное состояние.

Бороться с процедурами, расходующими ресурсы системы нецелевым образом, да и с иными деструктивными процедурами офицер безопасности может и принудительным завершением процессов-вредителей.

Использование SQL-инъекции для нештатного использования процедур и функций

Использование процедур и функций для параметрических запросов является широко распространенной практикой создания элементов информационных систем. Во многих случаях запрос к базе данных не зависит от параметров. Такой запрос называется статическим. При

использовании статического запроса программист создает PL/SQL программу, включая в код необходимые SQL-предложения. В зависимости от содержащихся в базе данных формируется некоторый результат, например, отчет о расходах или поступлениях товаров на склад на плановый период. В этом случае в тексте программы все SQL-предложения, которые необходимы в процессе ее работы, явно описываются, компилируются и могут храниться в вызываемых процедурах или функциях.

Но возможны и иные варианты. Например, значение параметра запроса (фамилия, адрес) зависит от ситуации. Для решения таких задач необходим механизм динамически формируемых SQL-предложений. Самое простое решение - предоставить пользователю оболочку для формирования запросов на языке SQL. Для реализации такого варианта пользователи должны владеть языком SQL. Это дорогое решение, к тому же создающее серьезные проблемы с безопасностью.

Чаще используется другой вариант: создаются программы на PL/SQL или ином языке с параметрами, которые отражают специфику запроса.

В комплект поставки сервера Oracle входит стандартный пакет DBMSSQL, обеспечивающий выполнение динамически формируемых SQL-предложений в программах на PL/SQL. Динамические SQL-предложения конструируются непосредственно во время выполнения программы в виде символьной строки, а затем передаются соответствующим программам пакета DBMS SQL для разбора и исполнения.

Ответственность за возможное нарушение объектных зависимостей и прав доступа, которые Oracle в данном случае во время компиляции не может проверить, теперь ложится на программиста.

Для динамически сформированных операторов SQL, которые возвращают данные (т. е. для операторов SELECT), используется следующая последовательность обработки: открытие курсора, разбор запроса, связывание переменных, определение столбцов, выполнение запроса, выборка строк, получение значений столбцов, закрытие курсора.

Конечно, более изящно решение выполнения параметрического запроса с использованием динамического SQL. Но и в этом случае, несмотря на кажущуюся жесткость управления действиями пользователя, возможно деструктивное действие, известное как SQL-инъекция.

Оборудование и материалы

1. Компьютер под управлением операционной системы Windows 8/10 с параметрами по умолчанию.
2. Установленные на компьютер программы Microsoft Visio, PostgreSQL, MySQL, VMWare/VirtualBox.
3. Текстовый процессор Microsoft Word 2010/2013/2016.

Указания по технике безопасности

К выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж по технике безопасности.

Задания

1. Определить все возможные риски.
2. Определить вероятность возникновения риска и последствия.
3. Определить связанные риски.
4. Рассчитать влияние.
5. Выделить существенные риски.
6. Разработать план снижения рисков.

Для выполнения задач № 1 - 4 удобно использовать нижеследующую таблицу.

№	Идент и-	Источ	Услов	Вероят	Послед	Влияние	Связан
		ник	ия	ность	ствия		ные

Заполнение полей осуществляется по следующим рекомендациям:

Идентификатор - уникальное название, определяющее риск.

Источник - объект или субъект, порождающий или реализующий данный риск. Источником, может быть, как пользователь (проектировщик, программист, администратор и т. д.), так и программное обеспечение (ОС, СУБД, прикладное программное обеспечение и т. д.).

Условия возникновения - описание условий, при которых риск может быть реализован.

Вероятность - оценка вероятности возникновения конкретного риска при определенных условиях.

Последствия - оценка последствий реализации риска. Может быть выражена числом по 5-ти или 10-ти балльной шкале. Показывает относительную важность проблемы. Каждое значение шкалы должно быть четко определено и расписано.

Влияние - сводная количественная оценка риска для проекта, равная произведению его вероятности на количественную оценку его последствий.

Связанные риски — перечень рисков, связанных с данным.

Задача 5 - осуществляется при ранжировании рисков по влиянию. Для определения существенных рисков необходимо определить нижнюю границу, значение, которое будет минимальным при определении существенности риска.

Задача 6 - Разработка плана снижения рисков, включает в себя определение мероприятий, которые должны снизить значение риска до нижней границы влияния, либо нивелировать риск совсем. Данный план должен включать в себя обоснования данных действий и мероприятий, а также расчеты, обосновывающие их применение.

Содержание отчета

Отчет о выполнении лабораторной работы должен содержать:

- название и тему лабораторной работы;

- цель лабораторной работы;
- краткие теоретические сведения;
- ход выполнения работы;
- выводы.

Раздел «Ход выполнения работы» должен содержать инфологическую модель БД на языке таблица-связь, описание разработанных ограничений целостности, рисунки с отчетами Data Dictionary Report для каждой таблицы БД.

Контрольные вопросы

1. Какие операторы использует язык SQL, для назначения отмены привилегий?
2. Что понимают под угрозой информационной безопасности автоматизированной информационной системе (АИС)?
3. Перечислите основные источники угроз информации баз данных.
4. Перечислите угрозы, специфичные для систем управления базами данных.
5. Перечислите методы подбора паролей пользователей.
6. Приведите пример использования SQL-инъекции для нештатного использования процедур и функций.

Лабораторная работа №2. Разграничение доступа к базе данных. 4 часа

Цель работы: получение навыков по проектированию и реализации разграничения прав доступа к базе данных.

Теоретическая часть

В современных условиях любая деятельность сопряжена с оперированием большими объемами информации, которое производится широким кругом лиц. Защита данных от несанкционированного доступа является одной из приоритетных задач при проектировании любой информационной системы.

Для СУБД важны три основных аспекта информационной безопасности - конфиденциальность, целостность и доступность. Общая идея защиты базы данных состоит в следовании рекомендациям, сформулированным для класса безопасности С2 в «Критериях оценки надежных компьютерных систем».

Конфиденциальная информация (sensitive information) - информация, которая требует защиты.

Доступ к информации (access to information) - ознакомление с информацией, ее обработка (в частности, копирование), модификация, уничтожение.

Субъект доступа (access subject) - лицо или процесс, действия

которого регламентируются правилами разграничения доступа.

Объект доступа (access object) - единица информации автоматизированной системы, доступ к которой регламентируется правилами разграничения доступа. Объектами доступа (контроля) в СУБД является практически все, что содержит конечную информацию: таблицы (базовые или виртуальные), представления, а также более мелкие элементы данных: столбцы и строки таблиц и даже поля строк (значения). Таблицы базы данных и представления имеют владельца или создателя. Их объединяет еще и то, что все они для конечного пользователя представляются как таблицы, то есть как нечто именованное, содержащее информацию в виде множества строк (записей) одинаковой структуры. Строки таблиц разбиты на поля именованными столбцами.

Правила разграничения доступа (security policy) - совокупность правил, регламентирующих права субъектов доступа к объектам доступа.

Санкционированный доступ (authorized access to information) - доступ к информации, который не нарушает правил разграничения доступа.

Несанкционированный доступ (unauthorized access to information) - доступ к информации,

который нарушает правила разграничения доступа с использованием штатных средств, предоставляемых средствами вычислительной техники или автоматизированными системами.

Идентификатор доступа (access identifier) - уникальный признак объекта или субъекта доступа.

Идентификация (identification) - присвоение объектам и субъектам доступа идентификатора и (или) сравнение предъявляемого идентификатора с перечнем присвоенных идентификаторов.

Пароль (password) - идентификатор субъекта, который является его секретом.

Аутентификация (authentication) - проверка принадлежности субъекту доступа предъявленного им идентификатора, подтверждение подлинности.

В СУБД на этапе подключения к БД производится идентификация и проверка подлинности пользователей. В дальнейшем пользователь или процесс получает доступ к данным согласно его набору полномочий. В случае разрыва соединения пользователя с базой данных текущая транзакция откатывается, и при восстановлении соединения требуется повторная идентификация пользователя и проверка его полномочий.

Уровень полномочий субъекта доступа (subject privilege) - совокупность прав доступа субъекта доступа (для краткости в дальнейшем мы будем использовать термин «привилегия»). *Нарушитель правил разграничения доступа (security policy violator)* - субъект доступа, который осуществляет несанкционированный доступ к информации.

Модель нарушителя правил разграничения доступа (security policy violator model) - абстрактное (формализованное или неформализованное) описание нарушителя правил разграничения доступа.

Целостность информации (information integrity) - способность средства вычислительной техники (в рассматриваемом случае - информационной системы в целом) обеспечить неизменность информации в условиях случайного и (или) преднамеренного искажения (разрушения).

Метка конфиденциальности (sensitivity label) - элемент информации, характеризующий конфиденциальность объекта.

Многоуровневая защита (multilevel secure) - защита, обеспечивающая разграничение доступа субъектов с различными правами доступа к объектам различных уровней конфиденциальности.

Дискреционная защита

В современных СУБД достаточно развиты средства дискреционной защиты.

Дискреционное управление доступом (discretionary access control) - разграничение доступа между поименованными субъектами и поименованными объектами. Субъект с определенным правом доступа может передать это право любому другому субъекту.

Однако дискреционная защита является довольно слабой, так как доступ ограничивается только к именованным объектам, а не собственно к хранящимся данным. В случае реализации информационной системы с использованием реляционной СУБД объектом будет, например, именованное отношение (то есть таблица), а субъектом - зарегистрированный пользователь. В этом случае нельзя в полном объеме ограничить доступ только к части информации, хранящейся в таблице. Частично проблему ограничения доступа к информации решают представления и использование хранимых процедур, которые реализуют тот или иной набор бизнес-действий.

Мандатная защита

Средства мандатной защиты предоставляются специальными (trusted) версиями СУБД.

Мандатное управление доступом (mandatory access control) - это разграничение доступа субъектов к объектам данных, основанное на характеризующей метке конфиденциальности информации, которая содержится в объектах, и на официальном разрешении (допуске) субъектов обращаться к информации такого уровня конфиденциальности.

Средства произвольного управления доступом характерны для уровня безопасности С. Как правило, их, в принципе, вполне достаточно для подавляющего большинства коммерческих приложений. Тем не менее они не решают одной весьма важной задачи - задачи слежения за передачей информации. Средства произвольного управления доступом не могут помешать авторизованному пользователю законным образом получить секретную информацию и затем сделать ее доступной для других, неавторизованных, пользователей. Нетрудно понять, почему это так. При произвольном управлении доступом привилегии существуют отдельно от данных (в случае реляционных СУБД - отдельно от строк реляционных таблиц), в результате чего данные оказываются «обезличенными» и ничто не

мешает передать их кому угодно даже средствами самой СУБД; для этого нужно лишь получить доступ к таблице или представлению.

Использование СУБД с возможностями мандатной защиты позволяет разграничить доступ, собственно, к данным, хранящимся в информационной системе, от доступа к именованным объектам данных. Единицей защиты в этом случае будет являться, в частности, запись о договоре N, а не таблица или представление, содержащее информацию об этом договоре. Пользователь, который будет пытаться получить доступ к договору, уже никак не сможет обойти метку конфиденциальности. Существуют реализации, позволяющие разграничивать доступ вплоть до конкретного значения конкретного атрибута в конкретной строке конкретной таблицы. Дело не ограничивается одним значением метки конфиденциальности - обычно сама метка представляет собой набор значений, отражающих, например, уровень защищенности устройства, на котором хранится таблица, уровень защищенности самой таблицы, уровень защищенности атрибута и уровень защищенности конкретного кортежа.

Оборудование и материалы

1. Компьютер под управлением операционной системы Windows 8/10 с параметрами по умолчанию.
2. Установленные на компьютер программы Microsoft Visio, PostgreSQL, MySQL, VMWare/VirtualBox.
3. Текстовый процессор Microsoft Word 2010/2013/2016.

Указания по технике безопасности

К выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж по технике безопасности.

Задания

1. Реализовать пример разграничения доступа на практическом макете.
2. Для своего индивидуального задания по курсовой работе спроектировать и реализовать разграничение доступа в виде ролевой или дискреционной моделей.

Описание предметной области

Проектируемая база данных должна содержать в себе *информацию*'.

- контактная информация заказчика;
- контактная информация поставщика;
- дата и время доставки товара после получения заказа.

Основные пользователи базы данных:

- администратор базы данных;
- администраторы по заполнению информации;
- заказчики.

Функции пользователей базы данных:

- администратор базы данных - поддержание работоспособности базы данных;
- администраторы по заполнению информации - добавление, изменение, удаления данных, просмотр всей информации;
- заказчик - только просмотр данных. Перечень вводимой информации:
 - имя;
 - место проживания;
 - номер телефона.

База данных должна поддерживать накопление и хранение информации об основных компонентах бизнеса и автоматизированное выполнение бизнес-процессов.

Бизнес-правила:

- Вносить изменения может только Администратор по заполнению информации.
- Иные зарегистрированные пользователи базы данных могут просматривать данные о заказах.

Для реализации базы данных необходимо спроектировать концептуальную модель и реализовать ее в выбранной СУБД. Процесс реализации уже готовой логической модели представлен на рисунках 2.1 - 2.3.



Рис. 2.1. Рабочая среда

WIN-BB6L39QWY...bo. Покупатели		WIN-BB6L39QWY...bo. Поставщики		
	id	Name	Adress	Phone
▶	1	Jane	New-York, USA	7777777
	2	Michal	Washington, USA	4444444
	3	Roma	Tomsk, Russia	777
	4	Vasya	Moskow, Russia	134124
	5	Elena	Surgut city, Russia	89222222222
#	NULL	NULL	NULL	NULL

Рис. 2.2. Таблица «Покупатели»

Server» и создаем имя входа (рисунок 2.6).

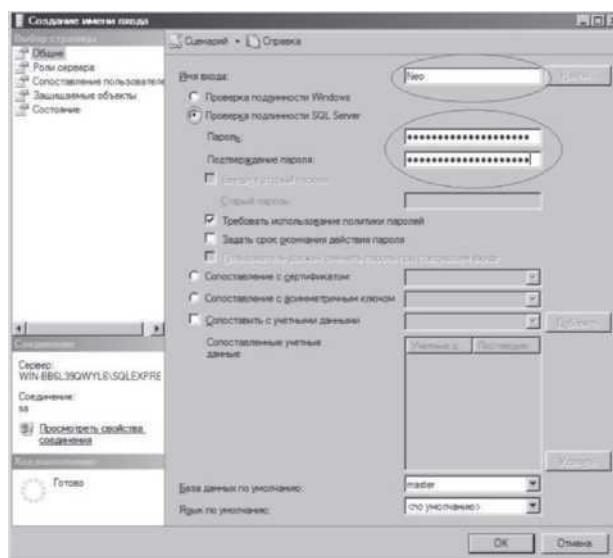


Рис. 2.6. Создание имени

После создания имени входа, присваиваем его конкретному пользователю (рисунки 2.7 - 2.10).

В поле «Имя пользователя» записываем то имя, которое будет распознаваться в БД, и присваиваем Имя Входа (рисунок 2.7).

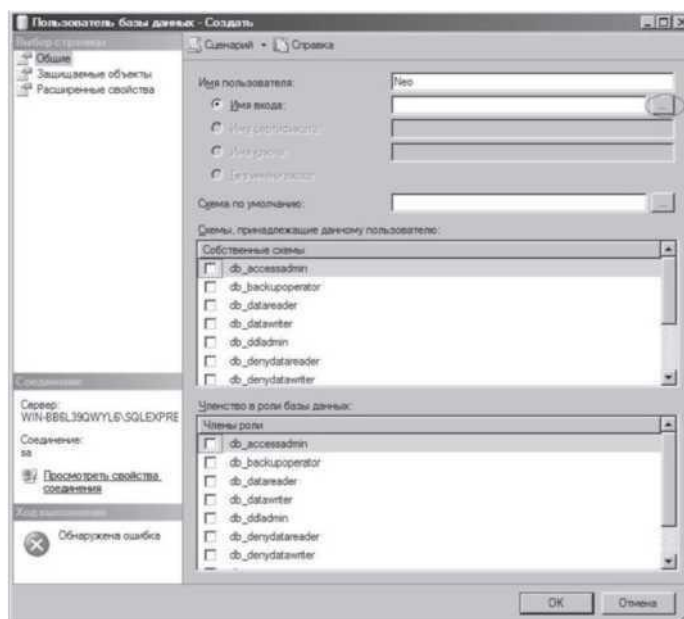


Рис. 2.7. Создание «Имени Пользователя»

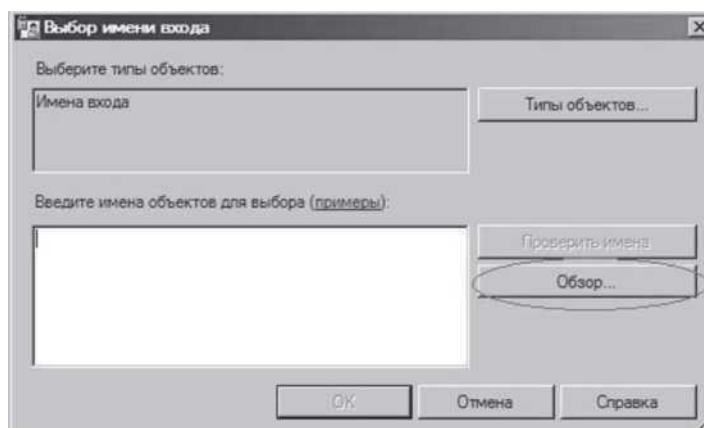


Рис. 2.8. Поиск «имени входа 1»

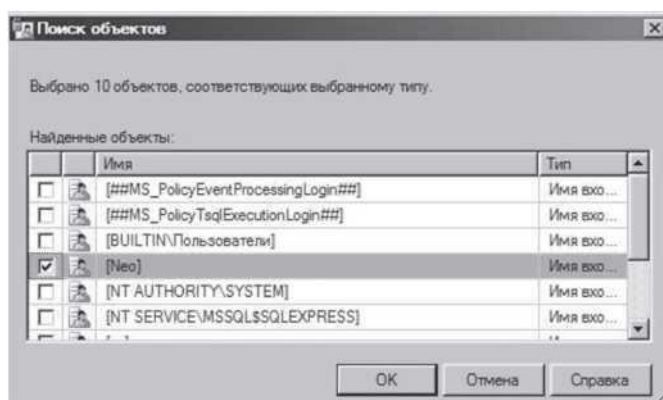


Рис. 2.9. Поиск «имени входа 2»

Задаем Схему базы данных и Роль для данного пользователя (рисунок 2.10).

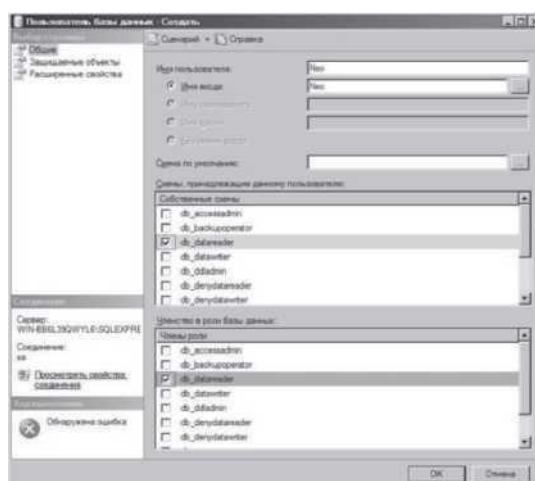


Рис. 2.10. Присвоения членства

Данные действия позволили пользователю присвоить роль «db_datareader», позволяющей пользователю читать все данные представленные в БД без права внесения каких-либо изменений в нее.

Для подключения к БД под соответствующим Пользователем необходимо выбрать в меню

«Файл» вкладку «Подключится к обозревателю объектов» (рисунок 2.11).

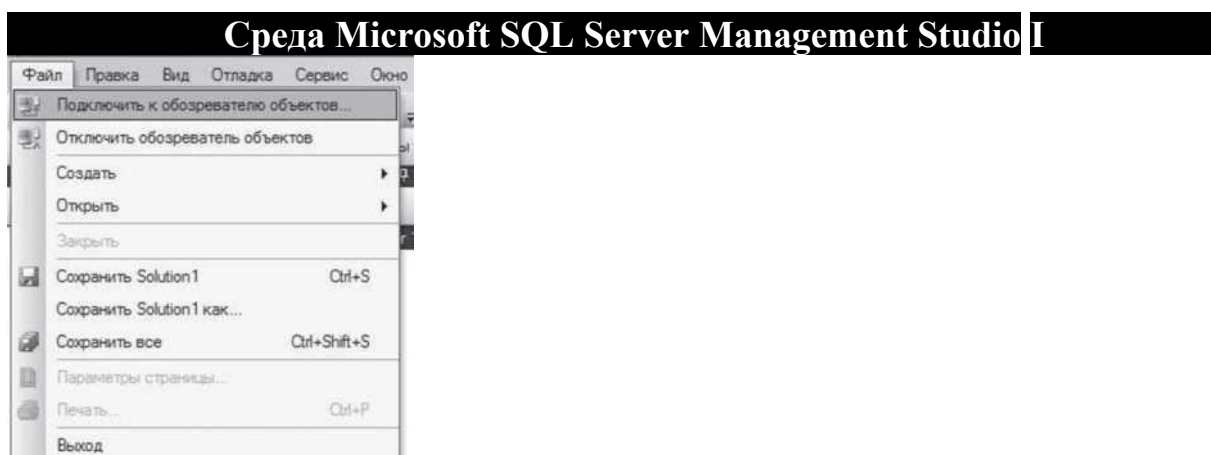


Рис. 2.11. Подключение к БД

И выбирать в пункте «Проверка подлинности» «Проверка подлинности SQL Server'a» (рисунок 2.12).



Рис. 2.12. Подключение к БД

Для проверки прав доступа, которые были присвоены Новому пользователю БД можно проделать, следующие действия.

Текущий пользователь не является владельцем базы данных J или системным администратором. Пользователь не может сохранять изменения в таблицах, владельцем которых он не является

Данные результаты демонстрируют правильность работы приложения по разграничению прав доступа.

Содержание отчета

Отчет о выполнении лабораторной работы должен содержать:

- название и тему лабораторной работы;
- цель лабораторной работы;
- краткие теоретические сведения;
- ход выполнения работы;
- выводы.

Раздел «Ход выполнения работы» должен содержать инфологическую

модель БД на языке таблица-связь, описание разработанных ограничений целостности, рисунки с отчетами Data Dictionary Report для каждой таблицы БД.

Контрольные вопросы

1. Что такое относительная защита информации в БД?
2. Дайте определение групп пользователей СУБД;
3. Что такое дискреционная защита?
4. Что такое данные о разграничениях доступа при дискреционной защите?
5. Что является объектом и субъектом защиты при настройке привилегий дискреционной защиты?
6. Что такое представление и какие ограничения по настройке безопасности имеются при работе средств SQL?
7. Что такое мандатная защита?
8. Почему при произвольном управлении доступом данные оказываются «обезличенными»?
9. Что такое логическая защита?
10. Что такое группы принадлежности?
11. Что включает в себя механизм меток безопасности;
12. Дайте определения каждого уровня доступа при реализации мандатной защиты?
13. От кого наследуют уровни пользователя при внесении данных?

Лабораторная работа №3. Использование и защита транзакций в базах данных. 4 часа

Цель работы: изучить транзакцию и осуществить ее в MySQL.

Теоретическая часть

Транзакция - действие или ряд действий, выполняемых одним пользователем или прикладной программой, которые осуществляют чтение или изменение содержимого базы данных. Транзакция является логической единицей работы, выполняемой в базе данных. Она может быть представлена отдельной программой, частью программы или даже отдельной командой (например, командой INSERT или UPDATE языка SQL) и включать произвольное количество операций, выполняемых в базе данных.

Фиксированная транзакция - транзакция выполнение, в которой выполнение всех действий успешно завершилось.

Любая транзакция завершается одним из двух возможных способов. В случае успешного завершения результаты транзакции фиксируются (commit) в базе данных, и последняя переходит в новое состояние. Если окажется, что зафиксированная транзакция была ошибочной, потребуется выполнить другую транзакцию, отменяющую действия, выполненные первой

транзакцией.

Ни в одной СУБД не может быть предусмотрен априорный способ определения того, какие именно операции обновления могут быть сгруппированы для формирования единой логической транзакции. Поэтому должен применяться метод, позволяющий указывать границы каждой из транзакций извне, со стороны пользователя. В большинстве языков манипулирования данными для указания границ отдельных транзакций используются операторы BEGIN TRANSACTION, COMMIT и ROLLBACK. В случае аварийного завершения программы в базе данных автоматически будет выполнена команда ROLLBACK.

Источники отказов:

1) Системные ошибки. Система вошла в нежелательное состояние. Такое как взаимоблокировка, не позволяющая нормально производить выполнение программы. Приводит к повреждению данных.

2) Отказ оборудования. Ошибка диска и потеря способности передачи данных через соединительные линии.

3) Логические ошибки. Плохие данные или их отсутствие.

Существуют определенные свойства, которыми должна обладать любая из транзакций. Ниже представлены четыре основных свойства транзакций, которые принято обозначать аббревиатурой ACID (Atomicity, Consistency, Isolation, Durability - неразрывность, согласованность, изолированность, устойчивость), состоящей из первых букв названий этих свойств.

Неразрывность. Это свойство, для описания которого применимо выражение «все или ничего». Любая транзакция представляет собой неделимую единицу работы, которая может быть либо выполнена вся целиком, либо не выполнена вообще. За обеспечение неразрывности отвечает подсистема восстановления СУБД.

Согласованность. Каждая транзакция должна переводить базу данных из одного согласованного состояния в другое. Ответственность за обеспечение согласованности возлагается и на СУБД, и на разработчиков приложений. В СУБД согласованность может обеспечиваться путем выполнения всех ограничений, заданных в схеме базы данных, таких как ограничения целостности и ограничения предметной области. Но подобное условие является недостаточным для обеспечения согласованности. Например, предположим, что некоторая транзакция предназначена для перевода денежных средств с одного банковского счета на другой, но программист допустил ошибку в логике транзакции и предусмотрел снятие денег с правильного счета, а зачисление - на неправильный счет. В этом случае база данных переходит в несогласованное состояние, несмотря на наличие правильно заданных ограничений. Тем не менее ответственность за устранение такой несогласованности не может быть возложено на СУБД, поскольку в ней отсутствуют какие-либо средства обнаружения подобных логических ошибок.

Изолированность. Все транзакции выполняются независимо друг от друга. Иными словами, промежуточные результаты незавершенной

транзакции не должны быть доступны для других транзакций. За обеспечение изолированности отвечает подсистема управления параллельным выполнением.

Устойчивость. Результаты успешно завершенной (зафиксированной) транзакции должны храниться в базе данных постоянно и не должны быть потеряны в результате последующих сбоев. За обеспечение устойчивости отвечает подсистема восстановления.

Модели транзакций. Плоские транзакции - это основные строительные блоки для реализации принципа атомарности; иначе говоря, выделение некоторой последовательности действий в виде плоской транзакции обеспечивает принцип «все или ничего». Во многих прикладных окружениях, в особенности с централизованными обработкой и управлением ресурсами (например, базами данных и файлами), механизм плоских транзакций на протяжении многих лет предоставлял вполне удовлетворительные возможности как для создания, так и для выполнения приложений; простые преобразования состояний системы вполне укладывались в рамки атомарных единиц работы.

Модель вложенных транзакций. Транзакция рассматривается как коллекция взаимосвязанных подзадач или субтранзакций, каждая из которых также может состоять из любого количества субтранзакций. Модель вложенных транзакций была предложена Моссом (Moss) в 1981 году. В этой модели вся транзакция рассматривается как набор связанных подзадач, называемых субтранзакциями, каждая из которых также может состоять из произвольного количества субтранзакций. В данном определении полная транзакция представляет собой древовидную структуру или некоторую иерархию субтранзакций. В модели вложенных транзакций присутствует транзакция верхнего уровня, содержащая некоторое количество дочерних транзакций, каждая из которых, в свою очередь, может включать вложенные транзакции, и т.д. В исходном варианте, предложенном Моссом, выполнять операции в базе данных разрешается только транзакциям самого нижнего уровня (субтранзакциям самого нижнего уровня вложенности).

Хроника. Последовательность (плоских) транзакций, которая может передаваться с другими транзакциями СУБД гарантирует, что-либо все входящие в хронику транзакции будут успешно завершены, либо будут запущены компенсирующие транзакции, необходимые для устранения достигнутых частичных результатов. В отличие от метода вложенных транзакций, допускающего произвольный уровень вложения, метод хроник разрешает наличие единственного уровня вложения. Более того, для каждой выделенной субтранзакции должна существовать соответствующая компенсирующая транзакция, которая будет семантически правильно аннулировать результаты, достигаемые с помощью данной субтранзакции.

Модель вложенных транзакций требует, чтобы процесс фиксации субтранзакций выполнялся снизу-вверх, в направлении транзакции верхнего уровня. Поэтому данную модель принято называть моделью закрытых вложенных транзакций. Это позволяет подчеркнуть, что свойство

неразрывности в транзакциях сохраняется до самого верхнего уровня. В противоположность этому, существует и модель открытых вложенных транзакций, в которой неразрывность нарушается и частичные результаты выполнения субтранзакций могут быть доступны вне транзакции.

Модели рабочих потоков. Все обсуждавшиеся в этом разделе модели были разработаны с целью преодоления ограничений, накладываемых моделью плоских транзакций, поэтому не приемлемых для транзакций большой продолжительности. Однако было отмечено, что все эти модели по-прежнему не обладают мощностью, необходимой для удовлетворения потребностей деловых процессов определенных видов. Поэтому были предложены более сложные модели, представляющие собой комбинации моделей открытых и вложенных транзакций. Однако, поскольку эти модели не в полной мере отвечают требованиям ACID плоских транзакций, для них используется более подходящее название - модели рабочих потоков.

Рабочий поток представляет собой некоторый вид деятельности, предусматривающий координированное выполнение множества заданий, осуществляемых различными обрабатывающими субъектами, которые могут представлять собой людей или некоторые программные комплексы (например, СУБД, прикладные программы или службы электронной почты). Рассмотрим пример с подготовкой соглашений о сдаче объектов недвижимости в аренду, который можно найти в учебном проекте DreamHome. Клиент, желающий арендовать некоторый объект недвижимости, устанавливает контакт с соответствующим сотрудником компании, отвечающим за этот объект недвижимости. Этот сотрудник обращается к контролеру компании, в обязанности которого входит проверка кредитоспособности клиента и определение того, может ли компания ему доверять. С этой целью контролер использует соответствующие источники информации, например, бюро проверки кредитоспособности. Собрав интересующие его сведения, контролер принимает решение о возможности дальнейшей работы с данным клиентом и сообщает о своем решении сотруднику, который сообщает клиенту о принятом решении.

Классификация систем обработки транзакций

С появлением множества стандартизированных систем обработки транзакций нового поколения полезным представляется проведение их классификации с точки зрения спектра предоставляемых ими функций. Подобную классификацию, включающую пять измерений, предложили Абрахам Лефф и Калтон Пу из Колумбийского университета:

М - множество машин; Р - множество процессов; Н - степень неоднородности машин и программного обеспечения; D - множество логических данных; S - множество узлов.

Эта классификация характеризует любую систему обработки транзакций, от простейших (P1, M1, H1, D1, S1) до более сложных многоузловых неоднородных окружений с поддержкой разнотипных наборов данных (Pn, Mn, Hn, Dn, Sn). В статье, написанной в 1991 г., эти авторы представили трехмерную классификацию, которую они применили для

оценки различных исследовательских и коммерческих систем.

Оборудование и материалы

1. Компьютер под управлением операционной системы Windows 8/10 с параметрами по умолчанию.
2. Установленные на компьютер программы Microsoft Visio, PostgreSQL, MySQL, VMWare/VirtualBox.
3. Текстовый процессор Microsoft Word 2010/2013/2016.

Указания по технике безопасности

К выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж по технике безопасности.

Задания

1. Перечисление денежных средств с одного счета на другой.
2. Удаление внесенных данных. Из нескольких таблиц.
3. Перечисление зарплаты работникам из фирмы.
4. Заказ билетов на самолет.
5. Резервирование мест в гостинице.
6. Аренда автомобиля.
7. Автоматизировать систему прописки.
8. Написать транзакцию для получения наличных денег из автоматического кассового автомата.
9. Запись товара в магазин на склады, используя штрих-код продукта.
10. Подготовка соглашений о сдаче объектов в аренду.
11. Запись в телефонную книгу. Используя максимум данных.
12. Покупки через интернет.
13. Оплата штрафов (пример с карт. Счета, через мультикассу или иными способами).
14. Регистрация по месту прибывания.

Содержание отчета

Отчет о выполнении лабораторной работы должен содержать:

- название и тему лабораторной работы;
- цель лабораторной работы;
- краткие теоретические сведения;
- ход выполнения работы;
- выводы.

Раздел «Ход выполнения работы» должен содержать инфо логическую модель БД на языке таблица-связь, описание разработанных ограничений целостности, рисунки с отчетами Data Dictionary Report для каждой таблицы БД.

Контрольные вопросы

1. Что такое транзакция?
2. Перечислите виды транзакции.
3. Краткое описание видов транзакции.
4. Основные команды для работы с транзакциями?
5. Что такое хранимая процедура?
6. Что такое откатка? И в каких случаях происходит?
7. Свойства транзакции.
8. Источники отказов.
9. Что такое фиксированная транзакция?
10. Классификация систем обработки транзакций

Лабораторная работа №4. Архитектура реляционных баз данных. 4 часа.

Цель работы: изучить архитектуру баз данных СУБД, поддерживающих реляционную модель данных; научиться средствами СУБД создавать базу данных и работать с ее объектами.

Теоретическая часть

СУБД PostgreSQL - это кроссплатформенная свободно распространяемая объектно-реляционная система управления базами данных, наиболее развитая из открытых (open source) СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.

Работа с данными в PostgreSQL основана на объектно-реляционной модели, что позволяет задействовать сложные процедуры и системы правил. Примерами нетривиальных возможностей этой категории являются декларативные запросы SQL, контроль параллельного доступа, поддержка многопользовательского доступа, транзакции, оптимизация запросов, поддержка наследования и массивов.

В PostgreSQL поддерживаются пользовательские операторы, функции, методы доступа и типы данных.

Полноценная поддержка SQL. PostgreSQL соответствует базовой спецификации SQL99.

Гибкость API. Гибкость API PostgreSQL позволяет легко создавать интерфейсы к реляционной СУБД PostgreSQL. В настоящее время существуют программные интерфейсы для Object Pascal, Python, Perl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/ C+ и Pike.

Процедурные языки. В PostgreSQL предусмотрена поддержка внутренних процедурных языков, в том числе специализированного языка PL/pgSQL, являющегося аналогом PL/SQL, процедурного языка Oracle. Одним из преимуществ PostgreSQL является возможность использования

Perl, Python и TCL в качестве внутренних процедурных языков.

Технология MVCC. MVCC (Multiversion Concurrency Control) используется в PostgreSQL для управления конкурентным доступом к данным на многовариантной основе. Эта технология позволяет предотвращать лишние блокировки (locking) операций чтения операциями, производящими обновление записей. PostgreSQL отслеживает все транзакции, выполняемые пользователями базы данных, что позволяет работать с записями без ожидания их освобождения. На практике это означает, что при запросе к БД каждая транзакция видит, как бы снимок данных (версию) на момент этого снимка, а не текущее состояние данных. Таким образом, транзакции защищаются от просмотра незафиксированных данных, которые в данный момент могут только формироваться конкурентными транзакциями в тех же самых строках таблицы. Основное преимущество MMVC состоит в том, что чтение данных никогда не блокирует запись, а запись никогда не блокирует чтение. MMVC позволяет избегать явного блокирования на уровне таблиц и отдельных записей, которое используется в традиционных СУБД, и, таким образом, минимизирует блокирование данных и увеличивает производительность в многопользовательских системах БД. Также реализовано отслеживание взаимных блокировок (deadlocks).

Клиент-серверная архитектура. В PostgreSQL используется архитектура «клиент-сервер» с распределением процессов между пользователями. В целом она напоминает методику работы с процессами в сервере Apache. Главный (master) процесс создает дополнительные подключения для каждого клиента, пытающегося установить соединение с PostgreSQL.

Полнотекстовый поиск. Начиная с версии 8.3 в ядро PostgreSQL включена функция полнотекстового поиска. Полнотекстовый поиск позволяет создавать такие запросы к текстовым документам, которые могут гибко настраиваться в зависимости от конкретных потребностей.

Сохраняющая регистрация WAL. WAL (Write-Ahead Logging) является стандартным методом для обеспечения целостности данных. Сохраняющая регистрация - метод регистрации (журналирования) транзакций, при котором запись в журнале делается до записи данных. Используется также в MS SQL Server. Суть WAL заключается в том, что изменения в файлах данных (таблицы и индексы) должны быть внесены только после записи в журнал (log), в котором фиксируются эти изменения. Эта процедура позволяет не переписывать страницы данных на диске при каждой транзакции, так как в случае аварии можно восстановить базу данных с помощью журнала. Механизм WAL обеспечивает следующие преимущества:

- повышение производительности работы СУБД за счет того, что записываются только внесенные изменения без переписывания всех данных в таблицах;
- повышение надежности хранения данных за счет предыдущего сохранения буферизованных данных в WAL;

- возможность отката состояния БД на любой момент времени, путем применения WAL к существующей резервной копии.

Репликация и технология Hot Standby. Начиная с версии 9.0, на основе WAL введена репликация по технологии Hot Standby. Технология позволяет получить на сервере вторую базу данных, которая является актуальной копией оригинальной базы данных, доступной лишь для чтения.

Технология может быть использована также и на отдаленном сервере, который подключается к primary- или master-серверу и загружает из него WAL-логи, предоставляя онлайн-овую репликацию базы данных и поддерживая копию базы данных на отдаленном сервере в актуальном состоянии, а также делая эту копию доступной для запросов на чтение.

Наследование. Таблицы могут наследовать характеристики и наборы полей от других таблиц (родительских). При этом данные, которые добавляются в порожденную таблицу, автоматически будут принимать участие в запросах к родительской таблице. Данная функция в настоящее время не является полностью реализованной, однако ее можно использовать;

Гибкая настройка сервера. Основным конфигурационным файлом postgresql.conf, включает более 150 настраиваемых параметров по разделам: файлы и пути к ним, авторизация и безопасность, выделение ресурсов и т.д. Дополнительный конфигурационный файл pg_hba.conf включает в себя настройку доступа к отдельным БД, такие как указание конкретных IP-адресов и (или) сетей, из которых разрешен доступ, а также метод авторизации для доступа в БД и возможность включения безопасных (зашифрованных) соединений.

Хранение больших значений. Таблицы TOAST

Для хранения больших по размеру данных в PostgreSQL используется техника TOAST (The Oversized-Attribute Storage Technique). PostgreSQL использует фиксированный размер страницы (обычно 8 Кб), и не позволяет кортежам занимать несколько страниц. Таким образом, невозможно хранить очень большие значения полей непосредственно в таблице. Чтобы преодолеть это ограничение, большие значения сжимаются и/или разбиваются на несколько физических строк. Этот процесс прозрачен для пользователя.

TOAST поддерживается только для определенных типов данных. Для поддержки TOAST тип данных должен иметь переменную длину (varlena), причем первое 32-разрядное слово содержит общую длину значения в байтах, включая само это слово.

TOAST резервирует два бита varlena-слова: старшие биты для машин с обратным порядком байтов (big-endian) и младшие биты для машин с прямым порядком байтов (little-endian). Тем самым размер любого значения ограничивается 1 Гб (2³¹ - 1 байт). Нулевые значения битов соответствуют обычным (не TOAST) значениям. В противном случае возможны две комбинации:

- самый старший/младший бит varlena-слова установлен, смежный бит сброшен. Это указывает на то, что данное значение имеет 1-байтовое, а

не 4-байтовое, varlena-слова, а оставшиеся биты этого слова дают общий размер значения (включая байт длины) в байтах. Если остальные биты varlena-слова равны нулю, то данное значение является указателем на данные, хранящиеся в отдельной TOAST-таблице. При этом размер TOAST - указателя дает второй байт значения. Данные с однобайтовыми заголовками не выравниваются по какой-либо конкретной границе;

- самый старший/младший бит varlena-слова сброшен, смежный бит установлен. Это говорит о том, что значение было сжато и должно быть распаковано перед использованием. В этом случае оставшиеся биты длины слова дают общий размер сжатых данных, а не исходных данных. Сжатие также возможно и для значений, хранящихся в TOAST-таблицах (информацию об этом содержит TOAST-указатель).

Механизм TOAST запускается только тогда, когда строка, которая будет храниться в таблице превышает TOAST_TUPLE_THRESHOLD байт (обычно 2 Кб). Значение столбца, для которого поддерживается TOAST, будет сжиматься и/или перемещаться в TOAST-таблицу пока длина строки не станет короче, чем TOAST_TUPLE_TARGET байт (также обычно 2 Кб). Механизм TOAST использует четыре различные стратегии для хранения данных:

- PLAIN предусматривает либо сжатие, либо хранение вне основной таблицы, кроме того, он исключает использование однобайтовых varlena-заголовков (единственная возможная стратегия для столбцов, для которых не поддерживается TOAST);

- EXTENDED позволяет как сжатие и хранение вне основной таблицы. Это - стратегия, используемая по умолчанию для большинства TOAST-типов данных. Сначала будет предпринята попытка сжатия, а затем - запись в TOAST-таблицу, если строка по-прежнему слишком велика;

- EXTERNAL позволяет хранение вне основной таблицы, но не сжатие. Использование этой стратегии позволяет ускорить подстроковые операции в текстовых и байтовых полях быстрее (но за счет увеличения пространства для хранения), поскольку эти операции оптимизированы для извлечения необходимых частей данных, когда они не сжаты;

MAIN позволяет сжатие, но не хранение вне основной таблицы. На самом деле, хранение вне основной таблицы будет по-прежнему доступно, но только в качестве крайней меры, когда нет возможности уменьшить строку так, чтобы она поместилась на странице.

Формат страницы базы данных

Каждая таблица и индекс в PostgreSQL хранится как массив страниц фиксированного размера (обычно 8 Кб). В таблице, все страницы логически эквивалентны, так что та или иная строка может быть сохранена на любой странице. В индексах, первая страница, как правило, защищена в качестве метастраницы, содержащей управляющую информацию, и в индексе могут быть различные типы страниц, в зависимости от метода доступа к индексу.

Каждая страница состоит из пяти частей:

- PageHeaderData (24 байт) - содержит общую информацию о странице,

в том числе указатель свободного пространства;

- ItemldData - массив пар (смещение, длина) по 4 байта каждая, указывающих на фактические элементы данных, хранящихся на странице;
- Free space - свободное место. Новые указатели на элементы данных локализуются в начале этой области, новые элементы - с конца;
- Items - фактические данные;
- Special space - специфические данные с индексным методом доступа, различным для различных данных (эта область - пустая в обычных таблицах).

Первые 24 байта на каждой странице отводятся под заголовок страницы (PageHeaderData), состоящий из нескольких полей, в которых содержится информация о частях страницы, описанных выше.

После заголовка страницы следуют 4-байтные идентификаторы элементов данных (ItemldData). Идентификатор элемента содержит байт - смещение начала элемента, его длину в байтах и несколько бит атрибутов, которые влияют на его интерпретацию. Новые идентификаторы выделяются по мере необходимости с начала свободного пространства. Поскольку идентификатор элемента никогда не смещается пока он не будет освобожден, его индекс может быть использован на долгосрочной основе для ссылки на элемент, даже если сам предмет перемещается по странице с целью сделать свободное пространство более компактным. Фактически, каждый указатель на элемент данных (ItemPointer, также известный как СТШ), созданный PostgreSQL, состоит из номера страницы и индекса идентификатора элемента.

Сами элементы данных хранятся в пространстве, выделяемом в обратном порядке от конца свободного пространства. Точная структура хранения меняется в зависимости от того, что таблица будет содержать. Все строки таблиц и курсоров используют структуру под названием HeapTupleHeaderData, в которой выделяется заголовок фиксированного размера (занимает 23 байта на большинстве машин), в котором содержатся физические параметры хранения строк, далее следуют необязательные битовый массив NULL-значений, ID объекта и пользовательские данные (т. е. собственно записи). Информация о наличии битового массива NULL-значений и ID объекта также указывается в заголовке.

Если битовый массив NULL-значений присутствует, он начинается сразу после фиксированного заголовка и занимает столько байт, сколько необходимо из расчета один бит на столбец данных. В этом массиве бит 1 указывает на не NULL-значение, бит 0 - на NULL- значение. Если битовый массив NULL-значений отсутствует, то предполагается, что ни один из столбцов не имеет NULL-значения.

Если ID объекта присутствует, то он размещается между битовым массивом NULL- значений и началом фактических данных с учетом того, что последние выравниваются по границе, определяемой параметром MAXALIGN для данной платформы, т.е. значение, соответствующее ID объекта, может быть дополнено нулевыми битами (padding value).

Чтобы прочитать данные, соответствующие каждому атрибуту по

очереди, сначала проверяется, имеет ли поле NULL-значение в соответствии с битовым массивом NULL- значений. Если это так, переходим к следующему полю. Если поле фиксированной длины, то все байты просто считываются с учетом выравнивания. Если поле переменной длины, то оно имеет заголовок `varlena`, который включает в себя общую длину сохраненного в этом поле значения, и некоторые флаговые биты. В зависимости от флагов, этот элемент данных может быть размещен в основной таблице либо в TOAST- таблице, он также может быть сжат.

Логическая архитектура баз данных PostgreSQL

К особенностям логической архитектуры PostgreSQL, кроме упоминавшихся выше табличных пространств, можно отнести следующие моменты: схемы, индексы, роли и привилегии, правила, хранимые процедуры и триггеры, функции, типы данных.

Схемы. PostgreSQL поддерживает схемы. Схемы являются как бы дополнительными областями видимости внутри БД. Также схему можно сравнить и с дополнительным путем (название схемы должно указываться перед названием таблицы) и с каталогом, внутри которого можно разместить таблицы. В любой БД по умолчанию существует схема `public`, в которой создаются все таблицы и которую не нужно указывать специально. Администратор БД может создавать другие схемы (и разграничивать доступ к ним), что обеспечивает еще один уровень распределения прав доступа для пользователей, позволяет выделить каждому пользователю как бы персональный раздел внутри БД с теми же названиями таблиц, что и у других пользователей.

Индексы. В PostgreSQL существует 4 типа индексов: B-tree, Hash, GiST (Generalized Search Tree) и GIN (Generalized Inverted Index). Каждый тип индекса имеет свой алгоритм реализации, что позволяет существенно увеличить быстродействие, если для определенного вида данных выбирать определенный тип индекса.

PostgreSQL позволяет также создавать индексы с использованием выражений и частичные (partial) индексы (с использованием служебного слова `WHERE`).

Роли и привилегии. PostgreSQL управляет привилегиями в БД, используя концепцию ролей. Ролью может быть, как отдельный пользователь БД, так и группа пользователей. Роли могут быть владельцами объектов в БД (например, таблиц), а также могут назначать привилегии доступа к этим объектам для других ролей. Возможно предоставить одной роли членство в другой роли и соответственно передать этой роли права той роли, членом которой она будет. Концепция ролей заменила старую концепцию пользователей и групп, предоставив ту же функциональность. Начиная с версии 9.0 в PostgreSQL поддерживаются права на схемы `in` права по умолчанию.

Правила

Механизм правил (rules) представляет собой механизм создания пользовательских обработчиков не только операций манипулирования, но и

операции выборки данных. Основное отличие от механизма триггеров заключается в том, что правила срабатывают на этапе разбора запроса, до выбора оптимального плана выполнения и самого процесса выполнения. Правила позволяют переопределять поведение системы при выполнении SQL- операций к таблице. Например, при создании представления (views) создается правило, которое определяет, что вместо выполнения операции выборки к представлению система должна выполнять операцию выборки к базовой таблице/таблицам с учетом условий выборки, которые лежат в основе определения представления. Для создания представлений, которые поддерживают операции обновления, правила для операций вставки, изменения и удаления строк, должны быть определены пользователем.

Система правил (более правильно говорить: система правил изменения запросов) позволяет изменять запрос согласно заданным правилам и потом передает измененный запрос планировщику запросов для планирования и выполнения. Система правил является очень мощным инструментом и может быть использована во многих случаях, таких как хранимые процедуры и представления.

Хранимые процедуры и триггеры

Хранимые процедуры в PostgreSQL могут быть написаны на любом из поддерживаемых встроенных языков. Хранимые процедуры могут быть использованы в триггерах и могут возвращать любой из поддерживаемых типов данных, а также массивы и списки. Начиная с версии 9.0, вызывать хранимые процедуры можно с указанием именуемых параметров.

Триггеры определяются как функции, которые инициируются операциями манипулирования. Триггеры могут быть назначены до или после операций INSERT, UPDATE или DELETE. Если произошло событие, на которое был назначен триггер, то вызывается закрепленная за этим триггером процедура. Например, операция INSERT может запускать триггер, проверяющий прибавленную запись на соответствии определенным условиям. При написании функций для триггеров могут использоваться разные языки программирования. Триггеры ассоциируются с таблицами и выполняются в алфавитном порядке.

В версии 9.0 введены триггеры на столбцы и, кроме того, при объявлении триггера можно использовать ключевое слово WHEN, которое добавляет дополнительное условие для срабатывания триггера.

Функции

Функции являются блоками кода, выполняемыми на сервере, а не на стороне клиента БД. Иногда функции отождествляются с хранимыми процедурами, однако между этими понятиями есть разница. Хотя они могут создаваться на чистом SQL, реализация дополнительной логики, например, условных переходов и циклов, выходит за рамки, собственно, SQL и требует использования некоторых языковых расширений. Функции могут писаться на одном из таких языков:

- Встроенный процедурный язык PL/pgSQL, во многом аналогичный языку PL/SQL, что используется в СУБД Oracle;

- Скриптовые языки – PL/Lua, PL/LOLCODE, PL/Perl, PL/PHP, PL/Python, PL/Ruby, PL/sh, PL/Tcl и PL/Scheme ;
- Классические языки – C, C ++, Java (через модуль PL/Java);
- Статистический язык R (через модуль PL/R).

PostgreSQL допускает использование функций, которые возвращают набор записей, который дальше можно использовать так же, как и результат выполнения обычного запроса (курсор). Функции могут выполняться как с правами их создателя, так и с правами текущего пользователя.

Начиная с 9.0, можно создавать функции без объявления имени (анонимные блоки) для выполнения блока операторов на любом встроенном языке, который поддерживает PostgreSQL, прямо в командной строке.

Типы данных

PostgreSQL поддерживает большой набор встроенных типов данных:

- Численные типы
 - Целые
 - С фиксированной точкой
 - С плавающей точкой
 - Денежный (отличается специальным форматом вывода, а в остальном аналогичен числам с фиксированной точкой и двумя знаками после запятой)
- Символьные типы произвольной длины
- Двоичные типы (включая большой двоичный объект BLOB – Binary Large Object)
- Типы «дата/время» (полностью поддерживают разные форматы, точность, форматы вывода, включая последние изменения в часовых поясах)
- Логический тип
- Перечислимый тип
- Геометрические примитивы
- Сетевые типы
 - IP i IPv6 -адреса
 - CIDR –формат

CIDR (Classless Inter-Domain Routing) – бесклассовая адресация – метод IP- адресации, позволяющий гибко управлять пространством IP-адресов, не используя жёстких рамок IP-адресации на основе классов сетей.

- MAC –адреса
- Уникальный идентификатор, присваиваемый каждой единице сетевого оборудован.*

- UUID тип
- UUID (Universally Unique Identifier) — это стандарт идентификации, используемый при создании программного обеспечения. Наиболее распространённым использованием данного стандарта является Globally Unique Identifier (GUID) фирмы Microsoft.*

- XML тип
- XML (Xtensible Markup Language) – текстовый формат, предназначенный для хранения структурированных данных (взамен*

существующих файлов баз данных), для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки.

- Массивы
- OID –типы

OID (Object identifiers) представляют идентификаторы различных объектов и используются обычно в PostgreSQL как первичные ключи для различных системных таблиц. Эти типы представляются как 4-байтовое целые числа без знака, т.е. имеют достаточно ограниченный диапазон значений, поэтому не могут использоваться в больших базах данных.

- Композитные типы

Композитный (составной) тип (composite type) представляет структуру ряда или записи, т.е. по существу список имен полей и их типов данных.

- Псевдотипы

Псевдотипы (pseudo-types) не могут использоваться в качестве типа данных столбца таблицы или представления, но могут использоваться для объявления аргументов функции или типа результата.

С любым объектом данных, представленным в PostgreSQL, связывается определенный тип, даже если на первый взгляд это и не очевидно. Тип данных одновременно определяет и ограничивает разновидности операций, которые могут выполняться с этими данными.

Хотя большинство типов данных PostgreSQL взято непосредственно из стандартов SQL, существуют и другие, нестандартные типы данных (например, геометрические и сетевые типы). В табл. 1 перечислены основные базовые типы данных PostgreSQL, а также их синонимы (альтернативные имена).

Таблица 1. Типы данных PostgreSQL

Тип данных	Описание	Стандарт
Логические и двоичные типы данных		
boolean, bool	Отдельная логическая величина (true или false)	SQL99
bit(n)	Битовая последовательность фиксированной длины (ровно n бит)	SQL92
bit varying(n), varbit(n)	Битовая последовательность переменной длины (до n бит)	SQL92
Символьные типы		
character(n), char(n)	Символьная строка фиксированной длины (ровно n символов)	SQL89
character varying(n), varchar(n)	Символьная строка переменной длины (до n символов)	SQL92
text	Символьная строка переменной или неограниченной длины	PostgreSQL
Числовые типы		

small int, int2	2-байтовое целое со знаком	SQL89
integer, int, int4	4-байтовое целое со знаком	SQL92
bigint, int8	8-байтовое целое со знаком, до 18 цифр	PostgreSQL
real, float4	4-байтовое вещественное число	SQL89
double precision, floats, float	8-байтовое вещественное число	SQL89
numeric(p.s), decimal (p.s)	Число из p цифр, содержащее 5 цифр в дробной части	SQL99
money	Фиксированная точность, представление денежных величин	PostgreSQL, считается устаревшим
serial	4-байтовое целое с автоматическим приращением	PostgreSQL
Время и дата		
date	Календарная дата (день, месяц и год)	SQL92
time	Время суток	SQL92
time with time zone	Время суток с информацией о часовом поясе	SQL92
timestamp	Дата и время	SQL92
interval	Произвольный интервал времени	SQL92
Геометрические типы		
box	Прямоугольник на плоскости	PostgreSQL
line	Бесконечная линия на плоскости	PostgreSQL
lseg	Отрезок на плоскости	PostgreSQL
circle	Круг с заданным центром и радиусом	PostgreSQL
path	Замкнутая или разомкнутая геометрическая фигура на плоскости	PostgreSQL
point	Точка на плоскости	PostgreSQL
polygon	Замкнутый многоугольник на плоскости	PostgreSQL
Сетевые типы		

cidr	Спецификация сети IP	PostgreSQL
inet	Сетевой IP-адрес с необязательными битами подсети	PostgreSQL
macaddr	MAC-адрес (например, аппаратный адрес адаптера Ethernet)	PostgreSQL
Системные типы		
oid	Идентификатор объекта (записи)	PostgreSQL
xid	Идентификатор транзакции	PostgreSQL

Помимо встроенных типов данных, пользователь может самостоятельно создавать новые необходимые ему типы и программировать для них механизмы индексирования с помощью методов GiST.

Пользовательские объекты

PostgreSQL может быть расширен пользователем для собственных потребностей практически в любом аспекте. Есть возможность добавлять:

- Типы данных и их преобразования
- Домены
- Функции (включая агрегатные)
- Индексы
- Операторы (включая переопределение уже существующих)
- Процедурные языки

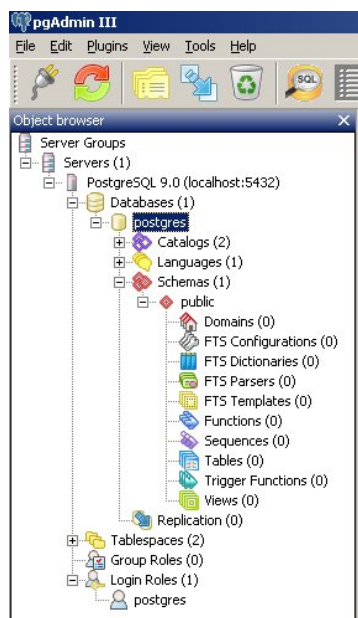


Рис. 1. Основные компоненты базы данных PostgreSQL 9.0.

Как и в SQL Server, данные в базе данных PostgreSQL организованы в нескольких разных объектах:(рис. 1):
домены (domains);

- ~ конфигурация полнотекстового поиска (FTS configuration);
- ~ словари полнотекстового поиска (FTS dictionaries);
- ~ синтаксические анализаторы полнотекстового поиска (FTS parsers);
- ~ шаблоны полнотекстового поиска (FTS templates);
- ~ функции (functions);
- ~ последовательности (sequences);
- ~ таблицы (tables);
- ~ триггеры (trigger functions);
- ~ представления (views).

2. Разработка логической модели базы данных

2.1. Средства для разработки и администрирования баз данных.

Для работы с базами данных существует несколько возможностей:

- Запуск интерактивной терминальной программы, которая позволяет вводить, редактировать и выполнять команды SQL:

СУБД	Интерфейс командной строки
MS SQL Server	isqlw
PostgreSQL	psql

- Использование пакета с графическим интерфейсом(GUI):

СУБД	GUI
MS SQL Server	SQL Server Enterprise Manager
PostgreSQL	pgAdmin

- Написание специального приложения, используя один из нескольких доступных языков программирования, которые поддерживаются СУБД.

Далее будет рассмотрена работа в среде PostgreSQL с использованием pgAdmin.

2.2. Пример создания базы данных.

Шаг 1: Создание базы данных BookShop

Параметры новой базы данных показаны на рис.2

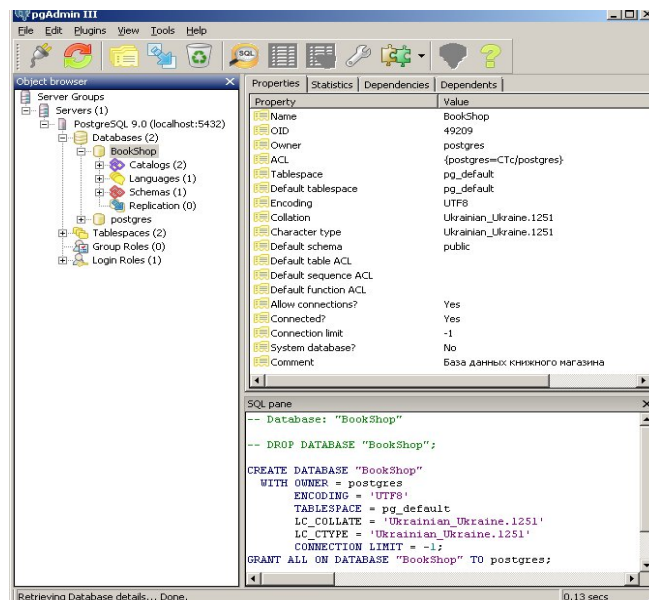


Рис. 2. Параметры базы данных BookShop.

При создании новой базы данных в системную таблицу pg_database (см. pg_catalog/Tables) добавляется строка, содержащая параметры этой базы данных.

Шаг 2: Создаем таблицы Книги, Поставщики, Заказчики, Заказы, Поставки.

После того, как база данных создана, можно приступить к созданию ее основных объектов – таблиц. Прежде всего, для каждого столбца любой таблицы необходимо указать определенный тип данных. В SQL тип данных задается после имени столбца с помощью соответствующего ключевого слова, потом вводятся параметры представления значений столбцов, такие как длина, значение по умолчанию и др. После определения тип данных столбца таблицы сохраняется в виде постоянной характеристики столбца и не может быть изменен.

В отношении типов данных следует помнить, что недопустимо называть объекты именами команд или использовать для этой цели другие зарезервированные слова. Типы данных – это полноценные объекты базы данных, которые хранятся в системной таблице pg_type вместе с их OID.

Создаем таблицы со следующими полями:

Таблица	Поле	Тип
Книги	Код книги	serial
	Автор	character varying(80)
	Название	character varying(160)
	Издательство	character varying(80)
	Цена	money
	Остаток	smallint
Поставщики	Код поставщика	serial
	Название	character varying(40)
	Город	character varying(40)
	Адрес	character varying(80)

	Телефон	character(13)
Заказы	Код заказа	serial
	Код книги	integer
	Код заказчика	integer
	Оплачен	character varying(3)
	Дата	date
Заказчики	Код заказчика	serial
	Имя	character varying(40)
	Адрес	character varying(80)
	Телефон	character(13)
Поставки	Номер	serial
	Код книги	integer
	Код поставщика	Integer
	Количество	integer
	Дата	date

Шаг 3: Введение ограничений целостности.

Следующий момент в процессе создания таблиц, которому необходимо уделить особенное внимание, связан с обеспечением целостности данных. Для обеспечения целостности данных в таблицах определяются ограничения на значения столбцов (constraints). Эти ограничения могут быть введены при создании таблицы для каждого столбца в отдельности или добавлены в таблицу позже с помощью специальной команды SQL ALTER TABLE. В PostgreSQL поддерживаются следующие основные ограничения целостности:

- PRIMARY KEY – первичный ключ
- FOREIGN KEY/REFERENCES – внешний ключ (ссылка)
- UNIQUE – уникальность
- CHECK – проверка условия на значение

Ограничение первичного ключа на значение столбца используется для обеспечения уникальности данных в столбцах и в целом для обеспечения ссылочной целостности (при связывании таблиц посредством внешних ключей). Определение условия primary key для таблицы имеет несколько эффектов. Во-первых, оно устанавливает определенные условия на значение первичного ключа – запрещается введения одинаковых значений и значений NULL в те столбцы, для которых оно определено. Во-вторых, primary key создает уникальный индекс для этих столбцов, что позволяет ускорить поиск строк в таблице.

Определение условия primary key в одной таблице сам по себе не обеспечивает целостность по ссылкам. Необходимо также определить соответствующие внешние ключи тех таблиц, строки которых будут комбинироваться со строками той таблицы, где определено ограничение на значение столбца PRIMARY KEY.

Ограничение внешнего ключа на значение столбца обычно применяется вместе с предварительно определенным ограничением primary key (на самом деле достаточно ограничения UNIQUE) в ассоциируемой

таблице. Условие на значение foreign key ставит в соответствие один или несколько столбцов таблицы идентичному набору столбцов другой таблицы, для которых определено ограничение primary key (или UNIQUE). Когда обновляются или удаляются значения тех столбцов таблицы, на которые ссылаются внешние ключи других таблиц, возникает вопрос: что делать с соответствующими значениями внешних ключей? Существует несколько вариантов решения этой проблемы:

- ничего не делать (no action) (это событие должно обрабатываться неким отличным от стандартного способом, иначе будет выдаваться сообщение об ошибке);
- запретить любые изменения(restrict);
- автоматически обновить/удалить значения соответствующих внешних ключей (cascade),
- установить для внешних ключей NULL-значения (set null) (для этого соответствующие столбцы не должны иметь ограничения NOT NULL);
- установить для внешних ключей значения по умолчанию (set default).

Автоматическое обновление соответствующих столбцов в разных таблицах после того, как для них определены ограничения на значение столбцов primary key и foreign key, называется декларативной ссылочной целостностью (declarative referential integrity).

Ограничение на значение столбцов primary key и foreign key обеспечивают соответствие строк связанных таблиц, потому столбцы с такими ограничениями используются для реализации операции соединения таблиц.

Наглядное представление о структуре связей между таблицами в базе данных можно получить с помощью диаграмм «таблица-связь», на которых указываются ограничения primary key и foreign key и та³к⁸-ая характеристика связей, как степень связи. На рис. 3 показана диаграмма базы данных BookShop.

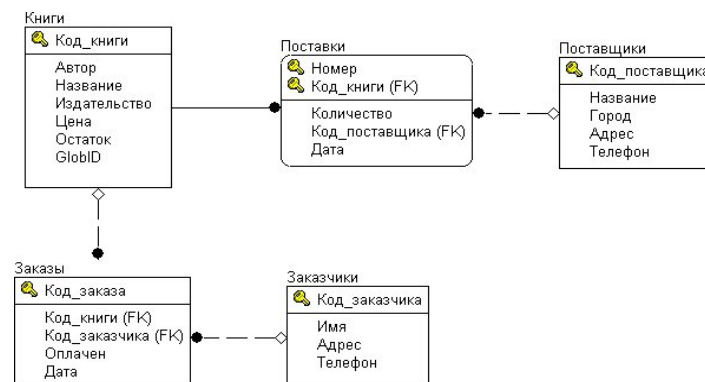


Рис. 3. Диаграмма «таблица-связь» базы данных BookShop.

Сплошная линия – идентифицирующие связи, пунктирные – не идентифицирующие связи. Все связи имеют степень 1:N (N – со стороны

зависимой таблицы).

Ограничение уникальности на значение столбца можно назначить для того, чтобы запретить повторение значений в любом столбце таблицы. Для столбца, входящего в состав первичного ключа, подобное ограничение не может быть определено, т.к. ограничение уникальности реализуется с помощью автоматического создания уникального индекса для столбца таблицы, а для первичного ключа также автоматически создается уникальный индекс. Однако, столбец (или столбцы, если ограничение UNIQUE определено сразу для нескольких столбцов) с ограничением UNIQUE может быть использован для связи с другими таблицами, т.е. на него могут ссылаться внешние ключи этих таблиц.

Проверочное ограничение на значение столбца устанавливает диапазон значений, которые могут быть введены в один или несколько столбцов таблицы базы данных. Ограничение check может использоваться, например, для того чтобы установить диапазон значений, которые допускается хранить в столбце, определенном для данных числовых типов. Процесс установки проверки значения для столбца таблицы называется связыванием (binding). Можно определить и ввести несколько проверок в один столбец. Проверка может быть определена для столбца даже в том случае, если для него уже существует какое-либо правило.

Хотя проверочные ограничения на значение работают быстрее и устанавливаются проще, но правила гибче. После определения правила оно может быть связано со столбцами нескольких таблиц, но для одного столбца можно задать лишь одно правило и несколько проверочных условий.

Далее вводим для столбцов созданных таблиц следующие ограничения:

Таблица	Поле	Ограничение
Поставщики	Код_поставщика	Primery key
	Название	Unique
	Телефон	Defaults (“000 111-11-11”)
Заказы	Код_заказа	Primery key
	Код_книги	Foreign key (Код_книги из Книги, NULL-значения не допускаются, автомат. обновление при изменении Код_книги из Книги, удаления из Книги не допускаются).
	Код_заказчика	Foreign key (Код_заказчика из Заказчики, NULL-значения не допускаются, автомат. обновление при изменении Код_заказчика из Заказчики, удаления из Заказчики не допускаются)
	Оплачен	Check (“Так”, “Hi”)
	Дата	Defaults (timenow())
Заказчики	Код_заказчика	Primery key

	Телефон	Defaults ("000 111-11-11")
Книги	Код книги	Primery key
Поставки	Номер	Primery key
	Код_книги	Primery key Foreign key (Код_книги из Книги, NULL-значения не допускаются, обновления при изменении Код_книги из Книги не допускаются, удаления из Книги не допускаются).
	Код_поставщика	Foreign key (Код_поставщика из Поставщики, NULL- значения допускаются, автомат. обновления при изменении Код_поставщика из Поставщики, при удалении из Поставщики установить в NULL)

В заключение для каждой таблицы генерируем отчет Data Dictionary Report. Для таблицы Поставки, например, он должен выглядеть так:

Table Data dictionary report - Поставки Generated: 25.09.2011 19:49:30

Server: PostgreSQL 9.0 (localhost:5432)

Database: BookShop Schema: public Columns

Name	Data type	Not Null?	Primary key?	Default	Comment
Номер	integer	Yes	Yes	nextval("Поставки_Номер_seq"::regclass)	
Код_книги	integer	Yes	No		
Код_поставщика	integer	No	No		
Количество	integer	No	No		
Дата	date	No	No		

Constraints

Name	Type	Definition	Comment
pk_поставки	Primary key	("Номер")	
fk_поставки_1	Foreign key	("Код_книги") REFERENCES "Книги" ("Код_книги") MATCH SIMPLE ON UPDATE RESTRICT ON DELETE RESTRICT	
fk_поставки_2	Foreign key	("Код_поставщика") REFERENCES "Поставщики" ("Код_поставщика") MATCH SIMPLE ON UPDATE CASCADE	

		ON DELETE SET NULL	
--	--	--------------------	--

Шаг 4: Ввод данных в таблицы.

Для просмотра и манипулирования данными в таблицах в pgAdmin предназначены соответствующие команды из меню Tools или кнопки на панели инструментов. При вводе данных в таблицы следует помнить об ограничениях, которые были определены для столбцов и таблиц базы данных. Не следует забывать, что: 1) внешние ключи могут принимать только те значения, которые уже введены (!) в поля тех таблиц, на которые эти внешние ключи ссылаются; 2) в столбцы с типом serial (счетчик) нельзя самим вводить значения; 3) если в столбец, для которого определено значение по умолчанию, не водится никакого значения, то автоматически будет введено значение по умолчанию.

Для того, чтобы ввести в таблицу новое значение, необходимо просто перейти к строке, отмеченной звездочкой (*) и начать вводить необходимые значения в соответствующие столбцы.

После того, как введены все новые строки или после изменения данных, необходимо зафиксировать этот факт в базе данных, для чего нужно выполнить команду Refresh, нажав одноименную кнопку на панели инструментов окна Edit Data.

Чтобы удалить какую-либо строку таблицы, достаточно выделить ее, щелкнув левой кнопкой мыши по ее номеру, и выполнить команду Delete контекстного меню. Но при этом следует помнить о целостности данных по ссылкам, если она применяется в базе данных.

Вводим несколько произвольных строк в таблицы базы данных BookShop с учетом определенных ограничений на значения столбцов.

3. Контрольное задание.

3.1. Используя метод нормализации универсального отношения, разработать инфологическую модель базы данных, определить ограничения целостности, создать БД и ввести тестовые данные в каждую из созданных таблиц.

№	Название разрабатываемой БД и атрибуты универсального отношения.
1	Торговля Код изделия, наименования, марка, производитель, номер приходной накладной, дата поступления, на состав, количество единиц, закупочная цена, розничная цена, номер счета-фактуры, дата продажи, количество проданных единиц, сумма, тип платежа нал/безнал, банковские реквизиты покупателя для безналичного расчета.
2	Комунальные платежи.

	Код услуги, наименование, единица измерения, тарифная зона, стоимость единицы, лицевой счет клиента, ФИО клиента, адрес, телефон, месяц/год, объем потребления услуги, сумма к оплате, сумма задолженности.
3	Услуги. Личный номер клиента, ФИО клиента, дата рождения, домашний адрес, телефон, дата и время приема заказа, тип выполняемой работы, тариф, личный номер мастера, ФИО мастера, номер ордера на выполнение заказа, дата и время начала работы, дата и время закрытия ордера, объем выполненной работы, стоимость.
4	Начисление зарплаты. Номер личного дела, ФИО сотрудника, домашний адрес, домашний телефон, рабочий телефон, дата приема, на работу, стаж работы по специальности, общий стаж работы, образование, квалификация, должность, ставка заработной платы, месяц/год, количество рабочих дней за месяц, фактически отработанных дней, премиальные, отпускные, удержания в процентах от начисления, аванс, сумма, к выдаче.
5	Поставки. Код продукции, наименование продукции, единица измерения, код производителя, месяц и год выпуска, код поставщика, юридическое наименование поставщика, юридический адрес поставщика, банковские реквизиты поставщика, телефон поставщика, номер договора на поставку, дата подписания договора, срок поставки, количество единиц, показатель качества, оптовая цена, условия поставки.
6	Билетная касса. Номер рейса, пункт отправления, пункт назначения, время отправления, время прибытия, категория билета, стоимость билета, тип транспортного средства, общее количество мест, количество мест данной категории, уникальный порядковый номер пассажира, ФИО пассажира, дата и время отправления по билету, дата и время продажи билета, дата и время бронирования билета.
7	Отдел кадров. Личный номер сотрудника, ФИО сотрудника, номер паспорта, дата рождения, домашний адрес, домашний телефон, образование, номер диплома, код специальности, наименования специальности, квалификация, код должности, наименования должности, должностной оклад, дата прохождения повышения квалификации, присвоена квалификация, свидетельство о повышении квалификации, дата поощрения, вид поощрения, дата наложенного взыскания, вид взыскания.

8	Отель.
	Номер паспорта клиента, ФИО, дата рождения, гражданство, номер комнаты, категория, стоимость проживания за сутки, дата и время поселения, дата и время выселения, дата бронирования, сумма к оплате, личный номер администратора, ФИО администратора.
9	Производство.
	Код продукции, наименование, код сырья, наименования сырья, норма затрат сырья для производства единицы продукции, процент потерь при изготовлении продукции, себестоимость продукции, оптовая цена, год выпуска, план выпуска, фактически произведено, процент бракованной продукции, валовой доход, чистая прибыль, номер
	экспортной партии, дата отправления на экспорт, наименование импортера, страна импортера, количество продукции, в партии.
10	Банк.
	Код клиента, юридическое наименование клиента, юридический адрес клиента, ФИО директора, телефон директора, ФИО главного бухгалтера, телефон бухгалтера, номер счета клиента, остаток на счете, номер счета дебитора, банк дебитора, юридическое наименование дебитора, номер счета кредитора, банк кредитора, юридическое наименование кредитора, номер договора о кредитовании, дата выдачи кредита, срок пользования кредитом, сумма кредита, процентная ставка.
11	Экспорт/импорт.
	Код предприятия, название предприятия, юридический адрес, телефон/факс, WWW, балансовый год, валовой доход, чистая прибыль, код товарной номенклатуры внешнеэкономической деятельности (ТНЗД), наименование ТНЗД, код группы производства (ГВ), наименования ГВ, год операции экспорта/импорта, объем экспорта/импорта, рейтинг предприятия.
12	Проект.
	Код работы, наименование работы, дата начала, самая ранняя дата начала, дата окончания, самая поздняя дата окончания, общий объем в человеко-часах, общая потребность в материалах в условных единицах, номер вехи, показатель объема выполненной работы в процентах к общему объему, показатель использованных материалов в процентах к общей потребности, код исполнителя, наименования исполнителя, юридический адрес исполнителя, номер недели с начала года, объем выполненной работы, количество использованных материалов.

2.2. Оформить отчет по результатам выполнения контрольного задания:
Отчет должен содержать:

1. Инфологическую модель БД (на языке таблица-связь).
2. Разработанные ограничения целостности.
3. Отчеты Data Dictionary Report, сгенерированные для каждой созданной при выполнении контрольного задания таблицы.
4. Отчеты Data Dictionary Report, сгенерированные для каждой таблицы БД BookShop.

Контрольные вопросы:

1. Сформулируйте основные свойства реляционной таблицы.
2. Что такое первичный ключ?
3. Что такое нормализация отношений? Дайте определение 1NF, 2NF, 3NF, NFBK.
4. Сформулируйте правила нормализации отношений.
5. Из каких файлов состоит база данных SQL Server?
6. Сформулируйте правила обеспечения целостности данных в реляционных СУБД.
7. Что такое инфологическая модель базы данных?
8. Что такое логическая модель базы данных?
9. Из каких стандартных объектов состоит модель базы данных SQL Server?
10. Что такое транзакция?
11. Что такое глобальный уникальный идентификатор?
12. Что такое декларативная ссылочная целостность?
13. Что такое экстенс? Какие типы экстенсов используются в SQL Server?
14. Почему в SQL Server используются несколько журналов транзакций, расположенных на разных физических дисках? Какова структура файла базы данных SQL Server?
15. Для чего и как определяются ограничения на значение столбцов?
16. Что такое ограничение первичного ключа?
17. Что такое ограничение внешнего ключа?
18. Что такое связывание?
19. Для чего предназначены правила? В чем разница между правилами и ограничениями на значение столбцов?
20. В чем разница между значением по умолчанию и установкой по умолчанию?
21. Что такое ниладические функции? Для чего они используются? Приведите пример.
22. Каков размер страницы в SQL Server?
23. Какие типы страниц поддерживает SQL Server ?
24. Для чего предназначены страницы GAM в файлах баз данных SQL Server? Какова их структура?
25. Что такое представление?
26. Что такое триггер?
27. Что такое ограничения целостности? Какие ограничения

целостности поддерживает PostgreSQL?

28. Для чего предназначена и в чем суть технологии MVCC?

29. Что такое WAL?

30. Что представляет собой технология TOAST? В чем особенности ее реализации

31. Какова структура страниц файла данных PostgreSQL?

32. В чем заключается концепция табличных пространств, используемая в PostgreSQL?

33. Что такое схемы в PostgreSQL? Для чего они предназначены?

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по выполнению Практических работ по дисциплине
«ОСНОВЫ ПОСТРОЕНИЯ ЗАЩИЩЕННЫХ БАЗ ДАННЫХ»
для студентов специальности 10.05.01 «Компьютерная безопасность»
специализации «Информационно-аналитическая и техническая
экспертиза компьютерных систем»

**Ставрополь,
2026**

Содержание

Практическая работа №1. Язык SQL. Выработка данных

Практическая работа №2. Определение данных и манипулирование данными

Практическая работа №3. Функции, определенные пользователем

Практическая работа №4. Создание программного приложения для работы с базой данных

Практическая работа №5. Проектирование реляционных баз данных на основе метода ER-диаграмм

Практическая работа №1. Язык SQL. Выработка данных

Цель работы: изучить язык запросов на выборку данных из БД.

Теоретическая часть

Знакомство с Query Tool pgAdmin

Для ввода и тестирования инструкций и операторов языка `plpgsql` можно использовать приложение Query Tool, входящее в комплект поставки PostgreSQL. Для запуска Query Tool необходимо выбрать одноименную команду из меню Tools командного меню pgAdmin или нажать комбинацию клавиш `Ctrl+E`. В открывшемся окне (рисунок 6.1) выбрать базу данных из списка и создать новый запрос или загрузить сохраненный ранее SQL-скрипт.

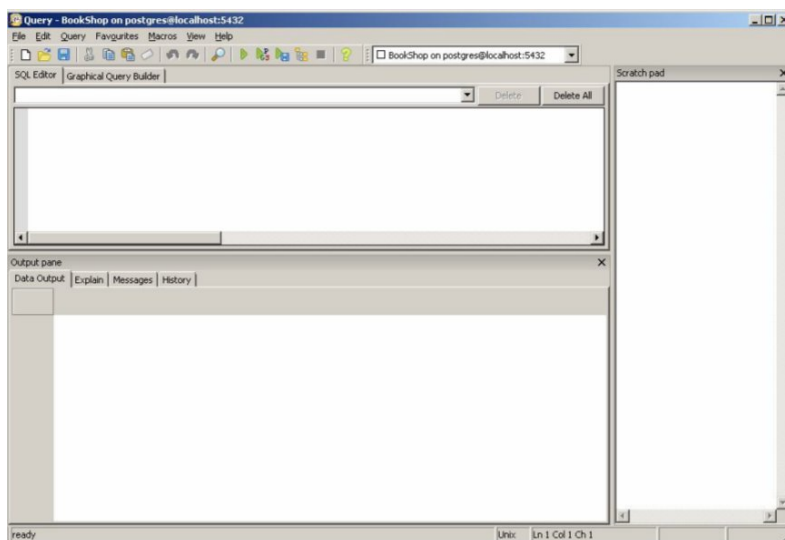


Рисунок 6.1 – Основное окно Query Tool

Query Tool – многооконное графическое средство, которое позволяет:

- вводить (New query, Load SQL script), редактировать, выполнять (Execute query), и сохранять (Save query/result) запросы и их результаты в текстовом (text) и графическом виде (image);
- визуально строить запросы (Graphical Query Builder);
- получать пояснения, как запросы анализируются, оптимизируются и выполняются, и представлять в графическом виде схему выполнения запроса (Explain query);
- получать статистическую информацию о запросе (Message);

- сохранять результаты выполнения запроса в файле (Export).

Синтаксис инструкции SELECT

Все запросы на получение практически любого количества данных из одной или нескольких таблиц выполняются с помощью единственного предложения SELECT. В общем случае результатом реализации предложения SELECT является другая таблица. К этой новой (рабочей) таблице может быть снова применена операция SELECT и т.д., т.е. такие операции могут быть вложены друг в друга. Представляет исторический интерес тот факт, что именно возможность включения одного предложения SELECT внутри другого послужила мотивировкой использования прилагательного «структурированный» в названии языка SQL.

Инструкция SELECT используется в основном как:

- самостоятельная команда на получение и вывод строк таблицы, сформированной из столбцов и строк одной или нескольких таблиц (представлений);
- элемент другого запроса, называемый «вложенный запрос»;
- фраза выбора в командах создания представления, курсора или вставки;
- средство присвоения переменным значений из строк сформированной таблицы.

В этой работе рассматриваются только два первых варианта использования SELECT и его синтаксис, ограниченный конструкциями, используемыми при реализации этих вариантов, а остальные – в последующих работах лабораторного практикума.

Здесь и далее в синтаксических конструкциях используются следующие обозначения:

- звездочка (*) для обозначения «все» – употребляется в обычном для программирования смысле, т.е. «все случаи, удовлетворяющие определению»;

- квадратные скобки ([]) – означают, что конструкции, заключенные в эти скобки, являются необязательными (т.е. могут быть опущены);
- фигурные скобки ({}) – означают, что конструкции, заключенные в эти скобки, должны рассматриваться как целые синтаксические единицы, т.е. они позволяют уточнить порядок разбора синтаксических конструкций, заменяя обычные скобки, используемые в синтаксисе SQL;
- многоточие (...) – указывает на то, что непосредственно предшествующая ему синтаксическая единица может повторяться один или более раз;
- прямая черта (|) – означает наличие выбора из двух или более возможностей. Например, обозначение ASC|DESC указывает, можно выбрать один из терминов ASC или DESC; когда же один из элементов выбора заключен в квадратные скобки, то это означает, что он выбирается по умолчанию (так, [ASC]|DESC означает, что отсутствие всей этой конструкции будет восприниматься как выбор ASC);
- точка с запятой (;) – завершающий элемент предложений SQL;
- запятая (,) – используется для разделения элементов списков;
- пробелы () – могут вводиться для повышения наглядности между любыми синтаксическими конструкциями предложений SQL;
- прописные латинские буквы и символы – используются для написания конструкций языка SQL и должны (если это специально не оговорено) записываться в точности так, как показано;
- строчные буквы – используются для написания конструкций, которые должны заменяться конкретными значениями, выбранными пользователем, причем для определенности

отдельные слова этих конструкций связываются между собой символом подчеркивания (_);

- термины *таблица*, *столбец*, ... – заменяют (с целью сокращения текста синтаксических конструкций) термины *имя_таблицы*, *имя_столбца*, ..., соответственно;
- термин *таблица* используется для обобщения таких видов таблиц, как *базовая_таблица*, *представление* или *псевдоним*; здесь *псевдоним* служит для временного (на момент выполнения запроса) переименования и (или) создания рабочей копии базовой таблицы или представления.

Инструкция **SELECT** (выбрать) имеет следующий формат (упрощенный):

```
SELECT [[ALL] | DISTINCT] { * | элемент_SELECT [[AS] [псевдоним_столбца]
                                     [,элемент_SELECT [AS] [псевдоним_столбца]] ...}
FROM      {базовая_таблица | представление} [AS] [псевдоним]
          [, {базовая_таблица | представление} [AS] [псевдоним]] ...
[WHERE     условие_where]
[GROUP BY  список_group_by]
[HAVING     условие_having]]
[ORDER BY  список_order_by]
[FOR UPDATE [ OF таблица [. ...]]] [ LIMIT { число | ALL } [ { OFFSET | . } начало ] ]
```

Различные элементы этой инструкции позволяют указать условия для выбора нужных данных и (если требуется) их обработки:

SELECT (выбрать) данные из указанных столбцов и (если необходимо) выполнить перед выводом их преобразование в

	соответствии с указанными выражениями и (или) функциями;
FROM	(из) перечисленных таблиц, в которых расположены эти столбцы;
WHERE	(где) строки из указанных таблиц должны удовлетворять указанному перечню условий отбора строк;
GROUP BY	(группируя по) указанному перечню столбцов с тем, чтобы получить для каждой группы единственное агрегированное значение, используя в списке элементов SELECT агрегирующие SQL-функции, например: SUM (сумма), COUNT (количество), MIN (минимальное значение), MAX (максимальное значение) или AVG (среднее значение);
HAVING	(имея) в результате лишь те группы, которые удовлетворяют указанному перечню условий отбора групп;
ORDER BY	(упорядочивая) результаты выбора данных; при этом упорядочение можно производить в порядке возрастания — ASC (ASCending) или убывания — DESC (DESCending), а по умолчанию принимается ASC;

Синтаксические конструкции, введенные выше, определяются следующим образом:

Элемент_SELECT ::= [таблица.]* | выражение

выражение ::= константа

| скалярная_функция

| [псевдоним.]столбец

| (выражение)

| (скалярный_подзапрос)

| {унарный оператор} выражение

| выражение {бинарный_оператор} выражение

где:

константа — одиночные алфавитно-цифровые символы, строки символов и числовые константы (символьные данные и данные типа дата/время берутся в одинарные кавычки);

скалярная_функция — SQL-функция, возвращающая скалярное значение, например SUM, CAST и др. (выражения, выступающие в качестве параметров SQL-функций, не должны содержать вложенных SQL-функций)

базовая_скалярная_функция ::=

| {SUM|AVG|MIN|MAX|COUNT} ([ALL|DISTINCT][таблица.]столбец)

| {SUM|AVG|MIN|MAX|COUNT} ([ALL] выражение)

| COUNT(*)

псевдоним — временный идентификатор, присваиваемый таблице с

помощью служебного слова AS в предложении FROM;

столбец – собственно имя столбца, но не четырехсоставное имя, состоящее из имени базы данных, владельца, таблицы и имени столбца;

скалярный_подзапрос – подзапрос, возвращающий скалярный результат; подзапрос – это инструкция SELECT, вложенная в:

- предложения WHERE, HAVING или SELECT другой инструкции SELECT;
- инструкции INSERT, UPDATE и DELETE, предназначенные для модификации данных в таблицах;
- другой подзапрос.

Предложение WHERE включает набор условий для отбора строк:

условие_where ::= [NOT] условие [[AND|OR][NOT] условие]...

условие ::= выражение { = | <> | < | <= | > | >= } { выражение | (подзапрос) }
выражение_1 [NOT] BETWEEN выражение_2 AND выражение_3
выражение [NOT] IN { (константа [, константа] ...) | (подзапрос) }
выражение IS [NOT] NULL
[таблица.]столбец [NOT] LIKE 'строка_символов' [ESCAPE 'символ']
EXISTS (подзапрос)

Кроме традиционных операторов сравнения (= | <> | < | <= | > | >=) в предложении WHERE используются условия BETWEEN (между), LIKE (похоже на), IN (принадлежит), IS NULL (не определено) и EXISTS (проверка на существование данных, удовлетворяющих критериям подзапроса; если данных нет, то результат – ложь), которые могут предваряться оператором NOT (не). Критерий отбора строк формируется из одного или нескольких условий, соединенных логическими операторами AND, OR, NOT (с учетом их приоритета и скобок).

При обработке условия числа сравниваются алгебраически – отрицательные числа считаются меньшими, чем положительные, независимо от их абсолютной величины. Строки символов сравниваются в соответствии с их представлением в коде, используемом в конкретной СУБД, например, в коде ASCII. Если сравниваются две строки символов, имеющих разные длины, более короткая строка дополняется справа пробелами для того, чтобы

они имели одинаковую длину перед осуществлением сравнения.

Синтаксис модификатора GROUP BY имеет вид:

GROUP BY [таблица.]столбец [, [таблица.]столбец] ... [HAVING условие_having]

GROUP BY инициирует перекомпоновку формируемой таблицы по группам, каждая из которых имеет одинаковое значение в столбцах, включенных в перечень GROUP BY. Далее к этим группам применяются агрегирующие функции, указанные в списке выбора SELECT, что приводит к замене всех значений группы на единственное значение (сумма, количество и т.п.). В результате выполнения GROUP BY остаются только уникальные значения столбцов, по умолчанию отсортированные по возрастанию.

С помощью предложения HAVING, где условие_having определяется так же, как и условие_where, можно исключить из результата группы, не удовлетворяющие заданным условиям.

Список параметров модификатора ORDER BY имеет вид

список_order_by ::=
 {{[таблица.]столбец | номер_элемента_SELECT} [[ASC] |
 DESC]

В отличие от GROUP BY модификатор ORDER BY не удаляет повторяющиеся значения.

Выборка без использования предложения WHERE

Простая выборка данных

По запросу

SELECT “Название”, “Адрес”

FROM “Поставщики”

будут выданы название и адрес поставщиков из таблицы Поставщики.

При необходимости получения полной информации о поставщиках запрос будет таким:

SELECT “Код_поставщика”, “Название”, “Город”, “Адрес”, “Телефон”

FROM “Поставщики” или можно заменить список столбцов символом «*», который служит кратким обозначением всех имен полей в таблице, указанной во фразе FROM, причем порядок вывода полей соответствует

порядку, в котором эти поля определялись при создании таблицы.

Для исключения дубликатов и одновременного упорядочения перечня необходимо дополнить запрос ключевым словом DISTINCT (различный, различные). Если после DISTINCT указывается несколько столбцов, то SELECT выдаст строки, уникальные для указанного набора столбцов. Например, по запросу

```
SELECT DISTINCT "Автор", "Название"  
FROM "Книги"
```

будут выданы все уникальные значения пары автор-название из таблицы Книги.

Выборка с использованием выражений

Из синтаксиса фразы SELECT видно, что в ней может содержаться не только перечень столбцов таблицы или символ «*», но и выражения. Следует иметь в виду, что вычисляемые столбцы не сохраняются в БД, а формируются во временной таблице, создаваемой при выполнении инструкции SELECT. Например, чтобы выдать список всех книг, их цен, размер 10% надбавки к ценам и новые цены, надо ввести запрос

```
SELECT "Код_книги", "Цена", Цена*0.1, Цена*1.1  
FROM "Книги"
```

В результате выполнения приведенного запроса, при выводе результата каждый столбец выводится с заголовком, совпадающим с его именем в таблице, а вычисляемые столбцы – с заголовком (No column name). Эти заголовки можно изменять с помощью псевдонимов. Например, в результате выполнения запроса

```
SELECT "Код_книги", "Цена" AS "Старая цена", "Цена"*0.1 AS  
"Надбавка", "Цена"*1.1 AS "Новая цена", "Остаток" AS "Имеется в наличии"  
FROM "Книги"
```

будут выведены значения столбца Цена под псевдонимом Старая_цена, столбца Остаток под псевдонимом Имеется в наличии и значения двух вычисляемых столбцов под псевдонимами Надбавка и Новая цена.

Выражения могут состоять из отдельных числовых или текстовых констант, причем последние должны заключаться в одинарные кавычки. Текстовые константы обычно используются в виде пояснений к данным, выводимым в результате выполнения запроса, например:

```
SELECT DISTINCT 'Книга', "Код_книги", 'Цена', "Цена", 'uah'  
FROM "Книги"
```

Выборка NULL-значений

Если в какой-либо из вводимых строк таблицы отсутствует значение для какого-либо столбца и для этого столбца допустимы NULL-значения, то СУБД введет в такое поле NULL-значение. NULL-значение используется для того, чтобы представить единым образом "неизвестные значения" для любых типов данных. При выводе NULL-значения на экран или печатающее устройство его код воспроизводится каким-либо специально заданным символом или набором символов, например, < NULL > или сочетанием -0-.

Как правило, в современных СУБД с помощью специальной команды можно установить один из режимов представления NULL-значений при выполнении числовых расчетов: запрет или разрешение замены NULL-значения нулем. В первом случае любое арифметическое выражение, содержащее неопределенный операнд, будет также иметь неопределенное значение (обычно устанавливается по умолчанию). Во втором случае результат вычислений будет иметь численное значение (если это значение попадает в диапазон представления соответствующего типа данных). Для проверки текущих установок СУБД введите в таблицу Книги строку с нулевым значением столбца Цена и затем – запрос:

```
SELECT "Остаток" + 100 FROM "Книги"
```

Поскольку в SQL все NULL-значения рассматриваются обычно как равные (предикат NULL=NULL имеет значение «истина»), два неопределенных значения рассматриваются как дубликаты друг друга при исключении дубликатов и поэтому предложение SELECT DISTINCT даст в результате не более одного NULL-значения. Введите в таблицу Книги еще

одну строку с нулевым значением столбца Цена и обратите внимание, сколько различных значений будут выведены при выполнении запроса:

```
SELECT DISTINCT    "Цена"
```

```
FROM "Книги"
```

Выборка с использованием предложения WHERE

Допустим, нас интересует OID БД BookShop. Его можно получить из системной таблицы pg_database по запросу, в котором указано имя интересующей нас БД:

```
SELECT OID, datname FROM pg_database
```

```
WHERE datname = 'BookShop'
```

Для отбора нужных строк таблицы в предложении WHERE можно использовать операторы сравнения, которые могут предваряться оператором NOT, создавая, например, отношения "не меньше" и "не больше", а также возможно использование нескольких условий, соединенных логическими операторами AND, OR, AND NOT и OR NOT, что позволяет осуществить более детальный отбор строк. Например, чтобы получить список книг из БД BookShop, поставленных за последний месяц, вводим запрос:

```
SELECT "Код_книги", "Количество", "Дата"
```

```
FROM "Поставки"
```

```
WHERE ("Дата" >= '1/9/2011') AND ("Дата" < '1/10/2011')
```

Если в БД нет строк, удовлетворяющих заданному(ым) условию(ям), на экран ничего не будет выводиться. Следует иметь в виду, что в SQL ситуация, когда запрос ничего не возвращается, не является ошибочной, поэтому контроль за корректностью вводимого запроса и результатах его выполнения должен осуществлять пользователь.

Использование BETWEEN

С помощью BETWEEN ... AND ... (находится в интервале от ... до ...) можно отобрать строки, в которых значение какого-либо столбца находится в заданном диапазоне. Например, предыдущий запрос можно переписать так:

```
SELECT "Код_книги", "Количество", "Дата"
```

FROM "Поставки"

WHERE "Дата" BETWEEN '1/9/2011' AND '1/10/2011'

Можно задать и NOT BETWEEN, если необходимо указать на данные, не принадлежащие некоторому диапазону. Оператор BETWEEN можно заменить предикатом, но он улучшает восприятие и читаемость текстов запросов SQL.

Использование IN

В запросе SELECT *

FROM "Поставщики"

WHERE "Город" IN ('Киев', 'С-Петербург', 'Петербург') оператор IN является в действительности просто краткой записью последовательности отдельных сравнений, соединенных операторами OR. Предыдущее предложение эквивалентно, очевидно, такому:

SELECT *

FROM "Поставщики"

WHERE "Город"='Киев' OR "Город"='С-Петербург' OR "Город"='С-Петербург'

Можно также вводить NOT IN (не принадлежит) и использовать вместо списка IN (NOT IN) подзапрос.

Использование LIKE

Обычная форма столбец LIKE текстовая_константа для столбца текстового типа позволяет отыскать все значения указанного столбца, соответствующие образцу, заданному текстовой_константой. Символы этой константы интерпретируются следующим образом:

- символ «_» заменяет любой одиночный символ;
- символ «%» заменяет любую последовательность из 0 или более символов,
- все другие символы означают просто сами себя.

Например, чтобы получить список книг, в названии которых есть подстрока «С++», надо сформировать запрос:

```
SELECT "Код_книги", "Название", "Автор"  
FROM "Книги"  
WHERE "Название" LIKE "%C++%"
```

Выборка NULL-значений

Если при загрузке данных не введено значение в какое-либо поле таблицы, то СУБД поместит в него NULL-значение. Обнаружить эти значения можно с помощью условий

столбец IS NULL или столбец IS NOT NULL. Например, запрос, приведенный ниже, возвратит строки в таблице Поставщики, в которых отсутствуют сведения в указанных столбцах:

```
SELECT *  
FROM "Поставщики"  
WHERE "Адрес" IS NULL OR "Телефон" IS NULL
```

Ограничение максимального размера выборки

В PostgreSQL есть специфические конструкции LIMIT и OFFSET, с помощью которых можно управлять количеством и порядком вывода записей. LIMIT ограничивает максимальное количества возвращаемых записей, а OFFSET задает номер записи в курсоре (0,1,2, ...), начиная с которой будут выводиться записи. Они имеют следующий синтаксис:

LIMIT { число | ALL } [OFFSET начало] где число – количество записей, ALL – все записи, начало – номер первой записи. Например, если число = 10 и начало = 3, то запрос вернет 10 записей, начиная с четвертой (номер 3) по порядку.

Использование данных типа timestamp

Данные типа timestamp используются в предложении WHERE для предотвращения одновременного обновления одной и той же строки двумя пользователями. Если при изменении значений данной строки поле типа timestamp также изменяется, то при одновременном редактировании строки двумя пользователями при сохранении одним из них обновленной информации поле timestamp изменится, и строка не будет соответствовать

прежней копии. Когда другой пользователь попытается сохранить свои изменения, предложение WHERE не даст ему этого сделать, поскольку команда не найдет выбранную ранее строку. Данные типа timestamp – это хорошее средство для поддержания сервером уникальности каждой строки.

Выборка с упорядочением

Простейший вариант использования модификатора ORDER BY – упорядочение строк результата по значению одного из столбцов с указанием порядка сортировки или без такого указания (по умолчанию строки будут сортироваться в порядке возрастания значений в указанном столбце.). Например, выдать перечень книг из таблицы Книги в алфавитном порядке по фамилиям авторов:

```
SELECT "Код_книги", "Автор", "Название"  
FROM "Книги"  
ORDER BY "Автор"
```

или в порядке убывания цены:

```
SELECT "Код_книги", "Цена"  
FROM "Книги"  
ORDER BY "Цена" DESC
```

При включении в список ORDER BY нескольких столбцов СУБД сортирует строки результата по значениям первого столбца списка пока не появится несколько строк с одинаковыми значениями данных в этом столбце. Такие строки сортируются по значениям следующего столбца из списка ORDER BY и т.д. Например, выдать содержимое таблицы Заказы, отсортировав ее строки по столбцам Код_заказчика и Дата:

```
SELECT *  
FROM "Заказы"  
ORDER BY "Код_заказчика", "Дата"
```

Кроме того, в список ORDER BY можно включать не только имя столбца, но и его порядковую позицию в перечне SELECT. Номер столбца в списке элементов SELECT отсчитывается слева направо. Благодаря этому

возможно упорядочение результатов на основе вычисляемых столбцов, не имеющих имен, например:

```
SELECT "Код_книги", "Цена" * 0.2 FROM "Книги"  
ORDER BY 2 DESC
```

В списке выбора инструкции SELECT столбцы можно задавать различными способами, не учитывая порядок их следования в базовой таблице. Можно даже один и тот же столбец указывать несколько раз, например:

```
SELECT "Код_книги", "Автор", "Название", "Издательство", "Цена",  
"Код_книги"  
FROM "Книги"  
ORDER BY 2, 3
```

Выдаваемые по этому запросу строки не помещаются полностью на экране, и поэтому при горизонтальной прокрутке первые столбцы выходят за пределы активного окна. Выводимый повторно в конце списка Код_книги облегчит работу с таким списком.

Выборка и агрегирование данных

Для выполнения объединения данных по некоторому критерию используется модификатор GROUP BY, который делит таблицу на группы строк, каждая из которых имеет одинаковые значения в столбце, указанном в GROUP BY.

Пусть необходимо узнать, какие поставщики осуществляли поставки. Для этого можно, очевидно, ввести запрос

```
SELECT DISTINCT "Код_поставщика"  
FROM "Поставки"
```

```
То же самое получим с помощью запроса SELECT "Код_поставщика"  
FROM "Поставки"  
GROUP BY "Код_поставщика"
```

Однако, если теперь в список SELECT добавить еще одно поле, например, Код_книги, то получим сообщение об ошибке:

ERROR: column "Поставки.Код_книги" must appear in the GROUP BY clause or be used in an aggregate function

т.е. поле Код_книги должно быть указано в строке GROUP BY или использоваться в агрегирующей функции. Это связано с тем, что SQL-функция создает единственное значение из множества значений столбца-аргумента, а для «свободного» столбца должно быть выдано все множество его значений. Поэтому подобный запрос отвергается системой.

Если добавить Код_книги в строку GROUP BY, то в группах с одинаковым значением Код_поставщика данные дополнительно будут сгруппированы по Код_книги.

Сама по себе группировка не имеет смысла – группировка (или агрегирование) обычно предшествует неким операциям, выполняемым над данными в группах. Такие операции вводятся с помощью агрегирующих SQL-функций: SUM – сумма, AVG – среднее, MIN – минимальное значение, MAX – максимальное значение, COUNT – количество.

Возвращаясь к последнему примеру, предположим, что необходимо получить данные о количестве книг, поставленных каждым поставщиком в отдельности. Это можно сделать с помощью запроса:

```
SELECT "Код_поставщика", "Код_книги", SUM("Количество") AS  
"Количество"  
FROM "Поставки"  
GROUP BY "Код_поставщика ", "Код_книги"
```

Если GROUP BY не используется, то при наличии в списке SELECT агрегирующих функций в этот список можно включать лишь SQL-функции или выражения, содержащие такие функции.

В столбце-аргументе перед применением любой функции, кроме COUNT(*), исключаются все неопределенные значения. Если оказывается, что аргумент – пустое множество, функция COUNT принимает значение 0, а остальные – NULL.

Предположим, что теперь требуется узнать, сколько книг каждого

издательства представлено в базе данных BOOKSHOP. Для этого введем запрос:

```
SELECT "Издательство", COUNT(*)  
FROM "Книги"  
GROUP BY "Издательство"
```

Строки с одинаковыми значениями столбца Издательство объединяются сначала в группы и для каждой группы выводится только одна строка, во втором столбце которой выводится результат выполнения функции COUNT для данной группы, т.е. количество строк в ней.

Т.о., GROUP BY инициирует перекомпоновку указанной таблицы по группам, далее к каждой группе применяется инструкция SELECT. Каждое выражение в списке SELECT должно принимать единственное значение для группы, т.е. оно может быть либо значением столбца, указанного в GROUP BY, либо арифметическим выражением, включающим это значение, либо константой, либо одной из скалярных SQL-функций, которая оперирует всеми значениями столбца в группе и сводит эти значения к единственному значению (например, к сумме, среднему значению и т.д.).

Модификатор GROUP BY не предполагает упорядочивание результата. Чтобы упорядочить результаты запроса, в конце инструкции надо добавить модификатор ORDER BY:

```
SELECT "Код_поставщика", "Код_книги", SUM("Количество") AS  
"Количество"  
FROM "Поставки"  
GROUP BY "Код_поставщика ", "Код_книги"  
ORDER BY 1, 3
```

Строки таблицы можно группировать по любой комбинации ее столбцов. Так, в процессе выполнения запроса:

```
SELECT "Код_заказчика", "Оплачен", COUNT(*) AS "Количество  
заказов" FROM "Заказы"  
GROUP BY "Код_заказчика", "Оплачен"
```

ORDER BY 1

сначала выполняется группировка по значениям столбца Код_заказчика, затем в каждой из полученных подгрупп выполняется группировка по значениям столбца Оплачен и применяется агрегирующая функция COUNT, т.е. вычисляет для каждого заказчика количество оплаченных и неоплаченных заказов.

Если в запросе используются WHERE и GROUP BY, то строки, не удовлетворяющие условию WHERE, исключаются до выполнения группирования. Например, предыдущий запрос можно модифицировать так, чтобы он выдавал список заказчиков, оплативших заказы, и количество оплаченных ими заказов:

```
SELECT "Код_заказчика", "Оплачен", COUNT(*) "Количество заказов"
FROM "Заказы"
WHERE "Оплачен" = 'ДА'
GROUP BY "Код_заказчика", "Оплачен"
ORDER BY 1
```

Предложение HAVING играет такую же роль для групп, что и WHERE для строк: она используется для исключения групп, точно так же, как WHERE используется для исключения строк. Эта фраза включается в предложение лишь при наличии фразы GROUP BY, а выражение в HAVING должно принимать единственное значение для группы. Например, выдать список поставщиков, поставивших более двух книг:

```
SELECT "Код_поставщика", COUNT(*) "Количество поставленных
книг" FROM "Книги"
GROUP BY "Код_поставщика"
HAVING COUNT(*) > 2
```

Преобразование типов данных в инструкции SELECT

В PostgreSQL существует четыре фундаментальные SQL-конструкции, требующие четких правил преобразования типов: вызовы функций, применение операторов, присваивание значений при вставке и

модифицировании данных, применение конструкций UNION (объединение результатов запросов) и CASE (аналог оператора if..then..else).

В PostgreSQL поддерживаются три варианта синтаксиса явного преобразования (приведения) типов:

- для строковых констант тип 'значение'

'значение'::тип

CAST ('значение' AS тип)

- для числовых констант значение:: тип

CAST (значение AS тип)

- для полей набора данных, возвращаемых запросом SQL
идентификатор:: тип

CAST (идентификатор AS тип)

Пример:

SELECT ("Остаток"+10)::float * 1.1,

"Цена"::char(8) || char(4) ' UAH', — *конкатенация двух строк*

CAST("Цена" AS text)

FROM "Книги"

Следующий пример демонстрирует использование функций преобразования типов для построения строкового выражения:

SELECT 'От ' || '01/01/2011' || ' до ' || CAST(now() AS varchar(64)) || '
прошло ' || (CAST (now() AS varchar(64)) :: date – date '01/01/2011') :: text || '
дней'

Возможно также использование преобразования типов в предложении WHERE. Следующий запрос возвратит данные об январских поставках в базе данных BookShop:

SELECT *

FROM "Поставки"

WHERE CAST("Дата" AS varchar) LIKE '%2011-01-%'

Тот же результат получается с помощью стандартной функции extract:

SELECT *

FROM "Поставки"

WHERE extract(month from "Дата") = 1 AND extract(year from "Дата") = 2011 Кроме синтаксических форм преобразования типов существуют некоторые функции, позволяющие добиться практически того же результата. Имена этих функций часто совпадают с именами итоговых типов, например, text() или timeofday().

Использование SELECT для соединения двух и более таблиц

Декартово произведение таблиц

Соединения таблиц – это подмножества их декартова произведения. Так как декартово произведение n таблиц – это таблица, содержащая все возможные строки r , такие, что r является сцеплением какой-либо строки из первой таблицы, строки из второй таблицы, ..., строки из n -й таблицы (с помощью SELECT можно получить любое подмножество реляционной таблицы), то осталось лишь выяснить, можно ли с помощью SELECT получить декартово произведение. Для получения декартова произведения нескольких таблиц надо указать во фразе FROM перечень перемножаемых таблиц, а во фразе SELECT – все их столбцы:

```
SELECT "Заказы".*, "Заказчики".*
```

```
FROM "Заказы", "Заказчики"
```

В результате SQL выдаст комбинацию каждой строки из таблицы Заказы со всеми строками таблицы Заказчики, т.е. количество строк в запросе равно произведению количества строк обеих таблицы. Большинство этих строк не имеют никакого смысла.

Эквисоединение таблиц

Если из декартова произведения убрать ненужные строки и столбцы, то можно получить вполне осмысленный результат, соответствующий любому из соединений. Например, если к предыдущему запросу добавить предложение WHERE:

```
SELECT "Заказы".*, "Заказчики".*
```

```
FROM "Заказы", "Заказчики"
```

WHERE “Заказы”.”Код_заказчика” = “Заказчики”.”Код_заказчика” то получим таблицу – эквисоединение двух таблиц, в которой останутся только строки с одинаковыми значениями в столбцах Код_заказчика, но таких столбцов будет два и они будут дублировать друг друга.

Естественное соединение таблиц

Если из эквисоединения таблиц исключить один из столбцов-дубликатов, по которым проводилось соединение, то получим естественное соединение тех же таблиц. Это можно сделать, изменив предыдущий запрос следующим образом:

```
SELECT “Код_заказа”, “Код_книги”, “Дата”, “Оплачен”, “Заказчики”.*  
FROM “Заказы”, “Заказчики”  
WHERE “Заказы”.”Код_заказчика” = “Заказчики”.”Код_заказчика”
```

Композиция таблиц

Если исключить все столбцы, по которым проводится соединение таблиц, то получим их композицию:

```
SELECT "Код_заказа", "Код_книги", "Дата", "Оплачен", "Имя",  
"Адрес", "Телефон"  
FROM "Заказы", "Заказчики"  
WHERE "Заказы".”Код_заказчика” = "Заказчики".”Код_заказчика”
```

При формировании соединения создается рабочая таблица, к которой применимы все операции, рассмотренные ранее: отбор нужных строк соединения (WHERE), упорядочение получаемого результата (ORDER BY) и агрегатирование данных (SQL-функции и GROUP BY). Например, следующий запрос позволит произвести выборку всех оплаченных заказов за период времени между двумя датами, упорядоченных по Код_книги:

```
SELECT "Код_заказа", "Код_книги", "Дата", "Оплачен",  
"Код_заказчика" FROM "Заказы", "Заказчики"  
WHERE ("Заказы".”Код_заказчика” = "Заказчики".”Код_заказчика”)  
AND ("Дата" BETWEEN ‘03/01/2001’ AND ‘04/30/2001’)  
AND ("Оплачен" LIKE ‘%да%’)
```

ORDER BY "Код_книги"

Соединение таблицы со своей копией

В ряде приложений возникает необходимость одновременной обработки данных какой-либо таблицы и одной или нескольких ее копий, создаваемых на время выполнения запроса. Например, при добавлении записи в таблицу Заказчики возможен повторный ввод данных о каком-либо заказчике с присвоением ему другого кода. Для выявления таких ошибок можно соединить таблицу Заказчики с ее временной копией, установив в предложении WHERE равенство значений всех одноименных столбцов этих таблиц, кроме столбцов с кодом заказчика (для последних надо установить условие неравенства значений). Временные копии таблицы можно сформировать с помощью псевдонимов, указав их за именем таблицы в предложении FROM. Так, если ввести

FROM "Заказчики" AS A, "Заказчики" AS "B" или просто

FROM "Заказчики" A, "Заказчики" B то будут сформированы две копии таблицы Заказчики с именами A, B. Тогда, чтобы узнать, есть ли в базе данных два разных заказчика с одинаковыми телефонами, надо ввести запрос:

SELECT A.*, B.*

FROM "Заказчики" A, "Заказчики" B WHERE A."Телефон" =
B."Телефон"

AND A."Код_заказчика" != B."Код_заказчика"

Вложенные подзапросы

Вложенный подзапрос – это подзапрос, заключенный в круглые скобки и вложенный в предложение WHERE (HAVING) инструкции SELECT или других инструкций, использующих предложение WHERE. Вложенный подзапрос может содержать в своем предложении WHERE (HAVING) другой вложенный подзапрос и т.д. Назначение вложенных подзапросов состоит в том, чтобы при отборе строк таблицы, сформированной основным запросом, можно было использовать данные из других.

Существуют простые и коррелированные вложенные подзапросы.

Простые вложенные подзапросы обрабатываются системой "снизу-вверх". Первым обрабатывается вложенный подзапрос самого нижнего уровня. Множество значений, полученное в результате его выполнения, используется при реализации подзапроса более высокого уровня и т.д.

Коррелированные вложенные подзапросы обрабатываются системой в обратном порядке. Сначала выбирается первая строка рабочей таблицы, сформированной основным запросом, и из нее выбираются значения тех столбцов, которые используются во вложенном подзапросе (вложенных подзапросах). Если эти значения удовлетворяют условиям вложенного подзапроса, то выбранная строка включается в результат. Затем выбирается вторая строка и т.д., пока в результат не будут включены все строки, удовлетворяющие вложенному подзапросу (последовательности вложенных подзапросов).

Подзапросы включаются в предложение WHERE (HAVING) с помощью ключевых слов IN, EXISTS или одного из условий сравнения (= | <> | < | <= | > | >=):

- подзапросы, не возвращающие ни одного или возвращающие несколько значений, начинаются с IN или операторов сравнения и содержат ключевые слова ANY или ALL;
- подзапросы, возвращающие единственное значение, начинаются с оператора сравнения;
- подзапросы, представляющие собой тест на существование или присутствие данных, начинаются с EXISTS.

Простые вложенные подзапросы

Простые вложенные подзапросы используются для представления множества значений, исследование которых должно осуществляться в каком-либо предикате IN, например:

```
SELECT *  
FROM "Поставщики"
```

```
WHERE "Код_поставщика" IN
(SELECT DISTINCT "Код_поставщика"
FROM "Книги"
WHERE "Автор" LIKE '%Дейт%')
```

Система выполняет прежде всего вложенный подзапрос. Этот подзапрос выдает множество уникальных кодов поставщиков, которые поставили книги указанного автора. Затем будет выполнен внешний запрос, который выведет данные об этих поставщиках.

Тот же результат можно получить с помощью соединения:

```
SELECT "Поставщики".*
FROM "Книги", "Поставщики"
WHERE "Книги"."Код_поставщика" =
"Поставщики"."Код_поставщика"
AND "Автор" LIKE '%Дейт%'
```

При выполнении последнего запроса система должна одновременно обрабатывать данные из двух таблиц, тогда как в предыдущем примере эти таблицы обрабатываются поочередно. Естественно, что для их реализации требуются различные ресурсы памяти и времени.

В подобных запросах можно использовать не только оператор IN, но и операторы сравнения (<>, <=, <, >= или >), однако, если вложенный подзапрос возвращает более одного значения и не используется оператор IN, будет возникать ошибка.

Коррелированные вложенные подзапросы

Рассмотрим запрос:

```
SELECT *
FROM "Заказы"
WHERE "Некто" IN
(SELECT "Имя"
FROM "Заказчики"
WHERE "Код_заказчика" = "Заказы"."Код_заказчика");
```

Такой подзапрос отличается тем, что вложенный подзапрос не может быть обработан прежде, чем будет обрабатываться внешний подзапрос, поскольку вложенный подзапрос зависит от значения Заказы. Код_заказч и ка, а оно изменяется по мере того, как система проверяет различные строки таблицы Заказы. Обработка этого подзапроса осуществляется следующим образом:

1. Система проверяет первую строку таблицы Заказы. Предположим, что в этой строке Код_заказчика имеет значение 1. Тогда значение Заказы. Код_заказч и ка в данный момент также примет значение 1 и система сможет обработать внутренний запрос (SELECT Имя FROM Заказчики WHERE Код_заказчика = 1) получив в результате некоторое подмножество значений поля Имя. Выборка значений для этого заказчика будет проведена тогда и только тогда, когда значение 'Некто' будет принадлежать этому множеству. На этом завершается обработка для заказчика с кодом 1.
2. Далее выбирается вторая строка таблицы Заказы и выполняется аналогичная обработка для следующего заказчика и т.д. до тех пор, пока не будут рассмотрены все строки таблицы Заказы. Подобные подзапросы называются коррелированными, так как их результат зависит от значений, определенных во внешнем подзапросе. Обработка коррелированного подзапроса, следовательно, должна повторяться последовательно для каждого значения извлекаемого из внешнего подзапроса.

Следующий пример демонстрирует использование одной и той же таблицы во внешнем подзапросе и коррелированном вложенном подзапросе:

```
SELECT "Код_книги", "Автор", "Название", "Издательство"  
FROM "Книги" A WHERE "Название" IN  
(SELECT DISTINCT "Название"  
FROM "Книги" B  
WHERE A."Издательство" != B."Издательство" )  
ORDER BY "Автор"
```

В этом запросе поочередно для каждой строки таблицы Книги

выбирается значение поля Издательство и для этого значения выполняется вложенный подзапрос, результатом которого является список уникальных названий книг, изданных другими издательствами. Текущая строка таблицы Книги включается в результат запроса, если значение в поле Название входит в список, сформированный вложенным запросом. Т.о., будут выданы данные о книгах, изданных более чем в одном издательстве, и результирующий список будет отсортирован по полю Автор.

Запросы, использующие EXISTS в предложении WHERE

В языке SQL подзапросы, выполняющие проверку на существование, представляются выражением

WHERE [NOT] EXISTS (подзапрос).

Такое выражение считается истинным только тогда, когда подзапрос возвращает непустое множество, т.е. когда существует какая-либо запись в таблице, указанной в предложении FROM подзапроса, которая удовлетворяет условию WHERE подзапроса (практически этот подзапрос всегда будет коррелированным множеством).

Рассмотрим пример запроса на выдачу названия поставщиков, поставляющих книги некоторого издательства, скажем BHV:

```
SELECT "Название"
FROM "Поставщики" A WHERE EXISTS (
SELECT *
FROM "Книги"
WHERE "Код_поставщика" = A."Код_поставщика"
AND "Издательство" = 'BHV')
```

Система последовательно выбирает строки таблицы Поставщики, выделяет из них значения столбца Название, а затем проверяет, существует ли в таблице Книги хотя бы одна строка со значением 'BHV' в столбце Издательство и значением в столбце Код_поставщика, равным значению того же столбца, выбранному из таблицы Поставщики. Если условие выполняется, то полученное значение столбца Название включается в результат.

Подзапросы с ANY и ALL

ANY (SOME) и ALL отличаются от EXISTS тем, что используются совместно с реляционными операторами. В этом отношении они подобны оператору IN, однако, в отличие от IN, они могут использоваться только с подзапросами.

Оператор ANY приводит к тому, что текущая строка внешнего запроса включается в результат, если любое (ANY) из значений, выведенных подзапросом, совпадает со значением в текущей строке внешнего запроса. Следующий запрос SELECT *

```
FROM "Заказчики"
```

```
WHERE "Код_заказчика" = ANY (SELECT "Код_заказчика" FROM  
"Заказы")
```

 равнозначен запросу SELECT *

```
FROM Заказчики
```

```
WHERE Код_заказчика IN (SELECT Код_заказчика FROM Заказы);
```

Отличие заключается в том, что оператор ANY может использовать другие операторы сравнения, а не только оператор «=» как в случае IN. Например, если в предыдущем запросе в предложении WHERE заменить оператор «=» на оператор «<=», то получим информацию о заказчике с минимальным значением в поле Код_заказчика, поскольку это – единственное значение, для которого условие основного запроса выполняется.

Оператор ALL аналогичен оператору ANY по взаимодействию с внешним запросом, но для того, чтобы текущее значение во внешнем запросе было выбрано, оно должно удовлетворять условию в предикате внешнего запроса для каждого значения, выбранного подзапросом.

В SQL сказать, что значение больше (или меньше), чем любое (ANY) из набора значений, – тоже самое что сказать, что оно больше (или меньше) чем любое одно отдельное из этих значений. И наоборот, сказать, что значение не равно всему (ALL) набору значений, равносильно тому, что нет такого значения в наборе которому оно было бы равно.

Любой запрос, который может быть сформулирован с ANY или с ALL, может быть также сформулирован с EXISTS, хотя обратное в общем случае неверно. Строго говоря, вариант с EXISTS не абсолютно идентичен вариантам с ANY или с ALL из-за различия в том, как обрабатываются NULL-значения. Всякий раз, когда допустимый подзапрос возвращает пустое множество, предикат с ALL автоматически верен, а с ANY – автоматически неверен.

Использование WITH

Конструкция WITH позволяет оформить результаты подзапроса в виде временной таблицы, которая создается на время выполнения запроса. Т.о., WITH может использоваться для структурирования сложных запросов с многоуровневой иерархией подзапросов. Синтаксис конструкции WITH достаточно прост:

WITH идентификатор AS (запрос)

Например,

WITH subquery_ as (

SELECT "Название", "Цена"

FROM "Книги"

WHERE 'Издательство' = 'BHV')

SELECT *

FROM _subquery_

WHERE "Цена" >= ANY(SELECT "Цена" FROM _subquery_)

Объединение двух или более запросов

Предложение UNION позволяет объединить результаты выполнения нескольких запросов к разным таблицам в одном подмножестве данных, реализуя один запрос. Синтаксис UNION имеет вид: запрос_1 UNION запрос_1

Непременное требование – одинаковые количество и типы полей в объединяемых предложением UNION запросах, хотя второе требование можно обойти, если использовать преобразование типов в запросах.

Например:

```
SELECT "Имя", "Адрес"  
FROM "Заказчики"  
  
UNION  
  
SELECT "Название", "Адрес"  
FROM "Поставщики"
```

Следует иметь в виду, что:

- избыточные дубликаты всегда исключаются из результата UNION;
- предложением с UNION можно объединить любое число таблиц (проекций таблиц);
- весь запрос может использовать только одно предложение ORDER BY. Оно применяется ко всему результату и должно находиться после последней инструкции SELECT.

Оборудование и материалы:

Для изучения материалов этой лабораторной работы необходимо:

1. Компьютер под управлением операционной системы Windows 8/10 с параметрами по умолчанию.
2. Установленные на компьютер программы Microsoft Visio, PostgreSQL, MySQL, VMWare/VirtualBox.
3. Текстовый процессор Microsoft Word 2010/2013/2016.

Указания по технике безопасности:

К выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж по технике безопасности.

Задания:

1. Напишите запрос для вывода названия, автора и цену книг из таблицы Книги.
2. Напишите запрос для вывода всех уникальных имен авторов книг в текущем порядке из таблицы Книги.

3. Напишите запрос для вывода всех заказов из таблицы Заказы со значениями суммы предоплаты выше 1,000 рублей.
4. Напишите запрос для вывода всех заказов из таблицы Заказы, для которых сумма предоплаты составляет не менее 30% от стоимости книги.
5. Напишите запрос к таблице Заказы для вывода всех оплаченных заказов, соответствующих заказчику с именем «Иванов И.».
6. Напишите запрос для вывода всех строк из таблицы Заказчики, относящихся к заказчикам, проживающим в г.Бресте.
7. Напишите запрос для вывода кодов всех книг, их цен и кодов поставщиков для книг, поставленных из Москвы.
8. Что может быть выведено в результате следующего запроса?

```
SELECT *
FROM "Книги"
WHERE ("Цена" < 10000 OR NOT ("Автор" IS NULL AND "Остаток" >
50))
```

9. Как можно иначе переписать запрос:

```
SELECT *
FROM "Заказы"
WHERE ("Дата" > '01/01/01' OR "Дата" < '01/03/01')
```

10. Напишите запрос для выборки всех заказчиков, чьи имена начинаются с буквы, попадающей в диапазон от А до М.
11. Напишите запрос для выборки всех книг, в названии которых есть словосочетание «база данных» или «базы данных» или «баз данных».
12. Напишите запрос для вывода всех поставщиков, данные о которых содержат NULL-значения.
13. Напишите запрос для подсчета суммы всех заказов на 31 января 2001 года.
14. Напишите запрос для подсчета числа различных не NULL-значений полей Адрес и Телефон в таблице Поставщики.

15. Напишите запрос для выборки наименьшей суммы предоплаты для каждого заказчика.
16. Напишите запрос для выборки заказчиков в алфавитном порядке, чьи имена начинаются с буквы М.
17. Напишите запрос для подсчета общей суммы заказов за каждый день с 01.02.01 по 01.04.01.
18. Напишите запрос к таблице Заказы для определения средней суммы предоплаты за каждый месяц.
19. Напишите запрос для вывода общей стоимости оплаченных заказов на каждый день в нисходящем порядке.
20. Напишите запрос для формирования рейтинга авторов в виде упорядоченного по убыванию списка авторов и количества их книг, проданных за некоторый период.

Содержание отчета:

Отчет о выполнении лабораторной работы должен содержать:

- название и тему лабораторной работы;
- цель лабораторной работы;
- краткие теоретические сведения;
- ход выполнения работы;
- выводы.

Раздел «Ход выполнения работы» должен содержать инфологическую модель БД на языке таблица-связь, тексты запросов на языке SQL с комментариями и результаты выполнения запросов в виде копий таблиц.

Контрольные вопросы:

В данном разделе приведено лишь несколько примеров контрольных вопросов, остальные будут приблизительно такого же содержания.

1. Объясните назначение операторов BETWEEN, IN, LIKE.
2. Что такое агрегирующие функции и как они используются?
3. Как называются функции SUM, AVG, MAX, MIN, COUNT? Почему?

4. Почему агрегирующие функции не могут применяться одновременно с именами полей в отсутствие модификатора GROUP BY?
5. Что будет выдано при использовании в списке выбора COUNT(*), COUNT(DISTINCT <Имя>)? Почему DISTINCT нельзя применять совместно с COUNT(*)?
6. Каково назначение предложений WHERE и HAVING?
7. Как можно вывести несколько данных разных типов в одной строке символов?
8. Что такое декартово произведение двух таблиц? Как его получить?
9. Что такое эквисоединение двух таблиц? Как его получить?
10. Что такое естественное соединение двух таблиц? Как его получить?
11. Что такое композиция двух таблиц? Как его получить?
12. Для чего используется объединение таблицы со своей копией?
13. Почему нельзя использовать операторы отношения для обработки результата простого вложенного подзапроса, если он возвращает более одного значения?
14. Что такое коррелированный вложенный подзапрос и как он обрабатывается?
15. В чем заключается сходство и отличие подзапросов с ANY и ALL от подзапросов с EXISTS?
16. Какие требования необходимо выполнить для объединения двух запросов?
17. Как средствами SQL выполнить объединение, пересечение и разность двух таблиц?
18. Для чего предназначена конструкция WITH?
19. Каковы особенности применения предложения UNION?

Список литературы, рекомендуемый к использованию по данной теме:

1. Братченко, Н. Ю. (Северо-Кавказский федеральный университет). Базы данных. Основы работы с СУБД ORACLE 10G EXPRESS

EDITION: учеб. пособие: Направление подготовки 230100 – Информатика и вычислительная техника / Н. Ю. Братченко; Сев.-Кав. федер. ун-т. – Ставрополь: СКФУ, 2013. – 202 с.: ил. – Гриф: Доп. УМО. – ISBN 978-5-88648-877-7

2. Разработка приложений на С с использованием СУБД PostgreSQL / И.А. Васюткина. – Новосибирск: НГТУ, 2015. – 143 с. – ISBN 978-5-7782-2699-9

3. Шацков, В.В. Программирование приложений баз данных с использованием СУБД MS SQL Server / В.В. Шацков: учебное пособие. Электронный ресурс: Санкт-Петербургский государственный архитектурно-строительный университет, ЭБС АСВ; Санкт-Петербург, 2015. – 80 с. – Книга находится в базовой версии ЭБС IPRbooks. – ISBN 978-5-9227-0607-0

4. Голицина, О. Л. Системы управления базами данных: [учеб. пособие] / О.Л. Голицина, Т.Л. Партыка, И.И. Попов. – М.: ФОРУМ, 2011. – 432 с.: ил. – (Профессиональное образование). – На учебнике гриф: Рек.МО. – Библиогр.: с. 315. – ISBN 5-8199-0251-3

5. Гудов, А. М. Базы данных и системы управления базами данных. Программирование на языке PL/SQL / А.М. Гудов; С.Ю. Завозкин; Т.С. Рейн. – Кемерово: Кемеровский государственный университет, 2010. – 134 с. – ISBN 978-5-8353-1005-0

6. М. В. Харченко. Организация баз данных. Методические указания к выполнению лабораторных работ для студентов направления 6.050102 – «Компьютерная инженерия». – Чернигов, ЧГТУ. 2013 – 141 с

Практическая работа №2. Определение данных и манипулирование данными

Цель работы: изучить инструкции языка определения данными для создания и удаления БД и ее объектов; изучить инструкции языка манипулирования данными.

Теоретическая часть

Инструкции языка определения данных

База данных

Для создания баз данных в языке SQL используется инструкция CREATE DATABASE, синтаксис которой (с сокращениями) имеет следующий вид:

```
CREATE DATABASE база_данных [ [ WITH ] [ OWNER [=]  
пользователь]
```

```
/* владелец базы данных (по умолчанию – пользователь, который ввел  
команду create) */ [ TEMPLATE [=] шаблон ]
```

```
/* используемый шаблон (по умолчанию – template1) */
```

```
[ ENCODING [=] кодировка ]
```

```
/* используемая кодировка (по умолчанию – кодировка шаблона) */
```

```
[ LC_COLLATE [=] порядок_сравнения]
```

```
/* порядок сравнения строк символов (по умолчанию – как в шаблоне)
```

```
*/
```

```
[ LC_STYPE [=] тип_категоризации]
```

```
/* категоризация символов (нижний/верхний регистр, цифра) (по  
умолчанию – как в шаблоне) */
```

```
[ TABLESPACE [=] табличное_пространство]
```

```
/* используемое табличное пространство (по умолчанию – как в  
шаблоне) */
```

```
[ CONNECTION LIMIT [=] connlimit ] ]
```

```
/* количество параллельных подключений (по умолчанию – «-1», т.е. не  
ограничено) */
```

Например, в результате выполнения команды

```
CREATE DATABASE a_new_database
```

будет создана новая база данных с параметрами по умолчанию:

```
WITH OWNER = postgres ENCODING = 'UTF8'
```

```
TABLESPACE = pg_default LC_COLLATE = 'UkrainianJJkraine.1251'
```

```
LC_CTYPE = 'Jkrainian_Jkraine. 1251'
```

```
CONNECTION LIMIT = -1
```

Для внесения изменений (изменение имени, владельца, табличного пространства, установление параметров конфигурации) в существующую базу данных служит команда ALTER DATABASE, а для удаления – DROP DATABASE. Например:

```
/* создаем базу данных BookShop_backup, используя базу данных BookShop как шаблон(BookShop не должна использоваться) */
```

```
CREATE DATABASE BookShop_backup TEMPLATE "BookShop";
```

```
/* создаем новую роль (пользователя) */
```

```
CREATE ROLE new_user;
```

```
/* изменяем владельца базы данных */
```

```
ALTER DATABASE BookShop_backup OWNER TO new_user;
```

```
— удаляем базу данных BookShop_backup DROP DATABASE BookShop_backup;
```

```
— удаляем роль DROP ROLE new_user;
```

Базовые таблицы

Инструкция CREATE TABLE

Таблицы создаются с помощью инструкции CREATE TABLE, которая в основном определяет тип таблицы (временная или базовая), имя таблицы, набор столбцов и ограничения целостности (ограничения для столбцов или для всей таблицы). Каждая таблица должна иметь, по крайней мере, один столбец. Синтаксис инструкции CREATE TABLE в PostgreSQL в сокращенном виде:

```
CREATE [ { TEMPORARY | TEMP } ] TABLE таблица
```

```

(
    [ { столбец тип_данных [ DEFAULT значение_по_умолчанию]
[<ограничение_столбца> [ ... ] ]
    | <ограничение_таблицы>
    | LIKE родительская_таблица [<параметры_LIKE> ] } [, ... ] ]
)
[ INHERITS (родительская_таблица [, ... ] ) ]
[ TABLESPACE табличное_пространство ]
где
<ограничение_столбца > ::= [CONSTRAINT ограничение]
{ { NOT NULL | [ NULL ] }
| CHECK (ограничение_на_значение_столбца)
| UNIQUE параметры_индексирования | PRIMARY KEY
| REFERENCES таблица [ (столбец) ]
| MATCH FULL – один столбец в составном внешнем ключе не может
быть NULL | MATCH PARTIAL – не реализовано
| [ MATCH SIMPLE] ] – один столбец в составном внешнем ключе
может быть NULL [ ON DELETE действие ] – действие при удалении
родительского ключа [ ON UPDATE действие] – действие при изменении
родительского ключа
}
<ограничение_таблицы > ::= [CONSTRAINT имя_ограничения ]
{ CHECK (ограничение_на_значение_столбца)
| UNIQUE (столбец [, ... ] ) параметры_индексирования | PRIMARY
KEY (столбец [, ... ] ) параметры_индексирования
| FOREIGN KEY (столбец [, ... ] ) REFERENCES таблица [ (столбец
[, ... ] ) ]
| MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
[ ON DELETE действие]
[ ON UPDATE действие] }

```

Если указано TEMPORARY или TEMP, создается временная таблица, т.е. такая таблица автоматически удаляется сразу после завершения транзакции.

При создании таблиц (или при их изменении) можно вводить ограничения на значения отдельных столбцов. Если ограничения введены, SQL будет отклонять любые значения, которые не удовлетворяют введенным ограничениям. Имеется два основных типа ограничений – ограничения столбца и ограничения таблицы. Различие между ними в том, что ограничение столбца применяется только к индивидуальным столбцам, в то время как ограничение таблицы применяется к группам из одного или более столбцов.

Ограничения NULL / NOT NULL разрешают или запрещают ввод в поле NULL-значений. Очевидно, что ограничение NOT NULL должно быть указано для первичных ключей, поскольку в противном случае под угрозой окажется целостность данных. Кроме того, отдельные поля таблиц по своему назначению могут требовать только определенных значений. Если поместить ключевые слова NOT NULL сразу после типа данных столбца, любая попытка поместить значение NULL в это поле будет отклонена. В противном случае, SQL будет считать, что для этого столбца NULL-значения разрешены.

Ограничение PRIMARY KEY (первичный ключ), используется для назначения первичных ключей. Поля с таким ограничением не могут принимать NULL-значений, и даже если ограничение NOT NULL не указывается для такого поля, SQL добавляет его по умолчанию. Ограничение PRIMARY KEY автоматически требует уникальности вводимых данных. Будучи назначенным для нескольких столбцов (составной первичный ключ) это ограничение задает уникальность комбинаций соответствующих значений, хотя по отдельности значения в каждом столбце составного ключа не обязательно должно быть уникальным.

Ограничение UNIQUE (уникальный), как и ограничение PRIMARY

KEY, ограничивает множество значений для указанных столбцов уникальными значениями. Как и ограничение PRIMARY KEY, ограничение UNIQUE может быть ограничением таблицы, и тогда оно определяет уникальность комбинаций значений соответствующих столбцов.

Ограничение FOREIGN KEY (внешний ключ) обеспечивает принцип ссылочной целостности. Когда столбец является внешним ключом, он определенным образом связан с таблицей, на которую он ссылается. Фактически это означает, что каждое значение в этом столбце (внешнем ключе) непосредственно привязано к значению в другом столбце (родительском ключе). Каждое значение (каждая строка) внешнего ключа должно недвусмысленно ссылаться к одному и только этому значению (строке) родительского ключа. Если это так, то система будет в состоянии ссылочной целостности. Понятно, что любое число внешних ключей может ссылаться к единственному значению родительского ключа. Понятно также, что в качестве родительского ключа могут выступать столбцы с ограничениями PRIMARY KEY или UNIQUE, т.е. столбцы с уникальными значениями и не содержащие NULL-значений. Внешний ключ может содержать только те значения, которые фактически представлены в родительском ключе или NULL-значения, попытка ввода других значений приводит к ошибке. Присутствие NULL-значений во внешнем ключе не нарушит ссылочную целостность, но даст возможность не вводить данные, если они пока не известны.

Подобно рассмотренным выше ограничениям, ограничение FOREIGN KEY может быть ограничением таблицы или столбца. В первом случае списки столбцов внешнего ключа и родительских ключей, перечисляемых после служебного слова REFERENCES и имени содержащей их таблицы, должны быть идентичны. Во втором случае ограничение относится только к одному столбцу и служебные слова FOREIGN KEY не нужны.

Ограничение CHECK позволяет установить условие, которому должно удовлетворять значение, вводимое в таблицу, прежде чем оно будет принято.

Ограничение CHECK состоит из ключевого слова CHECK, сопровождаемое предикатом, который использует указанное поле. Любая попытка модифицировать или вставить значение поля, которое могло бы сделать этот предикат неверным будет отклонена. Таким образом, можно предотвратить ввод нежелательных данных. Ограничение CHECK может использоваться в виде некоторой маски ввода, что обеспечит контроль заданного формата вводимых данных.

Создаваемая таблица может наследовать столбцы других таблиц, которые задаются в предложении LIKE или INHERITS. Различие между LIKE и INHERITS – в том, что при использовании INHERITS создается постоянная связь между дочерней и родительской таблицами, причем все изменения, производимые с родительской таблицей, наследуются дочерней и по умолчанию при выборке значений родительской таблицы данные из дочерней автоматически включаются в эту выборку. При использовании LIKE родительская и дочерняя таблица полностью независимы.

<параметры_ LIKE> :: =
 { INCLUDING | EXCLUDING } – *включая/исключая*
 { DEFAULTS – *значения по умолчанию* | CONSTRAINTS –
ограничения | INDEXES – *индексы* | STORAGE – *параметры хранения* |
 COMMENTS – *комментарии* | ALL }

Инструкция CREATE TABLE AS

С помощью инструкции CREATE TABLE AS можно создать таблицу, в том числе временную, на основе результатов запроса. Новая таблица получает те же столбцы с теми же типами данных, что и в запросе. При желании можно ввести иные имена столбцов для создаваемой таблицы, а также сохранить данные, полученные в результате выполнения запроса, в созданной таблице. Синтаксис инструкции CREATE TABLE AS (в упрощенном виде):

CREATE [{ TEMPORARY | TEMP }] TABLE таблица [(столбец
 [, ...])]

AS запрос [WITH [NO] DATA]

Инструкция DROP TABLE

Базовую таблицу можно в любой момент удалить с помощью предложения DROP TABLE (уничтожить таблицу):

DROP TABLE [IF EXISTS] базовая_таблица [, ...] [CASCADE | [RESTRIC]]

по которому удаляется описание таблицы, ее данные, связанные с ней представления и индексы, построенные для столбцов таблицы. Если указано IF EXISTS и таблицы нет, то сообщение об ошибке не выводится. Чтобы удалить таблицу, на которую ссылаются другие объекты базы данных, следует использовать CASCADE, иначе операция отклоняется (по умолчанию).

Для удаления содержимого таблицы (но не самой таблицы) можно использовать инструкции DELETE или TRUNCATE.

Инструкция ALTER TABLE

В SQL существует также инструкция ALTER TABLE (изменить таблицу), которая позволяет добавлять, удалять и модифицировать столбцы и ограничения. Синтаксиса ALTER TABLE имеет вид:

ALTER TABLE [ONLY] имя_таблицы [*] <действие> [, ...]

где (ограничиваясь лишь стандартными элементами)

<действие> :: = {

– *добавить столбец*

ADD [COLUMN] столбец тип [ограничение_столбца [...]]

– *удалить столбец*

| DROP [COLUMN] [IF EXISTS] столбец [RESTRIC | CASCADE]

– *изменить тип столбца*

| ALTER [COLUMN] столбец [SET DATA] TYPE тип

– *изменить значение по умолчанию*

| ALTER [COLUMN] столбец SET DEFAULT выражение | ALTER [COLUMN] столбец DROP DEFAULT

– *изменить ограничение NOT NULL*
 | ALTER [COLUMN] столбец { SET | DROP } NOT NULL
 – *переименовать столбец*
 | RENAME [COLUMN] имя_столбца TO новое_имя —
добавить/удалить ограничение таблицы | ADD ограничение таблицы
 | DROP CONSTRAINT [IF EXISTS] ограничение [RESTRICT |
 CASCADE] }

Однако модификация таблицы с помощью ALTER TABLE может изменить права доступа других пользователей к этой таблице, поэтому часто бывает проще просто создать новую таблицу с необходимыми изменениями при создании, и переписать в нее данные из старой таблицы.

Ниже приведены примеры описанных инструкций.

```
/* создаем новую таблицу Доставка */
CREATE TABLE "Доставка"
( "Заказ" integer,
  "Курьер" character varying(40) DEFAULT 'foo',
  "Дата_время" timestamp without time zone DEFAULT
CURRENT_TIMESTAMP, "Доставлен" boolean DEFAULT FALSE);

/* добавляем первичный ключ */
ALTER TABLE "Доставка" ADD CONSTRAINT "рк_доставка"
PRIMARY KEY ("Заказ");

/* добавляем внешний ключ для ссылки на Заказы */
ALTER TABLE "Доставка" ADD CONSTRAINT "Тк_доставка"
FOREIGN KEY ("Заказы") REFERENCES "Заказы" ("Код_заказа")
MATCH SIMPLE ON UPDATE CASCADE ON DELETE SET NULL;
```

Индексы

Индекс представляет собой отдельную физическую структуру в базе данных, созданную на основе одного или нескольких столбцов таблицы и предназначенную для ускорения поиска данных по значениям проиндексированных столбцов. Фактически в индексе фиксируются значения

проиндексированного столбца в упорядоченном виде и их расположение в памяти. Если для создания одного индекса указывается более одного столбца, то второй упорядочивается внутри первого, третий – внутри второго, и т.д., т.е. каждый последующий – внутри предыдущего. Однако следует помнить, что чем больше столбцов в индексе, тем медленнее он работает и тем больше занимает места в памяти системы. Обычно пользователи не знают о существовании индекса, SQL автоматически определяет, позволено ли пользователю использовать индекс, и если да, то разрешает использовать его.

PostgreSQL поддерживает четыре типа индексов: B-tree, hash, GiST, GIN. По умолчанию используется B-tree. Обычно при ведении ограничений первичного ключа и UNIQUE соответствующий индекс создается автоматически, но для первичного это делается неявно (его нельзя удалить как обычные индексы), В PostgreSQL создать индекс указанного типа для одного или нескольких полей таблицы можно с помощью инструкции (упрощенный вид):

```
CREATE [ UNIQUE ] INDEX [ индекс ]  
ON таблица [ USING тип_индекса ]  
( { столбец | выражение } [ [aSc] | DeSC] [, ...] )
```

Если имя индекса вводится, оно не может быть использовано для чего-то другого в базе данных (любым пользователем). Однажды созданный, индекс будет невидим пользователю – SQL сам решает, использовать его или нет.

Если при создании индекса указать UNIQUE, то дубликаты значений в соответствующих столбцах будут недопустимы. Если такой индекс создается для столбца, в котором уже содержатся дублирующие друг друга значения, то индекс не будет создан. Если же индекс создается для пустой таблицы, то ввод в дальнейшем повторяющихся значений в индексированный столбец будет отклоняться, поэтому можно сказать, что создание уникальных индексов также является средством обеспечения уникальности значений в

столбцах.

При выборе типа индекса следует учитывать, что создать UNIQUE-индекс и указать порядок сортировки (ASC/DESC) можно только для индексов типа B-tree. Индексировать по нескольким полям можно только при выборе индексов B-tree, GiST, GIN.

Любой индекс может быть кластерным (CLUSTERED). Это означает, что на его основе можно выполнить кластеризацию таблицы, т.е. переупорядочивание физических страниц данных в соответствии с логическим порядком данного индекса (что естественно увеличивает число дисковых операций ввода-вывода). Поэтому для каждой таблицы может быть создан только один кластерный индекс, тогда как не кластерных индексов может быть несколько (практически оптимальным количеством является 2-6 не кластерных индексов на таблицу). Кластеризация дает преимущество в том случае, если большую часть данных, которые выбираются из таблицы, составляют группы индексированных данных, а не случайно выбранные отдельные данные, т.к. такие группы располагаются на одной и той же физической странице (или последовательных страницах). Но необходимо учитывать, что при выполнении кластеризации таблицы все операции, в том числе чтение, с этой таблицей блокируются.

Для кластеризации в PostgreSQL используется инструкция

`CLUSTER [VERBOSE] [ таблица [ USING индекс ] ]`

где VERBOSE позволяет вывести сообщение о кластеризации каждой таблицы.

Упорядоченность данных, возникающая в результате кластеризации, не распространяется на данные, которые изменяются после проведения кластеризации, поэтому при обновлении данных кластеризацию необходимо повторить. Для этого достаточно ввести `CLUSTER таблица`. Если ввести `CLUSTER` без параметров, рекластеризации будут подвергнуты все таблицы базы данных, для которых ранее были созданы кластерные индексы, поэтому такие индексы создаются обычно для тех таблиц, данные в которых

изменяются редко.

В качестве альтернативы кластеризации можно использовать прием, заключающийся в создании копии таблицы с помощью CREATE TABLE AS, где в запросе осуществляется упорядочивание по индексированным полям, и последующем удалении исходной таблицы. В случае, когда удаление исходной таблицы не допустимо, следует использовать представление (см. далее).

Чтобы удалить индекс, надо знать его имя или его OID. Для удаления индекса по имени используется инструкция:

```
DROP INDEX имя_индекса
```

Индекс можно удалить непосредственно из системной таблицы pg_index, зная его OID. Удаление индекса не влияет на содержание полей.

Примеры инструкций управления индексами приведен ниже.

```
CREATE UNIQUE INDEX “_ui_” ON "Доставка" ("Заказ", "Курьер",  
“Дата_время”); CLUSTER VERBOSE "Доставка заказов" USING “_ui_”;
```

Представления

Представления – это виртуальные таблицы, формируемые из столбцов и строк других таблиц. Они работают в запросах и командах манипулирования данными точно так же, как и базовые таблицы, но, в отличие от последних, не содержат никаких собственных данных. Представления подобны окнам, через которые пользователь просматривает информацию, которая физически хранится в базовой таблице. Другими словами, представление – это фактически запрос, который выполняется всякий раз, когда представление упоминается в инструкции, при этом результаты этого запроса в каждый момент и составляют содержание представления.

Представление создается с помощью инструкции CREATE VIEW, синтаксис которой имеет следующий вид:

```
CREATE [OR REPLACE] [TEMP | TEMPORARY] VIEW  
имя_представления
```

[(столбец [, ...])] AS запрос

где запрос, следующий за AS и являющийся определением данного представления, не исполняется, а просто сохраняется в системном каталоге. Список имен столбцов должен быть обязательно определен лишь в тех случаях, когда хотя бы один из столбцов запроса не имеет имени (создается с помощью выражения, SQL-функции или константы) или два (или более) столбцов подзапроса имеют одно и то же имя. Если же список отсутствует, то представление наследует имена столбцов из подзапроса.

При использовании CREATE OR REPLACE, если представление с таким же именем уже существует в данной схеме, оно будет переопределено.

С помощью TEMP или TEMPORARY создаются временные представления. Если представление определяется на основе только временных таблиц, оно автоматически становится временным, независимо от того, указано ли TEMP/TEMPORARY при его создании.

Синтаксис инструкции удаления представления подобен синтаксису удаления базовых таблиц:

DROP VIEW имя_представления

Преимущество использования представлений

По сравнению с базовой таблицей преимущество представления заключается в том, что представление будет модифицировано автоматически всякий раз, когда таблица, лежащая в его основе, изменяется. Содержание представления не фиксировано, и переназначается каждый раз, когда какая-то инструкция ссылается на представление.

В представлениях можно использовать модификатор GROUP BY, и тогда они называются групповыми представлениями. Представления не требуют, чтобы их вывод осуществлялся из одной базовой таблицы. Так как почти любой допустимый запрос SQL может быть использован в представлении, он может выводить информацию из любого числа базовых таблиц, или из других представлений. Представления могут также использовать и подзапросы, включая коррелированные подзапросы.

Наконец, одна из основных задач, которую позволяют решать представления, – обеспечение независимости пользовательских программ от изменения логической структуры базы данных при ее расширении и/или изменении размещения столбцов, возникающего, например, при разбиении таблиц на две таблицы. В последнем случае можно создать представление-соединение с именем и структурой исходной таблицы, позволяющее сохранить программы, существовавшие до изменения структуры базы данных.

Модифицируемые и не модифицируемые представления

К представлениям, как и к базовым таблицам, в общем случае применимы инструкции манипулирования данными. Однако если из базовых таблиц можно удалять и обновлять любые строки и вводить в такие таблицы новые строки, то этого нельзя сказать о представлениях. Если команды модификации могут выполняться в представлении, представление называется модифицируемым; в противном случае оно предназначено только для чтения при запросе. В теории бах данных определены критерии, по которым определяют, является ли представление модифицируемым или нет. Считается, что представление является модифицируемым, если оно:

- относиться к одной и только одной базовой таблице;
- содержит первичный ключ этой таблицы;
- не имеет никаких полей, которые бы являлись агрегатными функциями;
- не содержит DISTINCT, GROUP BY или HAVING в своем определении;
- не использует подзапросы;
- может быть использовано в другом представлении, но это представление также модифицируемо;
- не использует константы, строки, или выражения среди выбранных полей вывода;
- содержит любые поля основной таблицы, которые имеют ограничение NOT NULL, если другое ограничение по умолчанию не определено.

На практике ответ на вопрос о модифицируемости представлений

оказывается неоднозначным, и теоретически модифицируемое представление может оказаться не модифицируемым или наоборот. Безусловно, модифицируемыми являются представления, полученные из единственной базовой таблицы простым исключением некоторых ее строк и/или столбцов, обычно называемые «представление-подмножество строк и столбцов».

Различия между модифицируемыми и не модифицируемыми представлениями очевидны. Модифицируемые представления используются в основном точно так же, как и базовые таблицы. Фактически, пользователи не могут даже осознать, является ли объект, который они запрашивают, базовой таблицей или представлением. Это – превосходный механизм защиты конфиденциальной информации или тех частей таблицы, которые не относятся к потребностям данного пользователя. Пользователям предоставляют только интересующие их данные в наиболее удобной для них форме (окно в таблицу или в любое соединение любых таблиц).

Не модифицируемые представления позволяют получать и переформатировать данные более рационально, фактически представляя собой библиотеку сложных запросов, которые можно выполнять многократно. Кроме того, такие представления могут также выполнять функции защиты. Например, можно разрешить пользователям видеть лишь агрегатные данные, такие как усредненные значения или значения минимакса, не видя деталей.

В PostgreSQL все создаваемые представления по умолчанию являются не модифицируемыми. Чтобы сделать представление модифицируемым (если это возможно!), необходимо создать правило (RULE), которое обеспечивает выполнение требуемых действий над другими таблицами. Вообще, с помощью правил можно исполнять дополнительные действия при выполнении тех или иных команд над любыми таблицами или представлениями. Создать правило можно с помощью инструкции:

```
CREATE [ OR REPLACE ] RULE { таблица | представление } AS  
ON { SELECT | INSERT | UPDATE | DELETE.}
```

TO таблица [WHERE условие]
DO [[ALSO] | INSTEAD] { NOTHING | команда | (команда ; команда
[; ...]) }

команда ::= SELECT | INSERT | UPDATE | DELETE | NOTIFY

где ALSO – указывает на то, что команды, содержащиеся в правиле, будут выполняться как дополнение к исходной команде, INSTEAD – вместо исходной команды, NOTHING – ничего не будет выполняться.

Примеры инструкций управления представлениями:

/ создаем не модифицируемое представление */*

```
CREATE VIEW "Просроченные _доставки_1" ("Заказ", "Получатель")  
AS
```

```
SELECT "Заказ", "Заказчик", "Дата_время"  
FROM "Доставка" INNER JOIN "Заказы"  
ON "Доставка"."Заказ" = "Заказы"."Код_заказа"  
WHERE NOT("Доставка"."Доставлен")  
ORDER BY 3;
```

```
SELECT * FROM "Просроченная доставка";
```

/ создаем модифицируемое представление */*

```
CREATE OR REPLACE VIEW "Просроченные _доставки_2" AS  
SELECT "Заказ", "Дата_время"  
FROM "Доставка"  
WHERE NOT("Доставлен");
```

Инструкции языка манипулирования данными

Инструкция INSERT

Таблицы создаются инструкцией CREATE TABLE. Эта инструкция создает пустую таблицу – таблицу без строк. Значения вводятся, удаляются или обновляются с помощью инструкций языка манипулирования данными (DML), основными из которых являются инструкции INSERT (вставить), DELETE (удалить), и UPDATE (обновить). Подобно предложению SELECT они могут оперировать как базовыми таблицами, так и представлениями.

Синтаксис инструкции INSERT:

```
INSERT [INTO] {базовая_таблица | представление} [(столбец [, ...] )]  
{ DEFAULT VALUES | VALUES ({ DEFAULT | NULL | выражение }  
[, ...] )  
| запрос }  
[ RETURNING { * | выражение [ [ AS ] имя ] } [, ...] ]
```

Предложение INTO необязательно и просто улучшает читабельность инструкции.

Предложение RETURNING (не стандартное) позволяет вывести на экран (так же, как и при использовании SELECT) значения выражений, построенных на основе вставляемых данных.

Возможны два варианта использования INSERT: без запроса и с запросом. В первом варианте в указанную таблицу вставляется строка со значениями полей, указанными в перечне предложения VALUES (значения), причем i-е значение должно соответствовать i-му столбцу в списке столбцов (как и в инструкции SELECT, порядок следования столбцов может быть произвольным). Столбцы, не указанные в списке, заполняются NULL-значениями или значениями по умолчанию. Если NULL-значениями для такого столбца не допустимы и не указано значение по умолчанию, то транзакция отменяется и выводится сообщение об ошибке. Если в списке VALUES указаны все столбцы модифицируемой таблицы и порядок их перечисления соответствует порядку столбцов в описании таблицы, то список столбцов после имени таблицы или представления можно опустить.

Во втором варианте использования инструкции INSERT сначала выполняется запрос, т.е. по предложению SELECT в памяти формируется рабочая таблица, а потом строки рабочей таблицы загружаются в INSERT-таблицу. При этом i-й столбец рабочей таблицы (i-й элемент списка SELECT) соответствует i-му столбцу в списке столбцов модифицируемой таблицы. Здесь также при выполнении указанных выше условий может быть опущен список столбцов предложения INTO. Инструкция INSERT с запросом чаще

применяется для переноса строк из одной таблицы в другую (возможно через представление, но, очевидно, модифицируемое).

Примеры:

/ вставляем в таблицу Доставка новую строку */*

INSERT INTO "Доставка"

VALUES (1000, – Заказ

DEFAULT, – Курьер

DEFAULT, – Дата_время

DEFAULT) – Доставлен

/ вставляем в таблицу Доставка все оплаченные заказы */*

INSERT INTO "Доставка" ("Заказ", "Дата_время")

SELECT "Код_заказа", "Дата" + '1 0:0':::interval FROM "Заказы"

WHERE "Оплачен" LIKE '%да%';

/ создаем правило для представления Просроченные_ доставки_2 на вставку */* CREATE RULE _ins_ AS ON INSeRt TO "Просроченные_ доставки_2"

DO INSTEAD

INSERT INTO "Доставка" VALUES (NEW.'^^', NEW."Дата_время");

/ проверяем, как работает правило _ins_ */*

INSERT INTO "Заказы" VALUES (DEFAULT, NULL, NULL, NULL, NULL); INSERT INTO "Просроченные_ доставки_2" VALUES ((SELECT max("Код_заказа") FROM "Заказы"), now()+ '1 0:0':::interval);

SELECT * FROM "Просроченные_ доставки_2";

SELECT * FROM "Доставка";

Инструкция DELETE

Инструкция DELETE позволяет удалить содержимое всех строк указанной таблицы или тех ее строк, которые удовлетворяют указанному условию. Эта инструкция имеет формат:

DELETE FROM [ONLY] {базовая_таблица | представление} [[AS] псевдоним]

[USING список]

[WHERE { условие | WHERE CURRENT OF курсор }]

[RETURNING { * | выражение [[AS] имя] [, ...] }]

Список в предложении USING аналогичен списку предложения FROM инструкции SELECT.

При отсутствии предложения WHERE удаляется содержимое всех строк указанной таблицы, в противном случае – те ее строки, которые удовлетворяют условию в предложении WHERE.

Предложение RETURNING аналогично такому же предложению инструкции INSERT.

Перед удалением строк из таблицы имеет смысл выполнить инструкцию SELECT с таким же условием отбора строк, как и условие в предложении WHERE инструкции DELETE, с тем, чтобы убедиться в корректности выполняемой операции удаления.

Для удаления всех строк инструкция DELETE используется редко. Если необходимо удалить все строки в таблице, во многих случаях следует отдать предпочтение инструкции TRUNCATE TABLE, т.к. эта инструкция работает более быстро, удаляя ссылки данной таблицы на физические страницы данных, а не удаляя данные построчно как инструкция DELETE.

Но следует помнить, что TRUNCATE TABLE, в отличие от DELETE, не записывает удаляемые строки в журнал транзакций, а значит, удаленные строки восстановить нельзя. Кроме того, TRUNCATE TABLE не применима к таблицам, на которые ссылаются другие таблицы через внешние ключи. Примеры:

/ удаляем из таблицы Заказы строки с неопределенными значениями кода заказчика */:* DELETE FROM “Заказы”

WHERE “Код_заказчика” IS NULL;

/ создаем правило для представления Просроченные_доставки_2 на удаление */*

CREATE OR REPLACE RULE _del_ AS

```

ON DELETE TO "Просроченные_ доставки_2" DO INSTEAD DELETE
FROM "Доставка"
WHERE "Доставка"."Заказ" = оИ."Заказ";
/* удаляем из Просроченные_ доставки_2 строки, соответствующие
неопределенным заказчикам */
SELECT "Код_заказа" FROM "Заказы" WHERE ".Код_заказчика" IS
NULL;
DELETE FrOm "Просроченные_ доставки_2"
USING "Заказы"
WHERE      ("Заказ"      =      "Заказы"."Код_заказа")      AND
("Заказы"."Код_заказчика" IS NULL);

```

Инструкция UPDATE

Инструкция UPDATE используется для изменения существующих значений. В разных диалектах SQL форматы этой инструкции могут отличаться как друг от друга, так и от формата ANSI SQL. В PostgreSQL принят следующий формат (в упрощенном виде):

```

UPDATE [ ONLY ] {базовая_таблица | представление} [ [ AS ]
псевдоним ]
SET { столбец = { выражение | DEFAULT } |
(столбец [, ...] ) = ( { выражение | DEFAULT } [, ...] ) } [, ...]
[ FROM список ]
[ WHERE условие | WHERE CURRENT OF курсор ]
[ RETURNING { * | выражение [ [ AS ] имя ] [, ...] } ]

```

В отсутствие необязательных предложений FROM и WHERE обновляются значения указанных в списке предложения SET столбцов во всех строках модифицируемой таблицы, но список предложения SET может включать лишь столбцы из обновляемой таблицы, т.е. значение одного из столбцов модифицируемой таблицы может заменяться на значение ее другого столбца или выражения, содержащего значения нескольких ее столбцов, включая изменяемый.

Предложение WHERE позволяет сократить число обновляемых строк, указывая условие их отбора.

Предложение FROM позволяет производить обновление значений модифицируемой таблицы по значениям столбцов из других таблиц, представлений или подзапросов. Перечень таблиц-источников предложения FROM содержит имя модифицируемой таблицы и тех таблиц, значения столбцов которых используются для обновления. При этом, естественно, таблицы должны быть связаны друг с другом в WHERE. В выражениях, значения которых присваиваются столбцам в списке предложения SET, следует уточнять имена используемых столбцов, предваряя их именем таблицы или псевдонимом, при этом в качестве выражения может выступать подзапрос, возвращающий скалярное значение.

Если команды манипулирования данными нарушают ограничения, установленные на столбцы таблиц, то соответствующие транзакции отменяются, и выводится сообщение об ошибке. Наиболее важным является вопрос о взаимоотношении команд манипулирования данными и внешних и родительских ключей. Для столбцов, определенных как внешние ключи, любые значения, которые помещаются в эти столбцы командами INSERT или UPDATE должны быть представлены в их родительских ключах. Можно помещать NULL-значения в эти столбцы, несмотря на то, что NULL-значения не допустимы в родительских ключах. Можно удалять или изменять любые строки с внешними ключами, но любое значение родительского ключа не может быть удалено или изменено. Это означает, например, что нельзя удалить запись о заказчике из таблицы Заказчики пока на эту таблицу есть ссылки внешних ключей других таблиц.

Для разрешения аномалий, возникающих при изменении или удалении текущего значения родительского ключа, говоря в целом об SQL, имеется три возможности:

- запретить изменения в родительском ключе (ограниченное изменение – RESTRICTED);

- разрешить автоматическое изменение внешнего ключа (каскадное изменение – CASCADES);
- разрешить изменение в родительском ключе и автоматически установить внешний ключ в NULL (SET NULL).

Примеры:

/ корректируем количество заказываемых книг с учетом имеющегося в наличии остатка */* UPDATE "Заказы"

SET "Количество" = T2."Остаток"

FROM "Заказы" T1, "Книги" T2

WHERE T1 ."Код_книги" = T2."Код_книги" AND T1."Количество" > T2."Остаток"

/ создаем правило для представления Просроченные_доставки_2 на обновление */*

CREATE RULE _upd_ AS ON UPDATE TO "Просроченные_доставки_2"

DO INSTEAD

UPDATE "Доставка"

SET "Заказ" = NEW.'^', "Дата_время" = NEW."Дата_время"

WHERE "Заказ" = OLD.'^';

/ изменяем данные в представлении Просроченные_доставки_2 */*

UPDATE "Просроченные_доставки_2" AS divr

SET "Дата_время" = "Дата_время" + '1 0:0'::interval

FROM "Заказы" AS ord

WHERE divr."Заказ" = o^."Код_заказа" AND o^."Код_книги" = 5;

Оборудование и материалы:

Для изучения материалов этой лабораторной работы необходимо:

1. Компьютер под управлением операционной системы Windows 8/10 с параметрами по умолчанию.
2. Установленные на компьютер программы Microsoft Visio, PostgreSQL, MySQL, VMWare/VirtualBox.

3. Текстовый процессор Microsoft Word 2010/2013/2016.

Указания по технике безопасности:

К выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж по технике безопасности.

Задания:

1. Напишите инструкцию, результатом выполнения которой была бы вставка нескольких строк в таблицу Сотрудники, созданную ранее.
2. Напишите инструкцию, результатом выполнения которой было бы удаление всех неоплаченных на текущую дату заказов из таблицы Заказы.
3. Напишите инструкцию, результатом выполнения которой было бы увеличение комиссионных трех сотрудников, имеющих наибольшее количество оформленных заказов.
4. Создайте таблицу такой же структуры, что и таблица Сотрудники, и напишите инструкцию для вставки в нее данных обо всех сотрудниках из таблицы Сотрудники, оформивших более трех заказов и имеющих стаж не менее 5 лет.
5. Напишите инструкцию, результатом выполнения которой было бы удаление из таблицы Заказчики тех заказчиков, которые сделали последний заказ более 1 года тому назад. Удалите соответствующие заказы из таблицы Заказы.
6. В таблицу Заказы добавьте столбец Номер_договора типа uuid, введите в каждой строке в этом столбце значение, равное количеству секунд, прошедших от '01.01.1900 00:00' до того момента, когда очередное такое значение вносится в таблицу, и сделайте Номер_договора первичным ключом, предварительно удалив созданный ранее первичный ключ.
7. Создайте представление таблицы Сотрудники, включив в него только поля личный_номер и размер_комиссионных. С помощью этого

представления увеличьте на 5% размер комиссионных для продавца, имеющего максимальное число принятых заказов.

8. Создайте представления для ролей «Директор», «Закупщик», «Продавец», исходя из того, что им необходимо создавать отчеты (таблица 7.1).

Таблица 7.1 – Отчеты для ролей

Роль	Отчеты
Директор	Отчет по продажам: показывает, какие книги пользуются спросом, т.е. формировать рейтинг книг. Отчет по продавцам: показывает количество проданных книг в месяц и тоже самое в денежном эквиваленте. Отчет по поставщикам: показывает список поставщиков и их рейтинги, с возможностью изменить рейтинг вручную и добавить какое-то примечание.
Закупщик	Отчет по поставщикам: показывает список поставщиков указанной книги, цены, условия поставки. Отчет по книгам: показывает наличие книг с возможностью заносить информацию о поступивших книгах.
Продавец	Отчет по книгам: показывает наличие книг с возможностью заносить информацию о проданных книгах. Отчет по продажам: показывает итоговые показатели продаж по продавцам на указанную дату.

Содержание отчета:

Отчет о выполнении лабораторной работы должен содержать:

- название и тему лабораторной работы;
- цель лабораторной работы;
- краткие теоретические сведения;
- ход выполнения работы;
- выводы.

Раздел «Ход выполнения работы» должен содержать инфологическую модель БД на языке таблица-связь, отчеты DDL для всех базовых таблиц разработанной БД, скрипты и результаты выполнения запросов контрольного задания.

Контрольные вопросы:

В данном разделе приведено лишь несколько примеров контрольных вопросов, остальные будут приблизительно такого же содержания.

1. Дайте характеристику основным типам ограничений.
2. Как можно обеспечить уникальность значений для данного столбца или группы столбцов?
3. Что такое индекс? Какие типы индексов поддерживает PostgreSQL?
4. Что такое кластеризация таблиц? Почему для таблицы может быть создан только один кластерный индекс?
5. Что будет введено в столбец, если в инструкции INSERT этот столбец не указан?
6. Чем отличается TRUNCATE от DELETE?
7. Как в SQL решаются аномалии удаления и обновления значений родительских ключей?
8. Что такое представление?
9. В чем заключается преимущество использования в запросах представлений по сравнению с базовыми таблицами БД?
10. Какие представления называются модифицируемыми?

Список литературы, рекомендуемый к использованию по данной теме:

1. Братченко, Н. Ю. (Северо-Кавказский федеральный университет). Базы данных. Основы работы с СУБД ORACLE 10G EXPRESS EDITION: учеб. пособие: Направление подготовки 230100 – Информатика и вычислительная техника / Н. Ю. Братченко; Сев.-Кав. федер. ун-т. – Ставрополь: СКФУ, 2013. – 202 с.: ил. – Гриф: Доп. УМО. – ISBN 978-5-88648-877-7
2. Разработка приложений на С с использованием СУБД PostgreSQL / И.А. Васюткина. – Новосибирск: НГТУ, 2015. – 143 с. – ISBN 978-5-7782-2699-9
3. Шацков, В.В. Программирование приложений баз данных с использованием СУБД MS SQL Server / В.В. Шацков: учебное пособие

Электронный ресурс: Санкт-Петербургский государственный архитектурно-строительный университет, ЭБС АСВ; Санкт-Петербург, 2015. – 80 с. – Книга находится в базовой версии ЭБС IPRbooks. – ISBN 978-5-9227-0607-0

4. Голицина, О. Л. Системы управления базами данных: [учеб. пособие] / О.Л. Голицына, Т.Л. Партыка, И.И. Попов. – М.: ФОРУМ, 2011. – 432 с.: ил. – (Профессиональное образование). – На учебнике гриф: Рек.МО. – Библиогр.: с. 315. – ISBN 5-8199-0251-3

5. Гудов, А. М. Базы данных и системы управления базами данных. Программирование на языке PL/SQL / А.М. Гудов; С.Ю. Завозкин; Т.С. Рейн. – Кемерово: Кемеровский государственный университет, 2010. – 134 с. – ISBN 978-5-8353-1005-0

6. М. В. Харченко. Организация баз данных. Методические указания к выполнению лабораторных работ для студентов направления 6.050102 – «Компьютерная инженерия». – Чернигов, ЧГТУ. 2013 – 141 с

Практическая работа №3. Функции, определенные пользователем

Цель работы: изучить инструкции создания и модификации хранимых процедур и триггеров в PostgreSQL.

Теоретическая часть

Типы функций PostgreSQL

PostgreSQL поддерживает четыре вида функций, определяемых пользователем:

- функции языка запросов или хранимые процедуры (функции, написанные на SQL);
- функции процедурного языка (функции, написанные, например, на PL/pgSQL);
- внутренние функции (статично связанные функции, написанные на языке C);
- C-функции (динамически связываемые функции, написанные на языке C/C++).

Все перечисленные функции могут иметь аргументы (параметры), относящиеся к базовым типам, составным типам или их комбинации. Кроме того, функции могут возвращать значения базового или составного типа или их последовательности.

Для создания функций в PostgreSQL используется инструкция CREATE FUNCTION, синтаксис которой имеет следующий вид:

```
CREATE [ OR REPLACE ] FUNCTION
    имя_функции ( [ [ [ IN ] | OUT | INOUT | VARIADIC ] [имя_параметра]
тип_параметра [ { DEFAULT | = } выражение ] [, ...] ] )
    [ RETURNS тип_возвращаемого_значения | RETURNS TABLE
( столбец тип [, ...] ) ] { LANGUAGE | WINDOW
| { IMMUTABLE | STABLE | [ VOLATILE ] }
| { [ CALLED ON NULL INPUT ] | RETURNS NULL ON NULL INPUT |
STRICT }
| { [ [ EXTERNAL ] SECURITY INVOKER ] | [ EXTERNAL ] SECURITY
```

DEFINER }

| AS 'тело_функции'

} ..

где

[[IN] | OUT | INOUT | VARIADIC – спецификация параметров: входных (IN), выходных (OUT), входных-выходных (INOUT) и массив входных параметров переменной длины (VARIADIC).

RETURNS | RETURNS TABLE – определяет тип возвращаемого значения. SQL-функция может ничего не возвращать (VOID), возвращать первую строку из результата последнего запроса, содержащегося в теле функции, или набор значений некоторого типа (RETURNS SETOF или RETURNS TABLE).

Примечание: Считается, что первая строка из множества строк, являющихся результатом выполнения некоторого запроса, определена нечетко, если в запросе не используется ORDER BY.

Если функция определена с возвращаемым значением, отличным от VOID, то последней командой в теле функции должна быть SELECT, INSERT, UPDATE или DELETE с выражением, которое совместимо по типу с возвращаемым значением функции.

LANGUAGE – язык, на котором написана функция (например, SQL). WINDOW – оконная функция. Оконная функция выполняет вычисление для набора строк, которые так или иначе связаны с текущей строкой курсора. Это похоже на вычисления с использованием стандартных агрегирующих функций, но в отличие от агрегирующих функций, использование оконной функции не требует объединения строк в группы с выводом единственной результирующей строки для каждой группы. Практически этот параметр применим лишь для функций, написанных на C.

IMMUTABLE | STABLE | [VOLATILE] – информация для оптимизатора запросов, указывающая, что: функция не будет модифицировать БД и всегда возвращает то же самое значение при одних и

тех же значениях параметров (IMMUTABLE), функция не будет модифицировать БД и не изменяет возвращаемое значение в рамках одной транзакции (однократного прохода таблицы) (STABLE), функция может изменять возвращаемое значение даже в рамках одной транзакции (VOLATILE).

`CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT` – функция может вызываться при null-значениях некоторых ее параметров (`CALLED ON NULL INPUT`), в противном случае функция всегда возвращает null, если любой из ее параметров имеет значение null.

`SECURITY INVOKER | SECURITY DEFINER` – функция будет выполняться с правами того пользователя, который ее вызывает (`INVOKER`), или того, который ее создал (`DEFINER`).

После служебного слова `AS` идет строка, содержащая тело функции. Это может быть имя встроенной функции, SQL-скрипт, текст на процедурном языке или путь к объектному файлу. Вместо одинарных кавычек для ограничения тела функции удобнее использовать два подряд стоящих символа доллара (`$$`), иначе при использовании в определении функции одинарных кавычек или обратной косой черты их придется дублировать.

Для изменения и удаления функций можно использовать инструкции `ALTER FUNCTION` и `DROP FUNCTION`. При использовании `ALTER`

`FUNCTION` сохраняются все права доступа, назначенные ранее модифицируемой функции.

Хранимые процедуры

Создание хранимых процедур

Функции языка запросов или хранимые процедуры (Stored Procedure – SP) – набор заранее скомпилированных операторов SQL и операторов управления программой, который хранится как физические объекты базы данных. Любой набор SQL-инструкций, содержащий `SELECT`, `INSERT`, `UPDATE`, `DELETE` и др., кроме `BEGIN`, `COMMIT`, `ROLLBACK` и `SAVEPOINT`, отделенные друг от друга точкой с запятой, может быть

определен как SP. Они расширяют стандартные возможности SQL, позволяя использовать входные и выходные параметры, операторы принятия решения и объявления переменных. SP может состоять из одной инструкции SELECT, а может включать целый набор операторов SQL и операторов управления программой, которые позволяют реализовать достаточно сложные операции над данными. SP допускают вложенность, т.е. одна SP может вызвать другую, которая, в свою очередь, может вызвать еще одну, и т.д.

По сравнению со стандартными операторами SQL SP обеспечивают значительный выигрыш в быстродействии, поскольку содержащиеся в них операторы SQL заранее скомпилированы. После первого выполнения SP создается план ее выполнения, который сохраняется в процедурном КЭШе, и поэтому все последующие выполнения протекают намного быстрее по сравнению с эквивалентным набором операторов SQL.

SP следует использовать во всех случаях, когда необходимо получить максимальное быстродействие и свести код SQL в единую программу. Чаще всего используются SP, выполняющие вставку, удаление и обновление данных, а также формирующие данные для отчетов.

SP также можно использовать в качестве механизма защиты. К ним прибегают, например, в тех случаях, когда по каким-то причинам нельзя предоставлять прямой доступ пользователям к таблицам и представлениям БД, чтобы не допустить случайной их модификации. Тогда пользователям предоставляется доступ к таблицам только по чтению, а для выполнения таких операций, как UPDATE и DELETE, создаются соответствующие SP, на выполнение которых пользователи и получают права, т.е. пользователи получают доступ к таблицам БД только путем выполнения SP.

В отличие от представлений, для использования которых пользователь должен иметь права на все заявленные в представлении объекты БД, пользователь может выполнять SP независимо от того, имеет ли он доступ к таблицам, представлениям или другим функциям, содержащимся в описании SP. Соответствующие права доступа на объекты, используемые в SP, нужны

только ее владельцу, который может предоставить другим пользователям разрешение на ее запуск.

Для SP существуют ограничения на применение некоторых инструкций SQL внутри тела SP. Внутри SP нельзя выполнять такие инструкции, как CREATE TABLE, CREATE VIEW, CREATE FUNCTION. Обмен данными между SP осуществляется посредством параметров или временных таблиц. Ниже приведены примеры хранимых процедур.

/ SP, возвращающие скалярное значение */*

```
CREATE OR REPLACE FUNCTION suppliers()
```

```
RETURNS bigint
```

```
AS $$ SELECT count(*) FROM "Заказы"; $$
```

```
LANGUAGE sql;
```

```
CREATE OR REPLACE FUNCTION customers()
```

```
RETURNS character varying
```

```
AS $$ SELECT "Заказчик" FROM "Заказы"; $$
```

```
LANGUAGE sql;
```

/ SP, возвращающие набор записей */*

```
CREATE OR REPLACE FUNCTION orders()
```

```
RETURNS "Заказы"
```

```
AS $$ SELECT * FROM "Заказы" ; $$
```

```
LANGUAGE sql;
```

```
CREATE OR REPLACE FUNCTION delivery()
```

```
RETURNS TABLE (a integer, b character varying(40), c timestamp without  
time zone, d boolean)
```

```
AS $$ SELECT * FROM "Доставка"; $$
```

```
LANGUAGE sql;
```

Для вызова SP надо ввести ее имя в том месте программы, в котором ожидается возвращаемое функцией значение. Для прямого запуска SP обычно используется SELECT или команда EXECUTE, например:

```
PREPARE orders AS SELECT orders();
```

EXECUTE orders;

Использование параметров в хранимых процедурах

Система передачи параметров позволяет создавать более гибкие SP, поскольку значения параметров определяются в момент вызова процедуры, а не во время ее компиляции. Имена параметров подчиняются общим требованиям для идентификаторов. Параметры могут быть именованными и неименованными, входными и выходными.

Входные параметры

Входные параметры используются для передачи значений в SP. По умолчанию параметр является входным. Явно определить входной параметр можно с помощью служебного слова IN. Для использования входного параметра в теле SP достаточно сослаться на него как на обычную

переменную с помощью выражения \$n, где n – порядковый номер параметра

в списке параметров, как указано в определении функции. Например:

/ SP для вставки строки в таблицу Заказы БД BookShop */*

```
CREATE OR REPLACE FUNCTION insert_order(custid character varying,  
bookid bigint, data_time timestamp without time zone, paid boolean) RETURNS  
bigint AS $$
```

```
INSERT INTO "Заказы"("Заказчик", "Дата_время", "Оплачен",  
"Код_книги") VALUES($1,$3,$4,$2) RETURNING $2; $$
```

```
LANGUAGE sql VOLATILE;
```

В эту SP должны передаваться четыре параметра, которые используются для формирования значений, добавляемых в столбцы таблицы Заказы. SP возвращает значение одного из своих входных параметров.

Для передачи параметров в SP можно использовать константы или выражения, имеющие значения соответствующего типа (например, выражения вызова функций или переменные), соблюдая при этом соответствие типов и порядок следования параметров, объявленных в SP. SP insert_order можно вызвать, например, так SELECT insert_orders(

```
'somebody'::character varying,  
100::bigint,  
now()::timestamp without time zone, true::boolean );
```

Параметры можно передавать в SP, используя имена параметров, объявленных в самой процедуре. В этом случае параметры можно передавать в любом порядке, но должны быть использованы имена для всех параметров. Возможен также комбинированный вариант, когда часть параметров передаются без имен, но они должны быть в начале списка параметров в выражении вызова SP и их позиции должны соответствовать позициям в определении SP, а часть - по именам. Еще один вариант вызова insert_orders выглядит следующим образом:

```
SELECT insert_orders(  
'somebody'::character varying,  
100::bigint,  
paid := true::boolean,  
data_time := now()::timestamp without time zone );
```

Входным параметрам при их объявлении могут быть присвоены значения по умолчанию. Тогда, если при вызове SP значения для таких параметров не указываются, то в процедуре будут использованы их значения по умолчанию. Если в SP для параметров указываются значения по умолчанию, то при вызове SP обычно используются имена параметров. Выходные параметры

Выходные параметры используются для возврата значений клиентскому приложению из SP, и это практически эквивалентно использованию возвращаемого значения. Более того, использовать выходные параметры совместно с RETURNS TABLE недопустимо.

Для объявления выходного параметра необходимо перед его именем вставить служебное слово OUT или INOUT.

Пример:

/ SP для удаления неоплаченных заказов из таблицы Заказы БД*

```
BookShop */CREATE OR REPLACE FUNCTION clear_unpaid_orders(
    OUT custid character varying, OUT data_time timestamp without time zone,
    OUT paid boolean, OUT orderid integer, OUT bookid bigint)
    RETURNS SETOF RECORD AS $$
    DELETE FROM "Заказы" WHERE "Оплачен" = false RETURNING *;
    $$ LANGUAGE sql VOLATILE;
```

В этой SP выходные параметры образуют столбцы набора записей, возвращаемых SP. В данном случае – это удаляемые строки. Чтобы SP возвращала не одну запись, а набор записей, в качестве типа возвращаемого значения надо указать SETOF RECORD. Тогда вызов такой SP может быть использован в предложении FROM инструкции SELECT. Например, в результате выполнения следующего набора команд CREATE TEMP TABLE deleted_orders (LIKE "Заказы") ON COMMIT DROP; INSERT INTO deleted_orders SELECT * FROM clear_unpaid_orders();

```
SELECT * FROM deleted_orders;
```

удаляемые с помощью clear_unpaid_orders строки загружаются во временную таблицу deleted_orders, которая создается по шаблону таблицы "Заказы" и удаляется после завершения транзакции.

PL/pgSQL функции

Создание PL/pgSQL функций

Недостатком SQL-функций является отсутствие средств управления программой. Процедурный язык PL/pgSQL дает возможность использовать в SQL-запросах стандартные методы программирования, такие как выполнение циклов и анализ значений переменных.

PL/pgSQL-функции имеют модульную структуру:

```
CrEaTE FUNCTION имя(параметры) REtUrNs тип AS $$
```

```
[<< метка_основного_блока>>]
```

```
DECLARE /* объявления */
```

```
BEGIN /* основной блок */
```

```
... ,
```

[<< метка_вложенного_блока>>]

DECLARE */* объявления */*

... ,

BEGIN */* вложенный блок */*

... ,

END [метка_вложенного_блока];

... ,

END [метка_основного_блока] ;

\$\$ LANGUAGE plpgsql;

Объявления переменных

Синтаксис выражения объявления переменных имеет вид: имя
[CONSTANT] тип [NOT NULL] [{ DEFAULT | := } выражение]; Примеры:

user_id integer DEFAULT 0; price numeric(5.2); name varchar (32) :='foo';

/ переменная row имеет тип строки таблицы table */* row
table%ROWTYPE;

/ переменная field имеет тип столбца column таблицы table */* field
table.column%TYPE;

/ переменная x типа запись (строка неопределенной структуры) */* x
RECORD;;

Операторы управления программой

Внутри PL/pgSQL-функции можно использовать любые инструкции SQL, а также операторы управления программой, к которым относятся:

- Операторы выхода из функции, возвращающей скалярное значение или набор значений RETURN, RETURN NEXT, RETURN QUERY
- Условные операторы IF...ELSE

CASE ... WHEN

- Циклические операторы LOOP – безусловный цикл WHILE – цикл с предусловием FOR – цикл со счетчиком EXIT – выход из цикла

CONTINUE – переход на следующую итерацию

- Вывод сообщений и сообщений об ошибках RAISE

Примеры:

/ Удаление неоплаченных заказов из таблицы Заказы БД BookShop */*

CREATE FUNCTION clear_unpaid_orders ()

RETURNS integer AS \$\$

DECLARE

n integer := 0;

BEGIN

– определяем количество неоплаченных заказов и записываем в n

SELECT count(*) INTO n FROM "Заказы" WHERE "Оплачен" = false;

– если таких заказов нет, завершаем вызов функции IF n =0 THEN

RETURN 0;

END IF;

– в противном случае – удаляем

DELETE FROM "Заказы" WHERE "Оплачен" = false;

– и выводим сообщение о количестве удаленных строк RAISE NOTICE

'Удалено % строк', n;

RETURN n;

END;

\$\$

LANGUAGE plpgsql;

Обработка ошибок в PL/pgSQL функциях

Ошибки, возникающие при выполнении команд в блоке, отслеживаются с помощью предложения EXCEPTION, синтаксис которого имеет вид:

BEGIN

... ; – тело блока EXCEPTION WHEN условие [OR условие ...] THEN

обработчик_исключения

[WHEN условие [OR условие...] THEN обработчик_исключения ...]

END;

Если возникает ошибка, то выполнение команд в блоке прерывается и

управление передается в раздел EXCEPTION, где производится поиск первого из условий, удовлетворяющих возникшей ошибке, и запускается соответствующий обработчик. После чего управление передается следующей после END команде. Если ни одно из условий не удовлетворяет ошибке, обработка исключения передается первому из объемлющих блоков, в котором есть раздел EXCEPTION. Если такого блока нет, выполнение функции останавливается и выводится сообщение об ошибке. Список кодов ошибок приведен в документации к PostgreSQL в Приложении А.

Для отслеживания ошибок при выполнении команд SQL в PostgreSQL можно использовать специальную системную переменную FOUND типа boolean. Эта переменная является локальной и меняет исходное значение false на true в следующих случаях:

- SELECT, UPDATE, INSERT, DELETE – если хотя бы одна строка включается в результат, т.е. выбирается, модифицируется, вставляется или удаляется;
- FOR – если выполняется хотя бы одна итерация цикла;
- RETURN QUERY, RETURN QUERY EXECUTE – если запрос возвращает хотя бы одну строку.

Примеры:

```
/* Добавление строки в таблицу Доставка БД BookShop */  
DROP TABLE "Доставка";  
CREATE TABLE "Доставка"  
( "Заказ" bigint NOT NULL,  
  "Курьер" character varying(40) DEFAULT 'foo'::character varying,  
  "Дата_время" timestamp without time zone DEFAULT now(),  
  "Доставлен" boolean DEFAULT false,  
  CONSTRAINT "PK_delivery" PRIMARY KEY ("Заказ") );  
CREATE FUNCTION insert_delivery( order_id bigint, courier character  
varying, date_time timestamp without time zone)  
RETURNS VOID AS $$ BEGIN
```

```

INSERT INTO "Доставка" ("Заказ", "Курьер", "Дата_время")
VALUES ($1,$2,$3);
RETURN;

/* нарушение ограничения уникальности для первичного ключа */
EXCEPTION WHEN unique_violation THEN LOOP
UPDATE "Доставка" SET "Курьер" = $2, "Дата_время" = $3 WHERE
"Заказ" = $1;
IF FOUND THEN RETURN;
END LOOP;
END; $$
LANGUAGE plpgsql;

```

В PostgreSQL не предусмотрен специальный отладчик процедурных функций, поэтому для отладки в таких случаях обычно используют операторы вывода сообщений об ошибках и переменные состояния.

Триггеры

Триггеры – это специальный тип функций, которые выполняются автоматически при выполнении инструкций языка манипулирования данными (INSERT, UPDATE, DELETE). Триггеры обычно используются для внесения каскадных изменений в связанные таблицы или для выполнения сложных ограничений, которые нельзя реализовать стандартными средствами SQL.

Триггеры делятся на два вида: триггеры (FOR EACH ROW), вызываемые для каждой из строк, участвующих в операции, и триггеры (FOR EACH STATEMENT), которые вызываются один раз для каждой операции. Триггеры также могут вызываться до (before-триггеры) начала выполнения операции и после (after-триггеры).

Триггеры создаются с помощью команды CREATE TRIGGER, в которой указывается, какая функция будет выполняться при вызове триггера. Эти функции могут создаваться на любом из поддерживаемых процедурных языков, например, на PL/pgSQL. Они должны быть определены без

параметров и с возвращаемым значением типа trigger. Когда такая функция запускается на исполнение, в ее основном блоке (блок самого верхнего уровня) создаются несколько переменных, в которых фиксируются параметры соответствующего триггера, и две буферные переменные:

NEW – переменная типа RECORD, в которой содержится новая строка для INSERT/UPDATE-триггеров типа FOR EACH ROW или значение NULL для триггеров типа FOR EACH STATEMENT и для INSERT/UPDATE-триггеров типа FOR EACH ROW;

OLD – переменная типа RECORD, в которой содержится старая строка для UPDATE/DELETE-триггеров типа FOR EACH ROW или значение NULL для триггеров типа FOR EACH STATEMENT и для INSERT-триггеров типа FOR EACH ROW.

Примеры:

```
/* Создаем таблицу */
```

```
CREATE TABLE emp ( a text NOT NULL,  
b integer NOT NULL CHECK (b>0), last_date timestamp, last_user text );
```

```
/* Создаем триггерную функцию */
```

```
CREATE FUNCTION stamp() RETURNS trigger AS
```

```
$stamp$
```

```
BEGIN
```

```
--Проверка a= null и b= null и b>0'
```

```
IF NEW.a IS NULL THEN
```

```
RAISE EXCEPTION 'a не может быть null !';
```

```
END IF;
```

```
IF NEW.b IS NULL THEN
```

```
RAISE EXCEPTION ' b не может быть null ! ', NEW.a;
```

```
END IF;
```

```
IF NEW.b < 0 THEN
```

```
RAISE EXCEPTION ' b = % должно быть положительным !', NEW.b;
```

```
END IF;
```

– Заносим в таблицу текущее время и текущего пользователя

```
NEW.last_date := current_timestamp;
```

```
NEW.last_user := current_user;
```

```
RETURN NEW;
```

```
END;
```

```
$stamp$ LANGUAGE plpgsql;
```

```
/* Создаем триггер */
```

```
CREATE TRIGGER stamp BEFORE INSERT OR UPDATE ON emp FOR  
EACH ROW EXECUTE PROCEDURE stamp();
```

Задания для самостоятельной работы

Создайте триггеры для таблиц БД BookShop:

№	Таблица	Параметры триггера
1	Книги	INSERT/UPDATE, FOR EACH ROW
2	Заказы	INSERT/UPDATE, FOR EACH STATEMENT
3	Заказчики	INSERT/UPDATE, FOR EACH STATEMENT
4	Доставка	INSERT/DELETE, FOR EACH ROW

Оборудование и материалы:

Для изучения материалов этой лабораторной работы необходимо:

1. Компьютер под управлением операционной системы Windows 8/10 с параметрами по умолчанию.
2. Установленные на компьютер программы Microsoft Visio, PostgreSQL, MySQL, VMWare/VirtualBox.
3. Текстовый процессор Microsoft Word 2010/2013/2016.

Указания по технике безопасности:

К выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж по технике безопасности.

Задания:

1. Создайте таблицу User_SP для хранения списка хранимых процедур в БД BookShop. Таблица должна содержать следующие поля:

- Имя SP
- Владелец

- Описание
- Входные параметры
- Выходные параметры
- Другие результаты
- Вызывается из (список вызывающих SP)
- Вызывает (список вызываемых SP)
- История обновлений (с момента создания):

Создайте набор SP, предназначенных для вставки, удаления и модификации таблицы User_SP.

Указать: При удалении функции список параметров должен быть специфицирован (PostgreSQL поддерживает перегрузку функций). При удалении функции должны быть удалены все связанные с ней объекты.

2. Создайте SP, предназначенные для генерации в БД BookShop отчетов, приведенных в таблице 8.1.

Таблица 8.1 – Задания для генерации отчетов в БД BookShop

Роль	Отчеты
Директор	– Отчет по продажам: показывает тенденции изменения спроса за последние 6 месяцев для книг, объем продаж которых за указанный период изменился более, чем на 100%.
Закупщик	– Отчет по поставщикам: показывает тенденции изменения цен с начала текущего года для поставщиков, объемы поставок которых за указанный период не уменьшились.

Содержание отчета:

Отчет о выполнении лабораторной работы должен содержать:

- название и тему лабораторной работы; цель лабораторной работы; краткие теоретические сведения;
- ход выполнения работы;
- выводы.

Раздел «Ход выполнения работы» должен содержать инфологическую модель БД на языке таблица-связь, SQL-скрипты созданных хранимых

процедур и триггеров и результаты их тестирования (для триггеров обязательно тест на обработку исключений).

Контрольные вопросы:

В данном разделе приведено лишь несколько примеров контрольных вопросов, остальные будут приблизительно такого же содержания.

1. Какие типы функций поддерживает PostgreSQL?
2. Чем отличаются внутренние функции от C-функций?
3. Что такое хранимые процедуры?
4. Почему хранимые процедуры работают быстрее, чем операторы SQL, непосредственно передаваемые серверу SQL?
5. Для чего используются хранимые процедуры?
6. Можно ли создать временную таблицу внутри хранимой процедуры? Какие еще команды SQL нельзя выполнять внутри хранимой процедуры?
7. Как удалить или переименовать хранимую процедуру?
8. Могут ли пользователи манипулировать данными, если они не имеют соответствующих прав доступа к этим данным? Если могут, то каким образом?
9. Как можно вернуть некоторое значение из хранимой процедуры в вызывающее приложение? Какого типа могут быть эти значения?
10. Как можно передать входные параметры в хранимую процедуру?
11. Как надо определить хранимую процедуру, чтобы можно было использовать ее вызов в предложении FROM инструкции SELECT?
12. Какова структура PL/pgSQL-функции?
13. Как в PL/pgSQL-функциях осуществляется обработка ошибок?
14. Как производится отладка PL/pgSQL-функций?
15. Что такое триггеры? Для чего они используются?
16. Какие существуют типы триггеров?
17. Как создаются и работают триггеры?

Список литературы, рекомендуемый к использованию по данной теме:

1. Братченко, Н. Ю. (Северо-Кавказский федеральный университет). Базы данных. Основы работы с СУБД ORACLE 10G EXPRESS EDITION: учеб. пособие: Направление подготовки 230100 – Информатика и вычислительная техника / Н. Ю. Братченко; Сев.-Кав. федер. ун-т. – Ставрополь: СКФУ, 2013. – 202 с.: ил. – Гриф: Доп. УМО. – ISBN 978-5-88648-877-7

2. Разработка приложений на С с использованием СУБД PostgreSQL / И.А. Васюткина. – Новосибирск: НГТУ, 2015. – 143 с. – ISBN 978-5-7782-2699-9
3. Шацков, В.В. Программирование приложений баз данных с использованием СУБД MS SQL Server / В.В. Шацков: учебное пособие. Электронный ресурс: Санкт-Петербургский государственный архитектурно-строительный университет, ЭБС АСВ; Санкт-Петербург, 2015. – 80 с. – Книга находится в базовой версии ЭБС IPRbooks. – ISBN 978-5-9227-0607-0
4. Голицина, О. Л. Системы управления базами данных: [учеб. пособие] / О.Л. Голицына, Т.Л. Партыка, И.И. Попов. – М.: ФОРУМ, 2011. – 432 с.: ил. – (Профессиональное образование). – На учебнике гриф: Рек.МО. – Библиогр.: с. 315. – ISBN 5-8199-0251-3
5. Гудов, А. М. Базы данных и системы управления базами данных. Программирование на языке PL/SQL / А.М. Гудов; С.Ю. Завозкин; Т.С. Рейн. – Кемерово: Кемеровский государственный университет, 2010. – 134 с. – ISBN 978-5-8353-1005-0
6. Харченко М. В. Организация баз данных. Методические указания к выполнению лабораторных работ для студентов направления 6.050102 – «Компьютерная инженерия». – Чернигов, ЧГТУ. 2013 – 141 с

Практическая работа №4. Создание программного приложения для работы с базой данных

Цель работы: средствами SDK QT 4 создать программное приложение, которое обеспечивает работу с базой данных (БД). Изучить способы подключения к БД, представления данных из БД, выполнения простых и параметрических запросов, выполнения команд ЯМД.

Теоретическая часть

Система Interview Framework в SDK Qt 4

Система Interview Framework в SDK Qt 4 представляет собой вариант реализации парадигмы «модель-контроллер-вид». В основе парадигмы «модель-контроллер-вид» лежит старая и плодотворная идея разделения «движка» программы и интерфейса. В рамках парадигмы «модель-контроллер-вид» (подробно описанной в многочисленной литературе по «правильному» программированию) модель представляет собой, по сути, «движок» приложения. Именно модель определяет, что и как программа может делать. Термином вид (представление) фактически описывается все, что имеет непосредственное отношение к интерфейсу пользователя. Вид позволяет пользователю получать информацию о состоянии модели и передавать программе команды. Команды пользователя обрабатывает контроллер, который вносит соответствующие изменения в состояние модели или вида и, в частности, не позволяет пользователю нарушить целостность модели в результате введения неправильных команд. Как и многие другие парадигмы, призванные формализовать процесс создания программ, парадигма «модель-контроллер-вид» редко применяется на практике в чистом виде. В частности, отдельные элементы парадигмы нередко объединяются друг с другом. Interview Framework превращает парадигму «модель-контроллер-вид» в парадигму «модель-вид», объединяя контроллер и вид в одно целое. Хотя парадигма «модель-контроллер-вид» (а, следовательно, и Interview Framework) может применяться при написании множества типов программ, она исключительно удобна именно для создания клиентских приложений для работы с базами данных. Поэтому на примере клиентского приложения БД проще всего понять, как работает Interview Framework. В клиентском приложении БД, использующем Interview Framework, модель выполняет роль посредника между БД и интерфейсом пользователя, представляющим данные БД. Именно модель определяет логику представления данных. Когда пользователь хочет получить информацию о текущем состоянии БД, пользовательский интерфейс (компонент «вид») обращается к модели напрямую. Для работы с

отдельными элементами данных служат делегаты, которые передают команды пользователя модели.

В основе системы Interview Framework лежат три абстрактных класса `QAbstractItemModel`, `QAbstractItemView` и `QAbstractItemDelegate`. Эти классы являются предками всех классов, реализующих, соответственно, модели, представления (виды) и делегаты.

Классы `QTableView`, `QTreeView` и `QListView`, являющиеся потомками `QAbstractItemView`, реализуют три наиболее популярные формы представления данных: таблицу, дерево и простой список. Если разработчику требуется более сложный компонент отображения модели, ему придется создавать собственный класс, основанный на `QAbstractItemView`.

Среди потомков `QAbstractItemModel` отметим `QStandardItemModel`, `QDirModel`, `QStringListModel`, `QAbstractTableModel`, `QAbstractListModel`, `QSqlQueryModel`, `QSqlTableModel` и `QSqlRelationalTableModel`.

Класс `QStandardItemModel` представляет собой реализацию модели в самом общем смысле. Помимо прочего, этот класс реализует ряд методов, предназначенных для работы с индексами. Индексы используются в Interview Framework для указания элементов данных, с которыми работает модель.

Класс `QDirModel` реализует модель для работы с директориями. Этот класс будет полезен при написании собственного файл-менеджера или нестандартной версии диалоговых окон открытия и сохранения файла. Следует отметить, что один и тот же объект, реализующий модель, может взаимодействовать (в том числе, одновременно) с объектами нескольких разных классов, отвечающими за представление данных. Например, `QDirModel` может использоваться для представления информации о директориях классы `QTableView`, `QTreeView`, и `QListView`.

Класс `QStringListModel`, как можно догадаться по его названию, реализует модель, основой которой является список строк.

Классы `QAbstractTableModel` и `QAbstractListModel` могут служить основой для классов-моделей, предполагающих представление данных в виде

таблиц и списков соответственно. Такие классы как QTreeView и QListView предназначены для работы с моделями, но использовать их в качестве самостоятельных виджетов затруднительно. Для решения этой проблемы на базе классов QTableView, QTreeView и QListView созданы классы QTableWidget, QTreeWidget и QListWidget. Объекты этих классов представляют собой обычные визуальные компоненты, при работе с которыми пользователь может добавлять и удалять данные, не заботясь о моделях и делегатах. На самом деле, эти классы просто реализуют свои собственные модели данных, незаметные для пользователя.

Классы QSqlQueryModel, QSqlTableModel и QSqlRelationalTableModel относятся к специальным классам для работы с базами данных и предназначены для создания модели приложения БД. Из этих трех классов QSqlQueryModel – самый простой. Его возможности, фактически, ограничиваются передачей результата запроса к базе данных. При этом, правда, стоит отметить, что класс QSqlQueryModel обладает определенными возможностями, позволяющими изменить структуру отображения данных перед передачей ее на уровень представления. Класс QSqlTableModel гораздо функциональнее. Этот класс логически организует результаты SQL-запросов как таблицы и включает в себя функции редактирования данных. Наконец класс QSqlRelationalTableModel позволяет задействовать в приложении основные возможности реляционной модели баз данных – работу с данными из нескольких таблиц, связанных внешними ключами. Для представления данных моделей SQL наиболее логично использовать объекты класса QTableView (хотя унифицированная структура Interview Framework позволяет использовать совместно с SQL-моделями и другие стандартные «виды», они, как правило, менее удобны и информативны при работе с данными БД).

Для создания программного приложения, взаимодействующего с БД, в среде разработки Qt 4 существует специальный модуль – QtSql, который обеспечивает независимый от платформы и типа СУБД интерфейс для

доступа с помощью языка SQL к базам данных. Этот интерфейс поддерживается набором классов, использующих архитектуру Qt модель/представление для интеграции средств доступа к БД с интерфейсом пользователя.

Классы модуля QtSql делятся на три уровня:

- уровень драйверов;
- программный уровень;
- уровень пользовательского интерфейса.

К первому уровню относятся классы для получения данных на физическом уровне. Это такие классы, как QSqlDriver, QSqlDriverCreator<T*>, QSqlDriverCreatorBase, QSqlDriverPlugin и QSqlResult. Данные классы используются при написании разработчиками собственных драйверов БД.

Классы второго уровня предоставляют программный интерфейс для обращения к БД. К ним относятся: QSqlDatabase, QSqlQuery, QSqlError, QSqlField, QSqlIndex и QSqlRecord.

Третий уровень предоставляет модели для отображения результатов запросов. Эти модели представлены следующими классами: QSqlQueryModel, QSqlTableModel и QSqlRelationalTableModel.

Связь с БД обеспечивается объектом класса QSqlDatabase. Qt использует драйверы для связи с программным интерфейсом различных баз данных. Версия Qt для настольных компьютеров (Qt Desktop Edition) включает в себя драйверы, приведенные в таблице 9.1.

Таблица 9.1 – Драйверы, поддерживаемые QT

Драйвер	База данных
QDB2	IBM DB2 версии 7.1 и выше
QIBASE	InterBase компании Borland
QMYSQL	MySQL
QOCI	Oracle (Oracle Call Interface – интерфейс вызовов Oracle)
QODBC	ODBC (включая Microsoft SQL Server)
QPSQL	PostgreSQL версий 6.x и 7.x
QSQUTE	SQLite версии 3 и выше

QSQUTE2	SQLite версии 2
QTDS	Sybase Adaptive Server

Из-за лицензионных ограничений не все драйверы входят в состав Qt с открытым исходным кодом (Qt Open Source Edition). Если драйвер не поставляется с Qt, то можно его собрать.

Ниже описывается как с помощью ODBC можно настроить источник данных и подключиться к БД.

Использование ODBC для подключения источника данных
ODBC (Open DataBase Connectivity) – это программный интерфейс (API) доступа к БД. ODBC позволяет единообразно оперировать с разными источниками данных, отвлекаясь от особенностей взаимодействия в каждом конкретном случае.

ODBC API позволяет разработчикам создавать приложения, работающие с БД разных производителей.

Для настройки подключения источника данных с помощью ODBC необходимо выполнить следующие действия.

1. В Пуск→Панель управления→Администрирование (рисунок 9.1) выбрать Источники данных (ODBC). В результате откроется окно Администратор источников данных ODBC (рисунок 9.2).

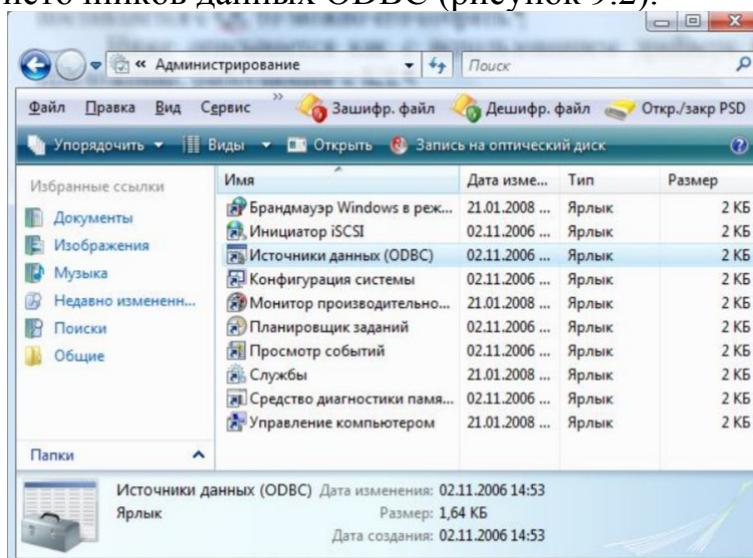


Рисунок 9.1 – Панель «Администрирование» Windows

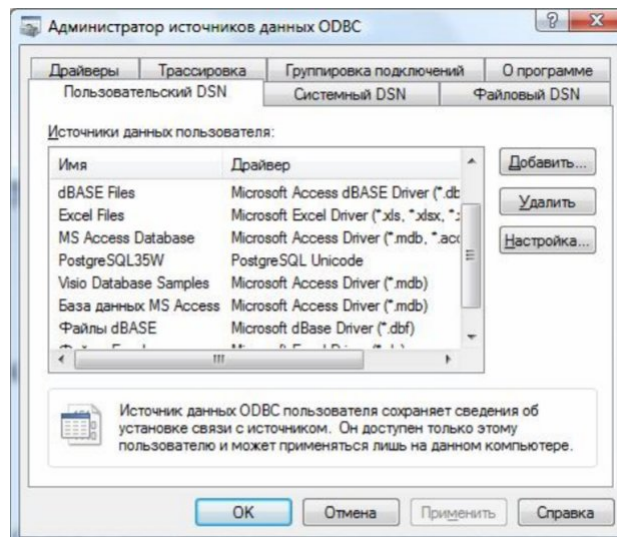


Рисунок 9.2 – Администратор источников данных ODBC

2. На вкладке Пользовательский DNS необходимо нажать кнопку Добавить и выбрать нужный драйвер. Так как в данной и предыдущих лабораторных работах использовалась БД PostgreSQL, то необходимо выбрать драйвер с именем PostgreSQL Unicode или PostgreSQL ANSI (рисунок 9.3) и нажать кнопку Готово.

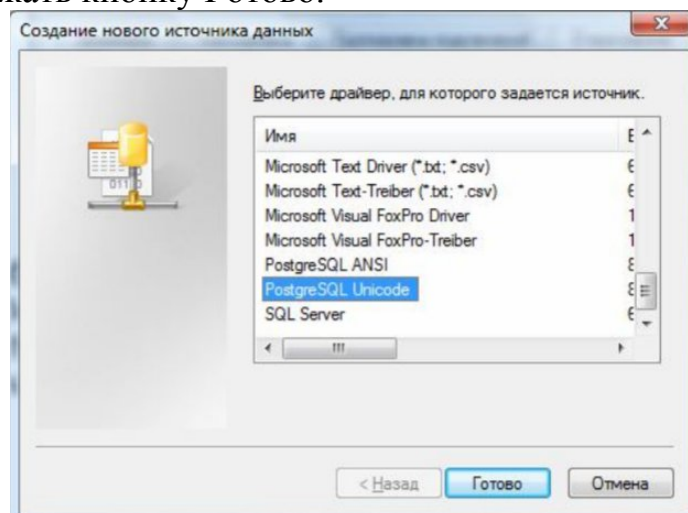


Рисунок 9.3 – Создание нового источника данных

3. В открывшемся окне параметров подключения (рисунок 9.4) необходимо указать: имя источника данных (Data Source); имя БД, с которой устанавливается соединение (Database); имя сервера, на котором работает PostgreSQL (Server); порт (Port); имя пользователя БД (User Name) и пароль (Password), с использованием которых будет осуществляться подключение к БД (это значит, что на сервере с PostgreSQL должен быть создан пользователь именно с таким именем и паролем).

Параметр Data Source определяет внешний (на уровне операционной системы) псевдоним (алиас) БД. Этот псевдоним скрывает путь к реальной БД, что позволяет изменять путь к конкретной БД не изменяя других настроек конкретных приложений.

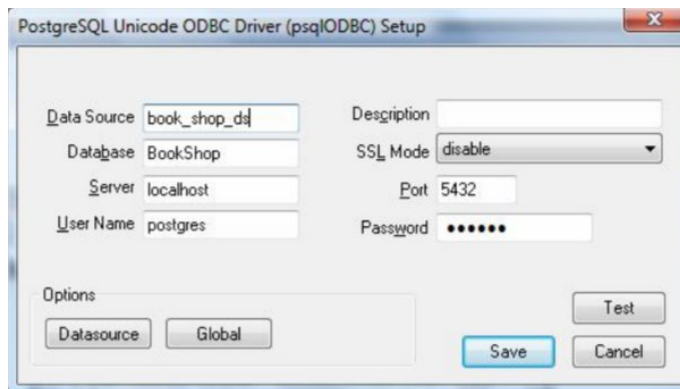


Рисунок 9.4 – Параметры подключения источника данных

Чтобы проверить успешно выполнится подключение или нет, можно нажать кнопку Test.

После сохранения введенных параметров, новый источник данных успешно добавится к существующим (рисунок 9.5).

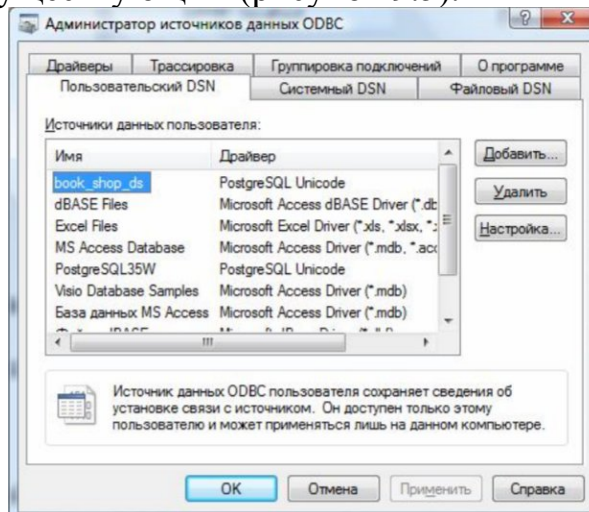


Рисунок 9.5 – Администратор источников данных ODBC

В результате выполнения этих действий создается внешний ODBC-псевдоним БД, после чего можно приступать к созданию программного приложения для работы с БД.

Соединение с базой данных

Перед тем, как выполнять какие-либо действия с БД, нужно сначала установить с ней соединение. Обычно настройка соединения с БД выполняется отдельной функцией, которую вызывают при запуске приложения. Например:

```

bool createConnection(QSqlDatabase db, QString dbName, QString host, QString
                      usr, QString pwd)
{
    db = QSqlDatabase::addDatabase("QPSQL");
    db.setDatabaseName(dbName);
    db.setHostName(host);
    db.setUserName(usr);
    db.setPassword(pwd);
    if (!db.open())
    {
        return false;
    }
    return true;
}

```

Здесь вызывается метод `QSqlDatabase::addDatabase()`, который создает объект `QSqlDatabase`. Первый аргумент задает драйвер БД, который Qt должна использовать для доступа к базе данных. В данном случае используется драйвер `QPSQL`.

Затем устанавливается название БД, имя сервера, на котором работает PostgreSQL и размещена БД, имя пользователя и пароль (методы `setDatabaseName()`, `setHostName()`, `setUserName()` и `setPassword()` соответственно).

Если использовать `QODBC` драйвер, то имя сервера не требуется, а вместо названия БД указывается `ODBC`-псевдоним. Например:

```

db = QSqlDatabase::addDatabase("QODBC");
db.setDatabaseName("book_shop_ds");
db.setUserName("postgres");
db.setPassword("pwd");
if (!db.open())
{
    return false;
}
return true;

```

После определения параметров подключения открывается соединение с БД – метод `open()`. Если данный метод завершается неудачей, функция `createConnection()` возвращает значение `false`, иначе – `true`.

Ниже приводится пример вызова функции `createConnection()`.

```

void MainWindow::on_action_triggered()
{
    if (!createConnection(db, "BookShop", "localhost", "postgres", "qwerty"))
    {
        QMessageBox::warning(this, "Error", "Can't create connection!");
    }
    else
    {
        QMessageBox::information(this, "Ok", "Connection successful!");
    }
}

```

Во всех последующих примерах используется БД “BookShop”, схема которой приведена на рисунке 9.6.

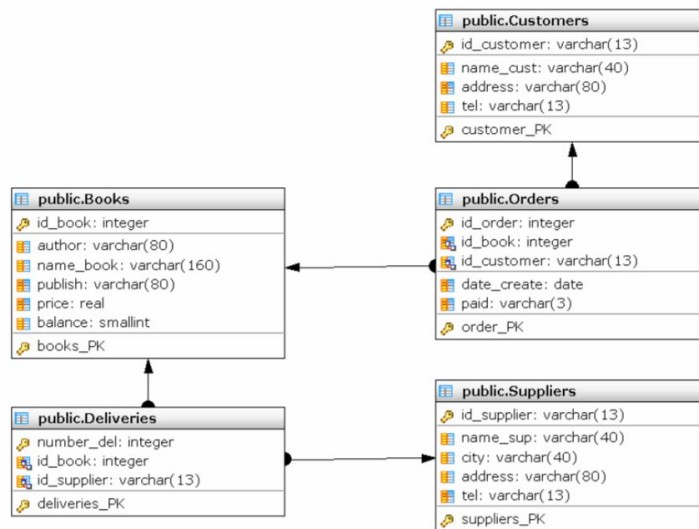


Рисунок 9.6 – Схема БД “BookShop”

Примечание:

Для использования модуля QSql необходимо в файл проекта (файл с расширением .pro) добавить следующую строку:

QT += sql

А для того, чтобы иметь возможность работать с классами этого модуля, необходимо включить заголовочный метафайл QSql:

#include <QSql>

Выполнение непараметрического SQL-запроса Select

После успешного соединения с БД можно выполнять запросы SQL и обрабатывать их результаты. Для этого используется класс QSqlQuery. Ниже приводится пример функции выполнения команды SELECT и обработки ее результатов:

```

void doSqlQuery(QString queryStr, QListWidget *listWidget)
{
    QSqlQuery query;
    if (!query.exec(queryStr))
    {
        qDebug() << "Can't do query!";
    }
    else
    {
        QSqlRecord rec = query.record();
        qDebug() << "Number of columns: " << rec.count();
        QString str;
        for (int i = 0; i < rec.count(); i++)
        {
            str = str+"\t"+rec.fieldName(i);
        }
        listWidget->addItem(str);
        while (query.next())
        {
            int i=0;
            QString strField;
            while (query.value(i).isValid())
            {
                strField = strField+"\t"+ query.value(i).toString();
                i++;
            }
            listWidget->addItem(strField);
        }
    }
}
  
```

Для выполнения запросов к БД используется метод `exec()`, которому в качестве параметра передается строка SQL-запроса. После его вызова можно просмотреть результат команды `SELECT`. Метод `next()` вызывается один раз для позиционирования объекта `QSqlQuery` на первую запись полученного набора данных (под набором данных понимается группа записей из одной или нескольких таблиц физической БД, полученная в результате выполнения SQL-запроса). Последующие вызовы `next()` продвигают указатель записи на одну позицию дальше, пока не будет достигнут конец набора данных и метод `next()` вернет `false`. Если результирующий набор данных (result set) пустой (или запрос завершается неудачей), первый вызов метода `next()` возвратит `false`.

Метод `value(номер_столбца)` возвращает значение поля типа `QVariant`, которое надо преобразовать к нужному типу с помощью методов `QVariant::toInt`, `QVariant::toLongLong`, `QVariant::toString`,

`QVariant::toDouble`, `QVariant::toDate`, `QVariant::toDateTime` и т.д. Поля пронумерованы начиная с 0 в порядке их указания в команде `SELECT`. Поскольку доступ по имени поля не предусмотрен, следует избегать запросов типа `SELECT *`, так как порядок полей в результирующем наборе не очевиден.

Кроме `next()`, для навигации по набору данных можно использовать методы `first()`, `last()`, `previous()`, `seek(int index, bool relative=false)`. Для

увеличения быстродействия набор данных лучше сделать однонаправленным, вызвав метод `QSqlQuery::setForwardOnly(true)` до выполнения запроса, после этого можно использовать только метод `next()`.

Вывод результата запроса происходит в компонент `ListWidget`, указатель на который передается в функцию (также можно выводить данные в компонент `TableWidget`). Ниже приводится пример вызова функции `doSqlQuery()`:

```
void MainWindow::on_pushButton_clicked()
{
    QString str = "Select * From \"Books\"";
    doSqlQuery(str, ui->listWidget);
}
```

На рисунке 9.7 показан пример интерфейса приложения для выполнения непараметрического запроса.

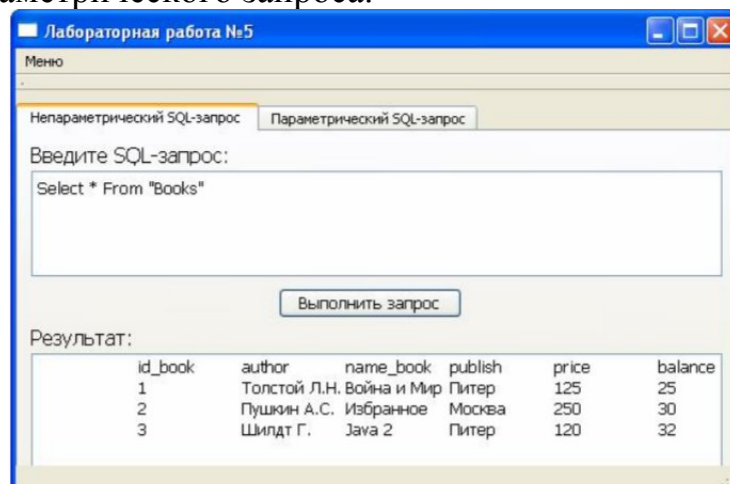


Рисунок 9.7 – Пример интерфейса приложения для выполнения непараметрических SQL-запросов Select

Кроме метода `QSqlQuery::exec()` SQL-запрос можно передавать в качестве аргумента в конструктор, который сразу же выполнит его. Например:

```
QSqlQuery query("SELECT author, name FROM \"Books\"  
                WHERE price >= 50").
```

Чтобы проверить наличие ошибки в результате выполнения запроса, вызывается метод `isActive()`:

```
if (! query.isActive())  
    QMessageBox::warning(this, tr("Database Error"),  
                          query.lastError().text());
```

Если ошибки нет, запрос становится активным и можно использовать метод `next()` для перемещения по результирующему набору данных.

Выполнение параметрических SQL-запросов Select

Как известно, SQL-сервер БД обладает средствами для предварительного разбора и оптимизации запросов, причем с кэшированием просчитанных сценариев. К оптимизации относится построение плана запроса, профилировка, фоновое создание дополнительных индексов и т.д. Этот процесс называется загрузкой, или подготовкой запроса. Для загрузки запроса в QT служит метод `QtSqlQuery::prepare()`.

Передаваемая строка-запрос может содержать ряд параметров для последующей параметризации запроса (нужно понимать, что каждый параметр потенциально снижает уровень оптимизации). Параметры могут быть заданы в двух форматах: именованные в стиле Oracle (`:parameter`) и вопросительные знаки ODBC (как в MS Access). Смешивание двух форматов не допускается.

В зависимости от техники задания параметров, можно подставлять их по имени или по позиции. Подстановка производится в подготовленный запрос. Для подстановки по имени используется один из перегруженных методов, из категории

`QSqlQuery::bindValue(QString, QVariant, QSql::ParameterType)`. Имя переменной указывается вместе с двоеточием, т.е. так же, как она была задана в запросе.

Для подстановки по порядковому номеру используются методы `bindValue(int, QVariant)`

или

`addBindValue(QVariant, QSql::ParameterType)`.

Последний метод просто формирует список параметров, добавляя по одному за раз – при этом нелегко понять, какой параметр добавляется в данный момент. Метод именованных параметров дает значительно более читабельный код. Ниже приводятся примеры использования именованной подстановки параметров, и подстановки параметров по номеру.

Для именованных параметров

```
QSqlQuery query();  
query.prepare("SELECT author, name FROM \"Books\"  
              WHERE price >= :param");  
query.bindValue(":param", 50);  
query.exec();
```

Подстановка параметров по номеру..

```
QSqlQuery query();  
query.prepare("SELECT author, name FROM \"Books\"  
              WHERE price >= ?");  
query.addBindValue(50);  
query.exec();
```

Ниже приводится пример функции выполнения простого параметрического SQL-запроса «Получить список книг издательства X, поставляемых поставщиком Y».

```
bool doQueryWithParam (QString param1, QString param2, QSqlQuery *query)  
{  
    query->prepare("SELECT author, name_book, price FROM \"Books\", "  
                  "\"Deliveries\", \"Suppliers\" "  
                  "WHERE \"Books\".id_book = \"Deliveries\".id_book and "  
                  "\"Deliveries\".id_supplier = \"Suppliers\".id_supplier and "  
                  "\"Books\".publish = :X and \"Suppliers\".name_sup = :Y");  
    query->bindValue(":X", param1);  
    query->bindValue(":Y", param2);  
    return query->exec();  
}
```

На рисунке 9.8 показан простой пример интерфейса приложения для выполнения параметрического запроса.

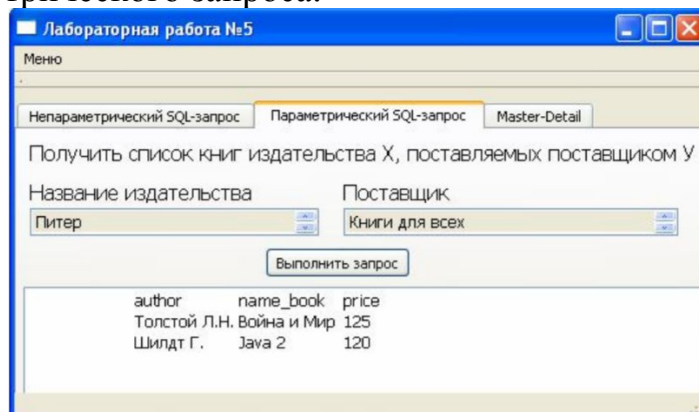


Рисунок 9.8 – Пример интерфейса приложения для выполнения параметрических SQL-запросов Select

Выполнение параметрических SQL-запросов Insert, Update, Delete

Помимо простых запросов на выборку данных, можно также выполнять запросы на их модификацию (Insert, Update, Delete). Ниже приводятся примеры запросов для добавления, удаления и изменения информации о заказчиках (таблица “Customers” БД “BookShop”).

«Добавить данные о заказчике»

```

bool addCustomer (int id, QString name, QString addr, QString telephone)
{
    QSqlQuery query;
    query.prepare("INSERT INTO \"Customers\" (id_customer, name_cust,
        address, tel) VALUES (:p1, :p2, :p3, :p4)");
    query.bindValue(":p1", id);
    query.bindValue(":p2", name);
    query.bindValue(":p3", addr);
    query.bindValue(":p4", telephone);
    if(!query.exec() || (query.numRowsAffected() == 0))
    {
        qDebug() << "ERROR in INSERT command";
        return false;
    }
    else
    {
        return true;
    }
}

```

«Удалить данные о заказчике».

```

bool delCustomerById (int id)
{
    QSqlQuery query;
    query.prepare("DELETE FROM \"Customers\" WHERE id_customer= :p1");
    query.bindValue(":p1", id);

    if(!query.exec() || (query.numRowsAffected() == 0))
    {
        qDebug() << "ERROR in DELETE command";
        return false;
    }
    else
    {
        return true;
    }
}

```

В качестве параметра в функцию передается идентификатор заказчика, которого нужно удалить из БД.

«Изменить данные о заказчике»

```

bool updateCustomer (int id, int newId, QString newName, QString newAddr,
    QString newTel)
{
    QSqlQuery query;
    query.prepare("UPDATE \"Customers\" SET id_customer = :new_id, name_cust=
        :new_name, address= :new_addr, tel = :new_tel WHERE
        id_customer= :p1");
    query.bindValue(":new_id", newId);
    query.bindValue(":new_name", newName);
    query.bindValue(":new_addr", newAddr);
    query.bindValue(":new_tel", newTel);
    query.bindValue(":p1", id);
    if(!query.exec() || (query.numRowsAffected() == 0))
    {
        qDebug() << "ERROR in UPDATE command";
        return false;
    }
    else
    {
        return true;
    }
}

```

В качестве параметра в функцию передается идентификатор заказчика, информацию о котором нужно изменить, и новые данные.

Из приведенных выше примеров следует обратить внимание на метод `QSqlQuery::numRowsAffected()`. При выполнении SQL-запросов `INSERT`, `UPDATE` или `DELETE` этот метод возвращает число успешно добавленных, обновленных или удаленных записей.

Выполнение транзакций

Под транзакцией принято понимать последовательность операторов манипулирования данными, выполняющуюся как единое целое (все или ничего) и переводящую базу данных из одного непротиворечивого состояния в другое непротиворечивое состояние.

Если СУБД поддерживает механизм транзакций, то для начала новой транзакции в Qt используется метод `bool QSqlDatabase::transaction()`. Для подтверждения транзакции надо вызвать `bool QSqlDatabase::commit()` а для отмены: `bool QSqlDatabase::rollback()`.

Если СУБД не поддерживает транзакций, то вызовы `transaction`, `commit` и `rollback` ничего не делают. С помощью метода

`QSqlDriver::hasFeature()` можно узнать, поддерживается ли данным драйвером и СУБД та или иная функция, в том числе и транзакции:

```
QSqlDriver *driver = QSqlDatabase::database().driver();  
if (driver->hasFeature(QSqlDriver::Transactions))
```

Каждое соединение с базой данных может иметь только одну активную транзакцию. Если этого недостаточно, всегда можно открыть ещё несколько соединений с той же базой данных.

Ниже приведен пример, как можно найти внешние ключи для таблицы заказов «Orders» и выполнить команду `INSERT` внутри транзакции. Подробное описание всех действий приводится в комментариях.

```

void addOrder(QSqlDatabase db, QString nameCust, QString nameBook, int
idOrder, QString dateCreate, QString paipOrd)
{
    int bId=0;           // Идентификатор книги
    QString cId=0;       // Идентификатор заказчика

    // Вызов функции transaction() для запуска SQL-транзакции. Эта функция
    // вызывается у объекта QSqlDatabase, представляющего соединение с БД
    db.transaction();

    // По названию книги получаем ее идентификатор (внешний ключ для
    // таблицы Orders)
    QSqlQuery query;
    query.prepare("SELECT id_book FROM \"Books\" WHERE name_book = :p1");
    query.bindValue(":p1", nameBook);
    query.exec();
    if(query.next())
    {
        bId = query.value(0).toInt();
    }

    // По имени заказчика получаем его идентификатор (внешний ключ для
    // таблицы Orders)
    query.prepare("SELECT id_customer FROM \"Customers\" WHERE name_cust=
:p2");
    query.bindValue(":p2", nameCust);
    query.exec();
    if(query.next())
    {
        cId = query.value(0).toString();
    }

    //Формирование параметрического запроса
    query.prepare("INSERT INTO \"Orders\"(id_order, id_book,
date_create, paid, id_customer)
VALUES( :ordId, :bId, :data, :paid, :custId");

    // Присвоение значений параметрам
    query.bindValue(":ordId", idOrder);
    query.bindValue(":bId", bId);
    query.bindValue(":data", dateCreate);
    query.bindValue(":paid", paipOrd);
    query.bindValue(":custId ", cId);

    // Выполнение запроса
    if (!query.exec())
    {
        qDebug() << " ERROR in INSERT command";
        // Если возникла ошибка при выполнении запроса, то происходит
        // откат транзакции
        db.rollback();
    }
    else
    {
        // Если ошибок не было - подтверждение транзакции
        db.commit();
    }
}

```

Разработка форм для просмотра таблиц в режиме «главный-детальный»

Во многих случаях табличное представление является самым простым и наглядным представлением набора данных для пользователя. В данном

разделе на примере таблиц “Customers” и “Orders” БД «BookShop» реализовано такое представление.

На главной форме приложения показано представление «главный-детальный» для заказчиков и всех заказов конкретного заказчика (рисунок 9.9).

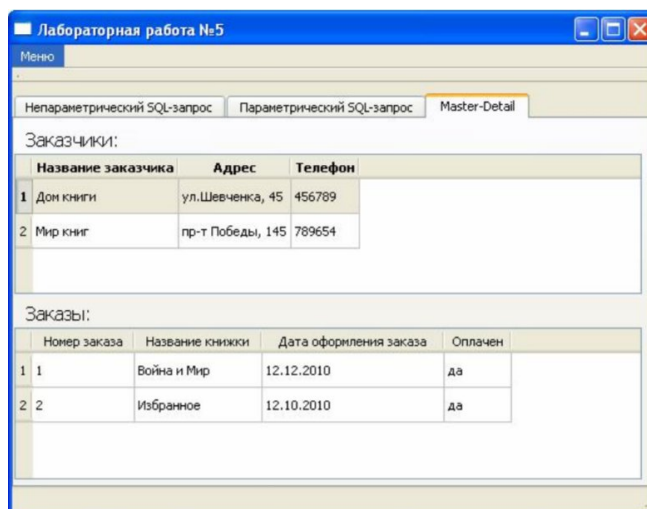


Рисунок 9.9 – Пример интерфейса для просмотра таблиц в режиме «главный-детальный»

Для отображения данных из БД в компонентах QTableView и QListView используется класс модели QSqlTableModel и его подкласс QSqlRelationalTableModel.

Модель QSqlTableModel используется для таблицы заказчиков “Customers”, а для таблицы заказов “Orders” – модель

QSqlRelationalTableModel, поскольку необходимо осуществлять работу с внешними ключами.

После установления соединения с БД, в том же методе можно реализовать просмотр таблиц в режиме «главный-детальный». Ниже приводится код с подробными комментариями.

```

void MainWindow::on_action_3_triggered()
{
    if (!createConnection(db, "BookShop", "localhost", "postgres", "qwerty"))
    {
        .....
    }

    // Вначале необходимо создать модель QSqlTableModel, которая управляет
    // таблицей заказчиков «Customers»
    customModel = new QSqlTableModel;
    // Далее осуществляется связывание этой модели с таблицей «Customers»
    // БД.
    customModel->setTable("\"Customers\"");
    // Установка сортировки по столбцу 1 (Имя заказчика). Нумерация
    // столбцов начинается с 0
    customModel->setSort(0, Qt::AscendingOrder);
    // Установка заголовков столбцов в модели. Т.к. идентификатор
    // заказчика выводится не будет, то для него заголовок
    // не формируется.
    customModel->setHeaderData(0, Qt::Horizontal,
                               QString::fromLocal8Bit("Название заказчика"));
    customModel->setHeaderData(1, Qt::Horizontal,
                               QString::fromLocal8Bit("Адрес"));
    customModel->setHeaderData(2, Qt::Horizontal,
                               QString::fromLocal8Bit("Телефон"));

    // Получение данных.
    customModel->select();

    // Связь QTableView с моделью
    ui->tableView->setModel(customModel);
    // Не показывать поле "id_customer"
    ui->tableView->setColumnHidden(3, true);
    // Параметры выбора элементов в QTableView
    ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);
    ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
    // Установка такой ширины столбцов, которой будет достаточно для
    // размещения в них текста без необходимости вывода многоточия
    ui->tableView->resizeColumnsToContents();
    // Для таблицы Заказы используется модель QSqlRelationalTableModel
    orderModel = new QSqlRelationalTableModel;
    // Связывание модели с таблицей заказов «Orders»
    orderModel->setTable("\"Orders\"");
    // Вызов метода setRelation() указывает модели на то, что ее поле
    // id_book, индекс которого равен 1, содержит идентификатор
    // id_book внешнего ключа из таблицы книг «Books» и что вместо
    // идентификаторов необходимо выводить на экран содержимое поля
    // name_book
    orderModel->setRelation(1, QSqlRelation("\"Books\"", "id_book",
                                             "name_book"));
    // Установка сортировки по столбцу 0 (Код заказа).
    orderModel->setSort(0, Qt::AscendingOrder);
    // Установка заголовков столбцов в модели.
    orderModel->setHeaderData(0, Qt::Horizontal,
                               QString::fromLocal8Bit("Номер заказа"));
    orderModel->setHeaderData(1, Qt::Horizontal,
                               QString::fromLocal8Bit("Название книжки"));
    orderModel->setHeaderData(2, Qt::Horizontal, QString::fromLocal8Bit("Дата

```

```

        оформления заказа"));
orderModel->setHeaderData(3, Qt::Horizontal,
        QString::fromLocal8Bit("Оплачен"));

// Получение данных.
orderModel->setFilter("id_customer = 1");
orderModel->select();

// Связь QTableView с моделью
ui->tableView_2->setModel(orderModel);
// Не показывать поле "id_customer"
ui->tableView_2->setColumnHidden(4, true);
// Параметры выбора элементов в QTableView
ui-> tableView_2->setSelectionMode(QAbstractItemView::SingleSelection);
ui-> tableView_2->setSelectionBehavior(QAbstractItemView::SelectRows);
// Установка такой ширины столбцов, которой будет достаточно для
// размещения в них текста без необходимости вывода многоточия
ui-> tableView_2->resizeColumnsToContents();
// Связывание модели выборки. Класс QItemSelectionModel используется
// для отслеживания выборок в представлениях. Связанный с моделью
// выборки представления таблицы слот currentCustChanged() будет
// вызываться при всяком перемещении пользователя от одной записи к
// другой в таблице заказчиков
QObject::connect(ui->tableView->selectionModel(),
        SIGNAL(currentRowChanged(const QModelIndex, const
        QModelIndex)), this, SLOT(currentCustChanged(const
        QModelIndex)));
}

```

Слот `currentCustChanged()` вызывается при каждой смене текущего заказчика. Это происходит при переходе пользователя к другой записи таблицы заказчиков «Customers» (щелкая мышкой по соответствующей строке или используя клавиши Up и Down). Ниже приводится код слота `currentCustChanged()`.

```

void MainWindow::currentCustChanged(const QModelIndex &index)
{
    if (index.isValid())
    {
        // Получить текущую запись модели
        QSqlRecord rec = customModel->record(index.row());
        // Получить идентификатор заказчика id_customer
        QString id = rec.value("id_customer").toString();
        // Установить фильтр и выбрать соответствующие записи заказов
        orderModel->setFilter(QString("id_customer = '%1'").arg(id));
    }
    else
    {
        // Если заказчик недействителен (например, если заказчиков
        // вообще нет), то идентификатор id_customer таблицы
        // заказов Orders устанавливается в значение -1 (недействительный
        // идентификатор, которому не соответствует никакая запись).
        orderModel->setFilter("id_customer = '\\'");
    }
    orderModel->select();
}

```

Оборудование и материалы:

Для изучения материалов этой лабораторной работы необходимо:

1. Компьютер под управлением операционной системы Windows 8/10 с параметрами по умолчанию.

2. Установленные на компьютер программы Microsoft Visio, PostgreSQL, MySQL, VMWare/VirtualBox.

3. Текстовый процессор Microsoft Word 2010/2013/2016.

Указания по технике безопасности:

К выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж по технике безопасности.

Содержание отчета:

Отчет о выполнении лабораторной работы должен содержать:

- название и тему лабораторной работы;
- цель лабораторной работы;
- краткие теоретические сведения;
- ход выполнения работы;
- выводы.

Раздел «Ход выполнения работы» должен содержать инфологическую модель БД на языке таблица-связь, листинг программы и экраны форм, демонстрирующие работу приложения согласно пунктам порядка выполнения работы.

Контрольные вопросы:

- 1 В чем суть парадигмы «модель-контроллер-вид»? Особенности реализации этой парадигмы в системе Interview Framework SDK Qt
- 2 Какова архитектура клиентского приложения БД, использующего Interview Framework?
- 3 На какие уровни делятся классы модуля QSql?
- 4 Что такое ODBC? Как создать подключение источника данных с помощью ODBC?
- 5 Как выполняется настройка соединения с БД?
- 6 Для чего предназначен класс QSqlQuery?
- 7 Как выполняется непараметрический SQL-запрос и обрабатывается его результат?
- 8 Как передать параметры в параметрический SQL-запрос?
- 9 Что такое транзакция? Как ее реализовать средствами Qt?
- 10 Как средствами Qt реализовать просмотр данных в режиме «главный-детальный»?

Список литературы, рекомендуемый к использованию по данной теме:

1. Братченко, Н. Ю. (Северо-Кавказский федеральный университет). Базы данных. Основы работы с СУБД ORACLE 10G EXPRESS EDITION: учеб. пособие: Направление подготовки 230100 – Информатика и вычислительная техника / Н. Ю. Братченко; Сев.-Кав. федер. ун-т. – Ставрополь: СКФУ, 2013. – 202 с.: ил. – Гриф: Доп. УМО. – ISBN 978-5-88648-877-7
2. Разработка приложений на С с использованием СУБД PostgreSQL / И.А. Васюткина. – Новосибирск: НГТУ, 2015. – 143 с. – ISBN 978-5-7782-2699-9
3. Шацков, В.В. Программирование приложений баз данных с использованием СУБД MS SQL Server / В.В. Шацков: учебное пособие Электронный ресурс: Санкт-Петербургский государственный архитектурно-строительный университет, ЭБС АСВ; Санкт-Петербург, 2015. – 80 с. – Книга находится в базовой версии ЭБС IPRbooks. – ISBN 978-5-9227-0607-0
4. Голицина, О. Л. Системы управления базами данных: [учеб. пособие] / О.Л. Голицина, Т.Л. Партыка, И.И. Попов. – М.: ФОРУМ, 2011. – 432 с.: ил. – (Профессиональное образование). – На учебнике гриф: Рек.МО. – Библиогр.: с. 315. – ISBN 5-8199-0251-3
5. Гудов, А. М. Базы данных и системы управления базами данных. Программирование на языке PL/SQL / А.М. Гудов; С.Ю. Завозкин; Т.С. Рейн. – Кемерово: Кемеровский государственный университет, 2010. – 134 с. – ISBN 978-5-8353-1005-0
6. Харченко М. В. Организация баз данных. Методические указания к выполнению лабораторных работ для студентов направления 6.050102 – «Компьютерная инженерия». – Чернигов, ЧГТУ. 2013 – 141 с

Практическая работа №5. Проектирование реляционных баз данных на основе метода ER-диаграмм

Цель работы: изучить технологию автоматизированного проектирования реляционной базы данных с помощью визуального редактора Azzurri Clay.

Теоретическая часть

Под проектированием базы данных (БД) следует понимать процесс создания схемы базы данных и определения необходимых ограничений целостности. Весь процесс проектирования можно разделить на 3 этапа:

- концептуальное (инфологическое) проектирование;
- логическое проектирование;
- физическое проектирование.

Концептуальное проектирование – это построение семантической модели предметной области (ПрО), то есть информационной модели наиболее высокого уровня абстракции. Такая модель отображает реальный мир в некоторые понятные человеку концепции, полностью независимые от параметров среды хранения данных, т.е. без ориентации на какую-либо конкретную СУБД и модель данных. Термины «семантическая модель», «концептуальная модель» и «инфологическая модель» являются синонимами.

Чаще всего концептуальная модель БД включает в себя:

- описание информационных объектов, или понятий ПрО и связей между ними;
- описание ограничений целостности, т.е. требований к допустимым значениям данных и к связям между ними.

На сегодняшний день существует множество подходов к построению таких моделей: графовые модели, семантические сети, модель "сущность-связь" и т.д. Наиболее популярной из них является модель "сущность-связь".

Логическое (дatalogическое) проектирование – создание схемы БД на основе конкретной модели данных, например, иерархической, реляционной, объектной и т.д. Для реляционной модели данных логическая модель представляет собой набор отношений обычно с указанием первичных

ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

Преобразование концептуальной модели в логическую модель, как правило, осуществляется по формальным правилам. На этапе логического проектирования учитывается специфика конкретной модели данных, но может не учитываться специфика конкретной СУБД.

Физическое проектирование – создание схемы БД для конкретной СУБД. Специфика конкретной СУБД может включать в себя ограничения на именование объектов БД, ограничения на поддерживаемые типы данных и т.п. Кроме того, специфика конкретной СУБД при физическом проектировании включает выбор решений, связанных с физической средой хранения данных (выбор методов управления дисковой памятью, разделение БД по файлам и устройствам, методов доступа к данным), создание индексов и т.д.

Концептуальная модель «сущность-связь»

Модель «сущность-связь» (англ. Entity-Relationship model), или ER-модель, предложенная П. Ченом в 1976 г., является наиболее известным представителем класса концептуальных (семантических, инфологических) моделей предметной области. ER-модель обычно представляется в графической форме, с использованием оригинальной нотации П. Чена, называемой ER-диаграмма.

Основные преимущества ER-моделей:

- наглядность;
- модели позволяют проектировать базы данных с большим количеством объектов и атрибутов;
- ER-модели реализованы во многих системах автоматизированного проектирования БД.

Основными элементами ER-модели являются:

- объекты (сущности);
- атрибуты объектов;

- связи между объектами.

Сущность – это реальный или представляемый объект, информация о котором должна сохраняться и быть доступной. В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности – это имя типа, а не некоторого конкретного экземпляра этого типа. Для большей выразительности и лучшего понимания имя сущности может сопровождаться примерами конкретных экземпляров этого типа (рисунок 10.1).



Рисунок 10.1 – Пример типа сущности

На рисунке 10.1 изображена сущность ГОРОД с примерными экземплярами «Киев» и «Харьков». Эта примитивная диаграмма несет важную информацию. Она показывает, что в БД будут содержаться однотипные структуры данных (экземпляры сущности), описывающие города.

При определении типа сущности необходимо гарантировать, что каждый экземпляр сущности может быть отличим от любого другого экземпляра той же сущности. Это требование в некотором роде аналогично требованию отсутствия кортежей-дубликатов в реляционных таблицах.

Связь – это графически изображаемая ассоциация, устанавливаемая между двумя типами сущностей. Как и сущность, связь – это типовое понятие, все экземпляры обоих связываемых типов сущностей подчиняются устанавливаемым правилам связывания. Поэтому правильнее говорить о типе связи, устанавливаемой между типами сущности, и об экземплярах типа связи, устанавливаемых между экземплярами типа сущности. В рассматриваемом варианте ER-модели эта ассоциация всегда является бинарной и может существовать между двумя разными типами сущностей или между типом сущности и им же самим (рекурсивная связь). В любой

связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указываются имя конца связи, степень конца связи (сколько экземпляров данного типа сущности должно присутствовать в каждом экземпляре данного типа связи), обязательность связи (т. е. любой ли экземпляр данного типа сущности должен участвовать в некотором экземпляре данного типа связи).

Связь представляется в виде ненаправленной линии, соединяющей две сущности или ведущей от сущности к ней же самой. При этом в месте «стыковки» связи с сущностью используются:

- трехточечный вход в прямоугольник сущности, если для этой сущности в связи могут (или должны) использоваться много (many) экземпляров сущности;
- одноточечный вход, если в связи может (или должен) участвовать только один (one) экземпляр сущности.

Связь между сущностями ГРУППА и СТУДЕНТ, показанная на рисунке 6.2, связывает студентов и группы. Конец связи с именем «учится в» позволяет связывать с одной группой более одного студента, причем каждый студент должен быть связан с какой-либо группой. Конец связи с именем «состоит из» показывает, что каждый студент может принадлежать только одной группе, причем группа должна состоять из студентов.

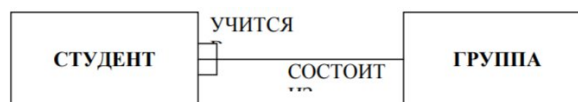


Рисунок 10.2 – Пример типа связи

Лаконичная устная трактовка изображенной диаграммы состоит в следующем:

- каждый СТУДЕНТ учится только в одной ГРУППЕ;
- каждая ГРУППА состоит из одного или более СТУДЕНТОВ.

Атрибутом сущности является любая деталь, которая служит для

уточнения, идентификации, классификации, числовой характеристики

или выражения состояния сущности. Имена атрибутов заносятся в прямоугольник, изображающий сущность, под именем сущности и изображаются малыми буквами, возможно, с примерами.

Пример типа сущности СТУДЕНТ с указанными атрибутами показан на рисунке 10.3. С технической точки зрения атрибуты типа сущности в ER-модели похожи на атрибуты отношения в реляционной модели данных. И в том, и в другом случаях введение именованных атрибутов вводит некоторую типовую структуру данных, имя которой совпадает с именем типа сущности в случае ER-модели или с именем переменной отношения в случае реляционной модели. Этой типовой структуре должны следовать все экземпляры типа сущности или все кортежи отношения.

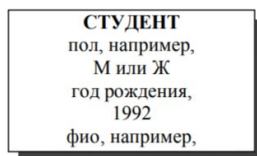


Рисунок 10.3 – Пример типа сущности с атрибутами

При определении атрибутов типа сущности в ER-модели указание домена атрибута не является обязательным, хотя это и возможно.

При определении атрибута отношения допускается использование имен атрибутов, совпадающих с именами своих доменов (это два разных пространства имен, и наличие одинаковых имен у атрибутов и доменов не вызывает коллизий). Поэтому при определении атрибутов типов сущности можно так подбирать их имена, что они в дальнейшем будут подсказывать, какие домены у этих атрибутов имеются в виду. Пониманию предполагаемой сути доменов способствует и возможность указания примеров значений атрибутов. Например, на рисунке 6.3 имеется атрибут год рождения, в качестве примерного значения которого указано «1992». Это подсказывает, что в реляционной схеме при определении соответствующего атрибута наиболее естественным базовым типом данных будет темпоральный тип ДАТА, значения которого задают дату с точностью до года.

В настоящее время существует множество программ, с помощью которых можно создавать ER-модели. К ним относится и программное решение Clay Mark II.

Использование Clay Mark II для проектирования баз данных

Clay Mark II представляет собой утилиту для работы с БД, которая построена на платформе Eclipse и содержит визуальный редактор моделей БД. По построенной модели можно сгенерировать SQL (DDL) код БД. Clay Mark II содержит редакторы для создания логического и физического описания модели данных и поддерживает наиболее распространенные реляционные СУБД.

Данная утилита не привязана к технологии какой-либо конкретной фирмы, поставляющей СУБД или средства разработки. Она поддерживает различные серверы БД, а также может обращаться к БД через интерфейс

JDBC. К поддерживаемым Clay Mark II СУБД относятся: MySQL, PostgreSQL, Firebird, DB 2, Informix.

Одним из преимуществ Clay Mark II является возможность использования совместно с IDE Eclipse для разработки клиентской части приложения.

Структура процесса моделирования в Clay Mark II

В Clay Mark II используются два уровня представления модели БД: логический и физический (что соответствует концептуальному и логическому уровню представления данных в архитектуре СУБД). На логическом уровне не рассматривается использование конкретной СУБД, не определяются типы данных (например, целое или вещественное число) и не определяются индексы для таблиц. Целевая СУБД, имена объектов и типы данных, составляют второй (физический) уровень модели.

Clay Mark II предоставляет возможности создавать и управлять этими двумя различными уровнями представления одной модели.

Процесс построения информационной модели БД состоит из трех этапов.

1. Создание логической модели БД:
 - определение сущностей;
 - определение зависимостей между сущностями;
 - задание первичных и внешних ключей;
 - определение неключевых атрибутов сущностей.
2. Переход к физическому описанию модели БД:
 - назначение соответствий имя сущности – имя таблицы, атрибут сущности - атрибут таблицы;
 - определение типов данных для атрибутов таблицы.
3. Генерация схемы БД.

Создание логической и физической модели БД, генерация схемы БД

С точки зрения пользователя Clay Mark II, процесс создания логической модели БД заключается в визуальном редактировании ER-диаграммы. Диаграмма строится из трех основных блоков: сущностей, атрибутов и связей.

Сущности и атрибуты

На диаграмме сущность изображается прямоугольником. В зависимости от режима представления диаграммы прямоугольник может содержать имя сущности, ее описание, список ее атрибутов и другие сведения. Основная информация, описывающая сущность, включает:

- атрибуты, составляющие первичный ключ;
- неключевые атрибуты.

Первичный ключ (Primary Key) – это атрибут или набор атрибутов, уникально идентифицирующий экземпляр сущности. Если несколько наборов атрибутов могут уникально идентифицировать сущность, то выбор одного из них осуществляется разработчиком на основании анализа предметной области (ПрО) и учета следующих требований к первичному ключу:

- первичный ключ не должен принимать пустые (NULL) значения;
- первичный ключ не должен изменяться в течение времени;

- размер первичного ключа должен быть минимальным.

При этом если разработчик считает, что какой-либо из оставшихся атрибутов или набор атрибутов, не идентифицируют уникально экземпляры сущности, но часто используются для доступа к данным, то он может определить его в качестве индекса (Index).

Рассмотрим вышесказанное на примере сущности СОТРУДНИК:

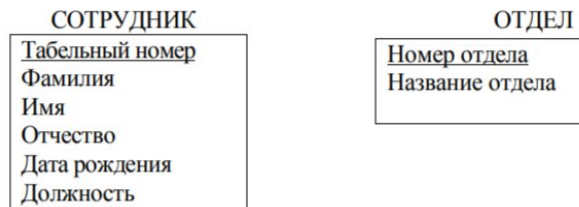


Рисунок 10.4 – Пример описания сущности

Среди всех атрибутов данной сущности на роль первичного ключа могут претендовать <табельный номер> и группа атрибутов <фамилия>, <имя>, <отчество>, <дата рождения> (последний необходим, т.к. на предприятии могут работать полные тезки). Очевидно, что по соображениям размера в качестве первичного ключа следует выбрать <табельный номер>.

На диаграмме атрибуты, составляющие первичный ключ, могут располагаться в любой части прямоугольника (рекомендуется указывать их первыми), и отделяются от прочих (не входящих в первичный ключ) линией подчеркивания.

Связь в Clay Mark II трактуется как функциональная зависимость между двумя сущностями (в частности, возможна связь сущности с самой собой – рекурсивная связь).

Если рассматривать диаграмму как графическое представление правил ПрО, то сущности именуются существительными, а связи – глаголами. Например, между сущностями ОТДЕЛ и СОТРУДНИК существует связь <состоит из> (ОТДЕЛ состоит из СОТРУДНИКОВ). На рисунке 10.5 приведен пример связи.

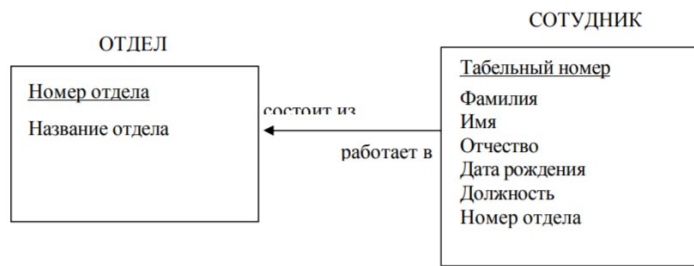


Рисунок 10.5 – Пример связи

В Clay Mark II связи определяются четырьмя основными элементами:

- родительская и дочерняя (зависимая) сущности;
- степень связи;
- роли связи;
- требования по обеспечению ссылочной целостности.

При определении связи миграция атрибутов первичного ключа родительской сущности в соответствующую область атрибутов дочерней сущности не происходит. Поэтому такие атрибуты следует вводить вручную.

Степень связи (Multiplicity) представляет собой отношение количества экземпляров родительской сущности к соответствующему количеству экземпляров дочерней сущности. В Clay Mark II предоставляется 4 предопределенных варианта степени связи и один пользовательский (когда пользователь сам определяет требуемое количество экземпляров сущностей, участвующих в связи):

- 0 или 1 (0..1);
- 0 или много (1 .. *);
- 1 или 1 (1 .. 1);
- 1 или много (1 .. *).

Роли связи (Source Entity Role). Для именования связи можно использовать название роли каждой сущности, участвующего в этой связи.

Контроль ссылочной целостности. Под ссылочной целостностью в Clay Mark II понимается обеспечение требования, чтобы значения внешнего ключа экземпляра дочерней сущности соответствовали значениям

первичного ключа в родительской сущности, В целях контроля ссылочной целостности для каждой связи могут быть заданы требования по обработке операций INSERT/UPDATE/DELETE для родительской и дочерней сущности. Clay Mark II предоставляет следующие варианты обработки этих событий:

- отсутствие проверки (NO ACTION);
- запрет операции (RESTRICT);
- каскадное выполнение операции (CASCADE);
- установка NULL-значения (SET NULL);
- установка заданного значения по умолчанию (SET DEFAULT).

Перед началом моделирования разработчику необходимо выбрать конкретную СУБД. Для переключения между уровнями отображения модели используются соответствующие кнопки меню Clay Mark II.

На уровне физической модели сущности соответствует таблица в реальной СУБД, атрибуту сущности – столбец таблицы, связи – внешний ключ, первичным ключам – уникальные значения. Clay Mark II не производит автоматическое присваивание имен элементов логической модели элементам физической модели, поэтому разработчику необходимо проделывать это вручную. Для идентификаторов объектов логической модели можно использовать национальный или латинский алфавит. Идентификаторы объектов физической модели (имена таблиц, столбцов и индексов) придется ввести на латинице.

Для каждого столбца разработчик должен указать тип данных, возможность пустых значений, значения по умолчанию и т.п. в зависимости от используемой СУБД. Например, логической модели, представленной на рисунке 10.5, будет соответствовать физическая модель, представленная на рисунке 10.6.

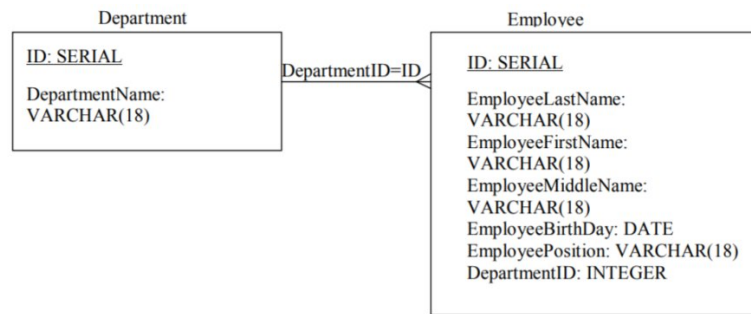


Рисунок 10.6 – Физическая модель БД

Для реализации связи «многие-ко-многим» необходимо создать отдельную сущность, которая будет представлена на логическом и физическом уровне модели. В Clay Mark II выполнить это автоматически нельзя.

Схема базы данных генерируется на основе физической модели в виде SQL-скрипта (Script File – файла с расширением sql). Например, для физической модели, изображенной на рисунке 10.6, под PostgreSQL будет сгенерирована следующая схема:

```

DROP INDEX IX_empl_1;

DROP TABLE Employee;
DROP TABLE Department;

CREATE TABLE Department (
    id SERIAL NOT NULL
    , DepartmentName VARCHAR(18) NOT NULL
    , PRIMARY KEY (id)
);
CREATE UNIQUE INDEX IX_empl_1 ON Department (DepartmentName);

CREATE TABLE Employee (
    id SERIAL NOT NULL
    , employeeLastName VARCHAR(18) NOT NULL
    , employeeFirstName VARCHAR(18) NOT NULL
    , employeeMiddleName VARCHAR(18) NOT NULL
    , employeeBirthDay DATE NOT NULL
    , employeePosition VARCHAR(18) NOT NULL
    , departmentID INTEGER NOT NULL
    , PRIMARY KEY (id)
    , CONSTRAINT dpFK FOREIGN KEY (departmentID)
        REFERENCES Department (id) ON DELETE RESTRICT
ON UPDATE CASCADE
);
  
```

В соответствии с данным SQL-скриптом в схему БД будут включены: две таблицы, уникальный индекс для атрибута «Название отдела», два первичных ключа, один внешний ключ и операторы удаления индекса и таблиц, если они ранее были созданы.

Прямое и реверсное проектирование

Прямое проектирование (Forward Engineer) – преобразование модели базы данных в схему БД.

Реверсное проектирование (Reverse Engineer) – создание модели базы данных на основе имеющейся схемы БД.

При реверсном проектировании модель БД восстанавливается, исходя из информации о системном каталоге СУБД, и может быть использована для создания схемы БД, ориентированной на другую СУБД. Таким образом, Clay Mark II не только обеспечивает генерирование схем для разных СУБД по разработанной модели, но и поддерживает конвертирование одной схемы в другую.

Clay Mark II позволяет также производить выборочную вставку объектов, получаемых в процессе реверсного проектирования, в уже существующую модель.

Пример разработки модели БД с помощью Clay Mark II

В данном разделе рассматриваются два способа проектирования БД: прямое и реверсное.

Прямое проектирование

Постановка задачи

В некотором учреждении создается локальная информационная система (ИС), предназначенная для автоматизации процесса учета сделок купли-продажи акций. Создаваемая система должна обеспечить ввод, хранение и поиск информации о совершенных сделках. Каждой сделке присваивается уникальный цифровой код. Информация о сделке должна содержать следующие сведения:

- дата и время сделки,

- количество покупаемых и/или продаваемых акций и их номинальная стоимость,
- фамилии, имя, отчество клиента,
- номер паспорта клиента
- личный идентификационный код, если клиент – физическое лицо, или регистрационный номер доверенности, если клиент представляет юридическое лицо,
- фамилия, инициалы и учетный номер личного дела кассира в отделе кадров.

Система должна позволять подводить итоги за произвольное количество дней, а также осуществлять поиск информации о сделках по личному идентификационному коду клиента или номеру доверенности. Задача состоит в проектировании структуры базы данных разрабатываемой ИС.

Создание логической модели БД

Проведем анализ ПрО с целью выделить основные сущности. Поскольку речь идет об учете сделок купли-продажи акций, ясно, что в модели должна присутствовать сущность СДЕЛКА (Transaction). Понятие сделка подразумевает участие по крайней мере двух совершающих ее сторон, а также наличие предмета сделки. Так как участниками сделки являются клиент и кассир, а предметом сделки являются акции, при чем клиенты могут быть двух категорий – ФИЗИЧЕСКОЕ ЛИЦО (Natural person) и ЮРИДИЧЕСКОЕ ЛИЦО (Legal entity), необходимо ввести еще сущности: АКЦИЯ (Share), ФИЗИЧЕСКОЕ ЛИЦО (Natural person), ЮРИДИЧЕСКОЕ ЛИЦО (Legal entity) и КАССИР (Teller). Таким образом, получаем модель, состоящую из пяти сущностей – АКЦИЯ, КАССИР, СДЕЛКА, ФИЗИЧЕСКОЕ ЛИЦО и ЮРИДИЧЕСКОЕ ЛИЦО.

Теперь для каждой сущности необходимо указать первичный ключ и неключевые атрибуты. Источником информации в этом случае должен стать перечень требований, приведенных в задании. Рассмотрим каждую из

сущностей.

Атрибутами сущностей ФИЗИЧЕСКОЕ ЛИЦО и ЮРИДИЧЕСКОЕ ЛИЦО будут <фамилия>, <имя>, <отчество> и <номер паспорта>. Кроме общих атрибутов сущность ФИЗИЧЕСКОЕ ЛИЦО содержит атрибут <идентификационный код>, а сущность ЮРИДИЧЕСКОЕ ЛИЦО – <номер доверенности>. Эти атрибуты следует определить как ключевые, т.к. они однозначно будут идентифицировать сущности.

Из атрибутов сущности КАССИР – <учетный номер> и <фамилия и инициалы> – первичным ключом будет <учетный номер>.

Для сущности АКЦИЯ формально определим три атрибута: <код акции>, <наименование акции> и <номинал>, из которых первый будет первичным ключом.

Что касается сущности СДЕЛКА, то часть атрибутов у нее будет внешними ключами: <код продаваемой акции>, <код покупаемой акции>, <учетный номер кассира>, <идентификационный номер физического лица>, <номер доверенности юридического лица>, а часть будет добавлена: <уникальный цифровой код сделки> в качестве первичного ключа, <количество купленных акций>, <количество проданных акций> и <дата и время сделки>.

Для создания новой модели в Clay Mark II необходимо в окне редактора модели в режиме логического представления разместить все необходимые сущности. Переключения между логическим и физическим представлением осуществляется с помощью кнопок (рисунок 10.7).

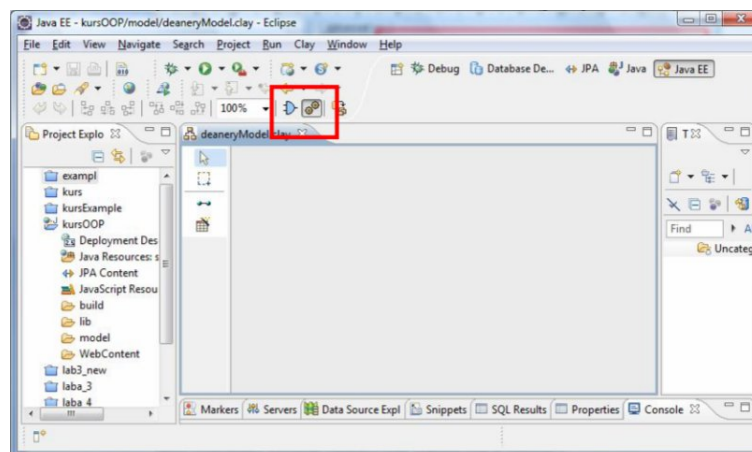


Рисунок 10.7 – Редактор модели БД

Для добавления сущности в модель используется кнопка Add Table (рисунок 10.8).

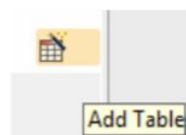


Рисунок 10.8 – Кнопка добавления сущности (таблицы)

Нажав кнопку Add Table и кликнув в любом свободном месте редактора модели, будет размещен прямоугольник с именем по умолчанию TABLE_1. Если дважды щелкнуть на нем левой кнопкой мыши, будет открыто окно редактирования параметров сущности (таблицы) (рисунок 10.9).

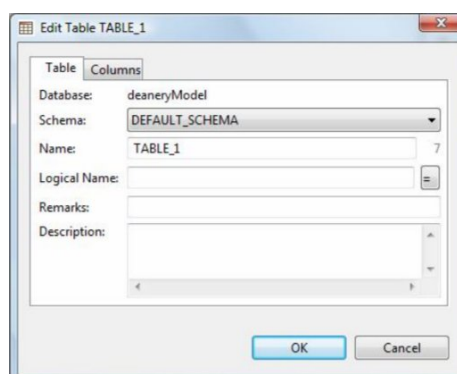


Рисунок 10.9 – Редактор параметров сущности

На этапе логического проектирования можно редактировать только

параметры: логическое имя, первичный ключ. Эта информация будет отображаться в режиме просмотра логической модели БД (рисунок 10.10).

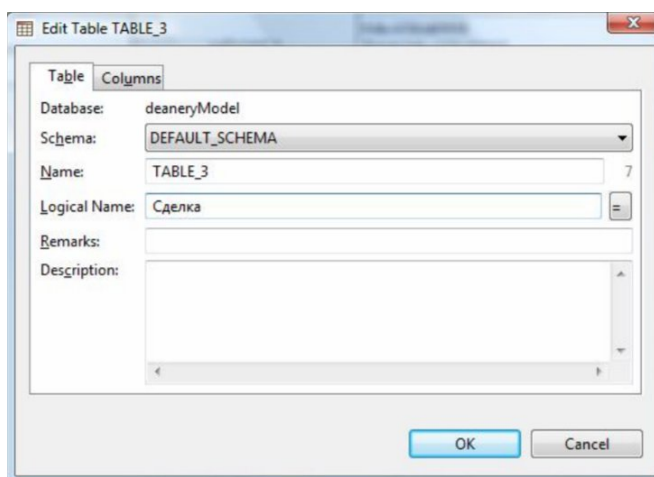


Рисунок 10.10 – Редактирование параметров сущности

Для редактирования атрибутов сущности предназначена вкладка Columns редактора сущностей (рисунок 10.11).

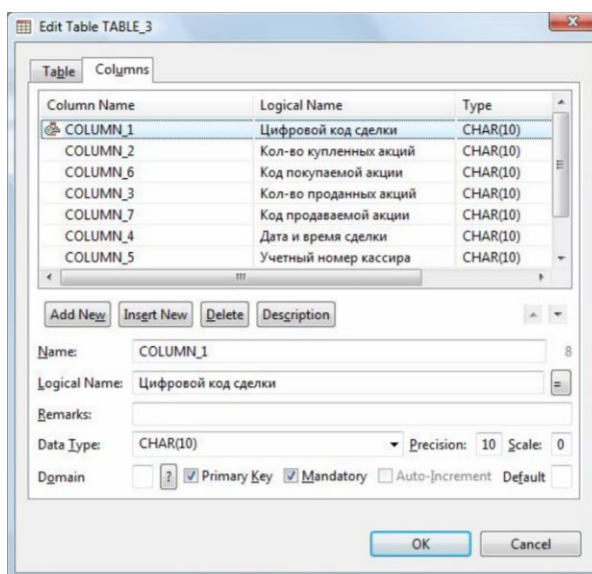


Рисунок 10.11 – Редактор атрибутов сущности

Как видно из рисунка, пользователю доступны следующие действия:

- добавить новый атрибут (колонку) в конец списка – Add New, или перед текущим атрибутом – Insert New);
- удалить атрибут;
- добавить описание;

- ввести имя;
- задать логическое имя;
- указать тип данных (Data Type);
- определить значение по умолчанию (Default);
- указать обязательность ввода значений (Mandatory);
- определить атрибут как первичный ключ (Primary Key).

На диаграмме атрибуты, составляющие первичный ключ, принято располагать в верхней части прямоугольника сущности. Они отделяются от прочих атрибутов линией подчеркивания.

После добавления таблиц логическая модель примет вид, как показано на рисунке 10.12.

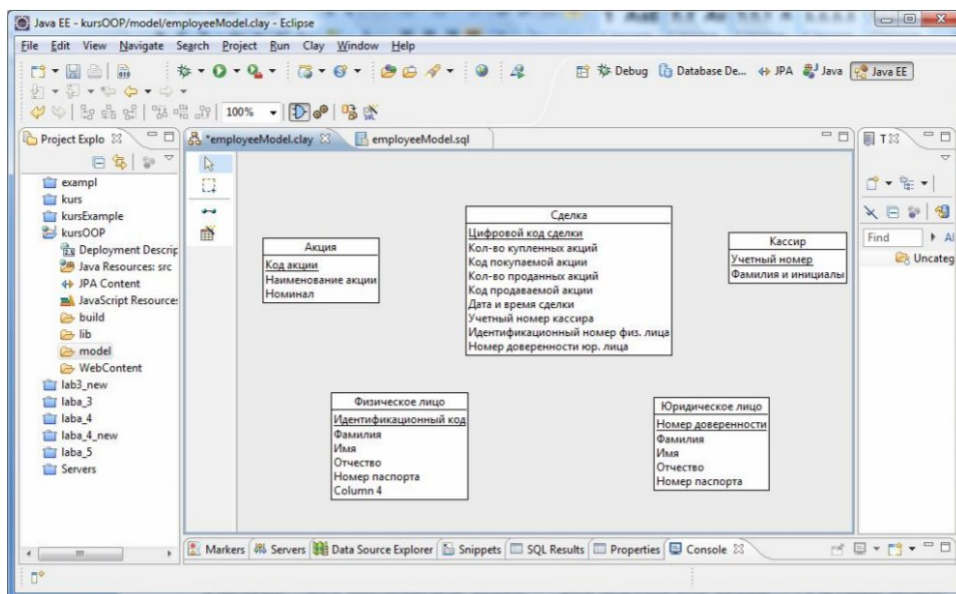


Рисунок 10.12 – Логическая модель

Следующим шагом в процессе создания логической модели должно стать определение связей между сущностями. Для задания связи между двумя сущностями необходимо указать направление связи (родительская и дочерняя сущности), мощность связи, допустимость пустых (null) значений, требования по обеспечению ссылочной целостности, а в некоторых случаях и роли.

Для определения параметров связей между указанными сущностями

сначала составим описание ПрО при помощи ряда истинных высказываний на естественном языке:

- ~ любой КЛИЕНТ должен совершить одну или несколько СДЕЛОК;
- ~ каждую СДЕЛКУ должен совершать только один КЛИЕНТ;
- ~ любой КАССИР может обслуживать нуль, одну или несколько СДЕЛОК;
- ~ каждую СДЕЛКУ должен обслуживать только один КАССИР;
- ~ любая АКЦИЯ может покупаться и продаваться при разных СДЕЛКАХ;
- ~ при совершении СДЕЛКИ АКЦИИ могут только покупаться, только продаваться или покупается один тип АКЦИЙ и продается другой тип АКЦИЙ или наоборот.

Анализ приведенных высказываний позволяет выделить четыре связи.

1. КЛИЕНТ совершает (make a bargain) СДЕЛКА.
2. КАССИР обслуживает (serve) СДЕЛКА
3. АКЦИЯ покупается при (is purchased) СДЕЛКА.
4. АКЦИЯ продается при (is soled) СДЕЛКА.

Все четыре связи являются связями "один-ко-многим". Во всех четырех случаях сущность СДЕЛКА является дочерней. При определении этих связей необходимо учесть, что любая сделка становится фактом только в том случае, если в наличии имеются действующие лица и предмет сделки. Отсюда следует, что во всех четырех связях внешние ключи, ссылающиеся на родительские сущности, не могут принимать пустые значения. В то же время могут быть экземпляры родительских сущностей, на которые не ссылается ни один из экземпляров сущности СДЕЛКА, например, могут быть акции, которые не использовались ни в одной из сделок. Таким образом, все связи, кроме первой, могут иметь мощность 0, 1 или более. Первая связь не может иметь мощность 0, т.к. в данном случае любой человек становится КЛИЕНТОМ только тогда, когда он совершает хотя бы одну сделку.

Для установления связей между сущностями используется кнопка

Foreign Key Reference (рисунок 10.13).



Рисунок 10.13 – Кнопка добавления связи между сущностями (таблицами)

Для установления связи между сущностями надо последовательно щелкнуть левой клавишей мыши по кнопке соответствующего типа на панели инструментов (рисунок 10.13), по прямоугольнику дочерней сущности и затем – по прямоугольнику родительской сущности.

Параметры связей задаются при помощи редактора связей (рисунок 10.14). Вызвать этот редактор можно двойным щелчком левой клавиши мыши по линии связи.

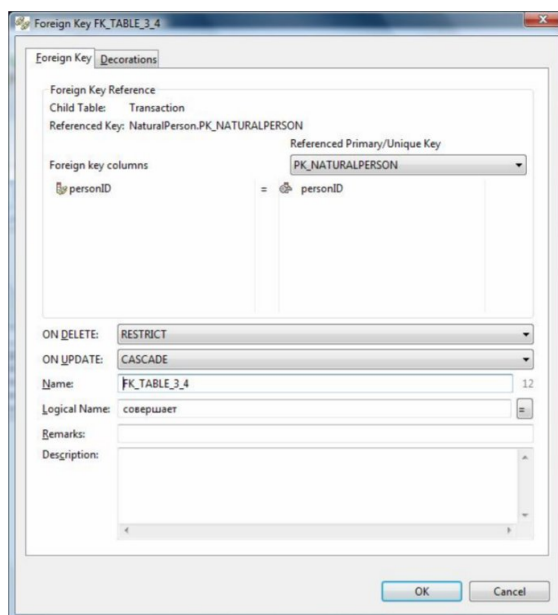


Рисунок 10.14 – Редактор связи

В открывшемся окне необходимо выбрать поле, являющееся внешним ключом у дочерней сущности, определить ограничение ссылочной целостности, задать имя связи и логическое имя.

На вкладке Decorations задаются степени концов связи и, если нужно, определяются роли сущностей, участвующих в связи (рисунок 10.15).

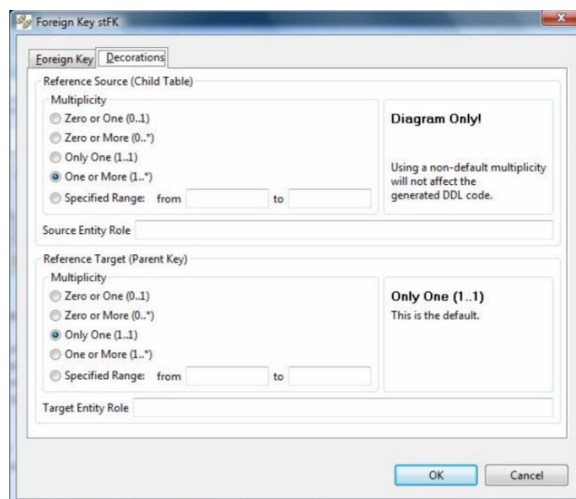


Рисунок 10.15 – Редактирование степени связи и ролей сущностей

В результате выполнения описанных действий ER-модель примет вид, как показано на рисунке 10.16.

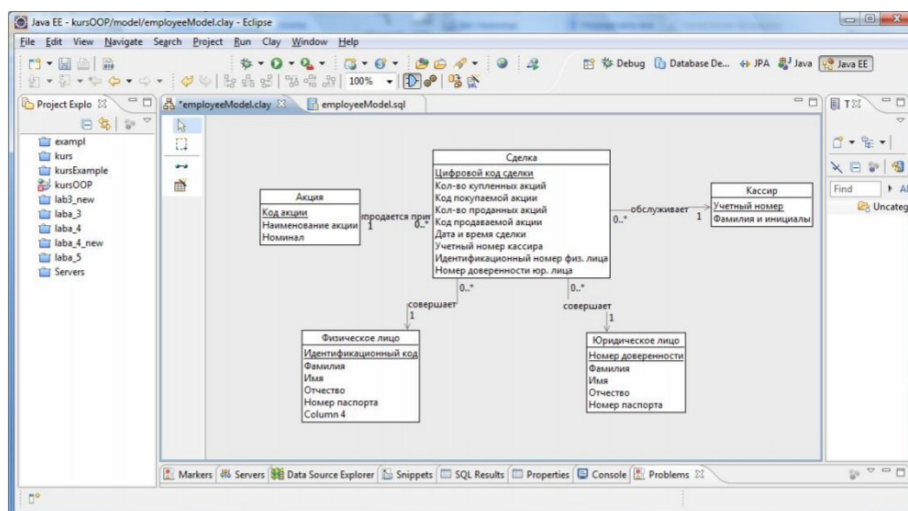


Рисунок 10.16 – Логическая модель

На этом процесс создания логической модели БД завершается

Создание физической модели БД

На уровне физической модели сущности соответствует таблица в реальной СУБД, атрибуту – столбец (поле) таблицы, связи – внешний ключ, первичным ключам - уникальные индексы, а атрибутам, часто используемым для доступа к данным - неуникальные.

Описание таблиц моделируемой БД, с указанием типов полей и

ограничений целостности приводится в таблице 10.1.

Таблица 10.1 – Описание таблиц БД

Название таблицы	Поле	Описание	Тип данных	Ограничение
NaturalPerson	CustomerLastName	фамилия клиента	строка, символов 20	обязательное для ввода
	CustomerFirstName	имя клиента	строка, символов 20	обязательное для ввода
	CustomerMiddleName	отчество клиента	строка, символов 20	обязательное для ввода
	CustomerPassportNum	серия и номер паспорта клиента	строка, символов 10	обязательное для ввода; уникальные значения
	PersonID	идентификационный код клиента	целое число	первичный ключ
Legaffintity	CustomerLastName	фамилия клиента	строка, символов 20	обязательное для ввода
	CustomerFirstName	имя клиента	строка, символов 20	обязательное для ввода
	CustomerMiddleName	отчество клиента	строка, символов 20	обязательное для ввода
	CustomerPassportNum	серия и номер паспорта клиента	строка, символов 10	обязательное для ввода; уникальные значения
	ProxyNum	номер доверенности	целое число	первичный ключ
Teller	TellerID	учетный номер кассира	Целое число	первичный ключ
	TellerNameInitials	фамилия и инициалы кассира	строка, символов 25	обязательное для ввода
Share	ShareCode	код акции	целое число	первичный ключ
	ShareName	наименование акции	строка, символов 30	обязательное для ввода
	ShareFvalue	номинал акции	вещественное число	обязательное для ввода; значение по умолчанию «0».
Transaction	TransID	уникальный цифровой код сделки	целое число	первичный ключ

	PurchaseQuantity	количество купленных акций	целое число	обязательное для ввода; допустимые значения «>0»; значение по умолчанию «1».
	SaleQuantity	количество проданных акций	целое число	обязательное для ввода; допустимые значения «>0»; значение по умолчанию «1».
	TransDateTime	дата и время сделки	дата	обязательное для ввода; значение по умолчанию «текущая дата».
	PersonID	идентификационный номер физ. лица	целое число	внешний ключ, каскадное обновление, запрет удаления; необязательное для ввода.
	ProxyNum	номер доверенности юр. лица	целое число	внешний ключ, каскадное обновление, запрет удаления; необязательное для ввода.
	TellerID	учетный номер кассира	целое число	внешний ключ, каскадное обновление, запрет удаления; обязательное для ввода.
	ShareCodeSold	код проданных акций	целое число	внешний ключ, каскадное обновление, запрет удаления; необязательное для ввода.

	ShareCodePurchased	код купленных акций	целое число	внешний ключ, каскадное обновление, запрет удаления; необязательное для ввода.
--	--------------------	---------------------	-------------	--

Переключившись в режим отображения физической модели, изменяем типы данных столбцов во всех таблицах с помощью редактора таблицы, который вызывается двойным щелчком левой клавиши мыши по прямоугольнику таблицы (рисунок 10.9). Все редакторы для физической модели вызываются из контекстного меню, которое появляется при щелчке правой клавишей мыши по прямоугольнику таблицы.

В окне редактора таблицы нужно ввести имя таблицы (рисунок 10.17) и задать ограничения и типы данных для полей таблиц (рисунок 10.18).

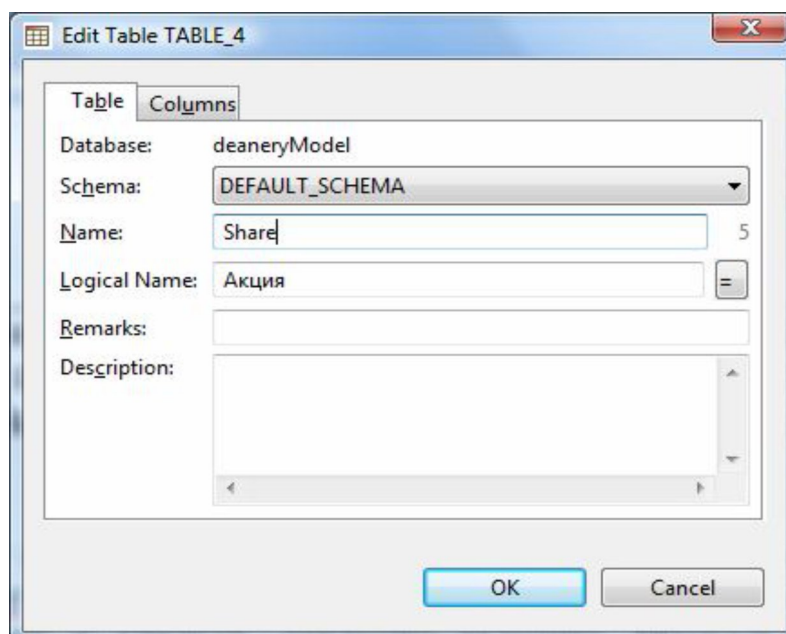


Рисунок 10.17 – Редактор параметров таблицы

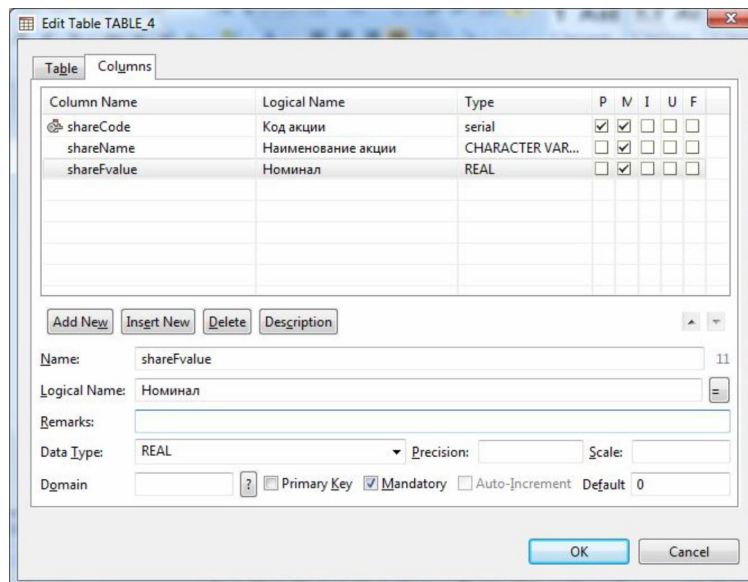


Рисунок 10.18 – Редактор колонок таблиц

Физическая модель БД после выполнения описанных действий примет вид, как показано на рисунке 10.19.

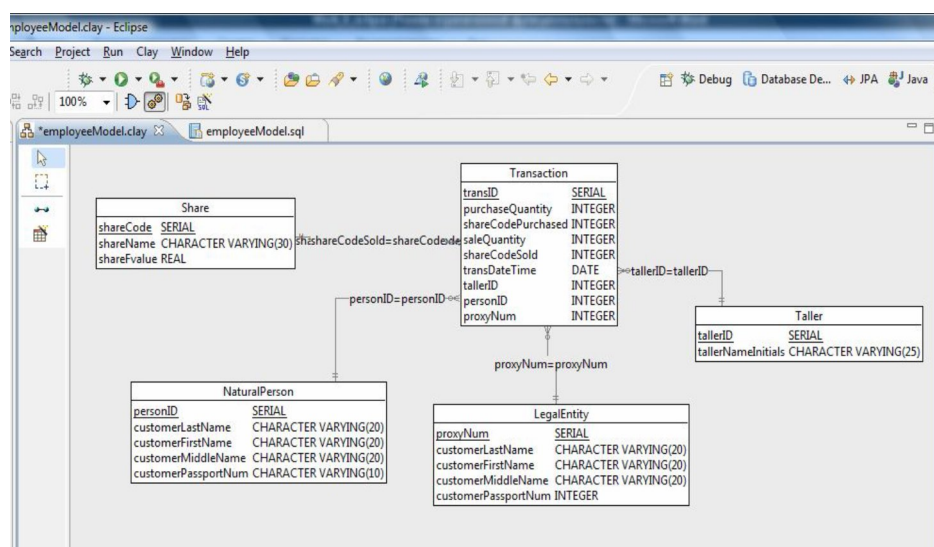


Рисунок 10.19 – Физическая модель

Для атрибутов <имя>, <фамилия> и <отчество> таблиц NaturalPerson и LegalEntity создадим индекс, чтобы обеспечить возможность быстрого поиска информации о сделках, согласно заданию. Поскольку в задании указано, что создаваемая система должна позволять подводить итоги оборота акций за произвольное количество дней, целесообразно сделать атрибут <дата и время сделки> таблицы Transaction индексом, т.к. он довольно часто

будет использоваться для доступа к данным.

Создать индекс можно с помощью редактора индексов (рисунок 6.20) выбрав пункт Edit Table Indexes контекстного меню таблицы.

В окне редактора нужно: ввести имя создаваемого индекса; определить уникальный индекс или нет; указать колонки таблицы, которые образуют индекс; задать режим сортировки для каждой колонки индекса (Ascending, Descending).

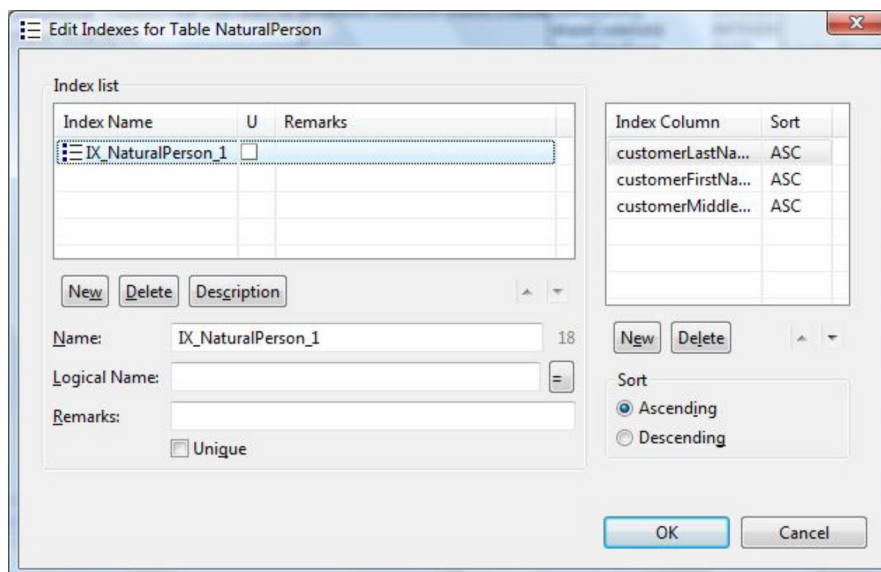


Рисунок 10.20 – Редактор индексов

Значения поля customerPassportNum (<номер паспорта>) таблиц NaturalPerson и LegalEntity должны быть уникальными. Для определения уникальных полей используется специальный редактор (рисунок 10.21), который вызывается путем выбора пункта Edit Table Unique Keys контекстного меню таблицы.

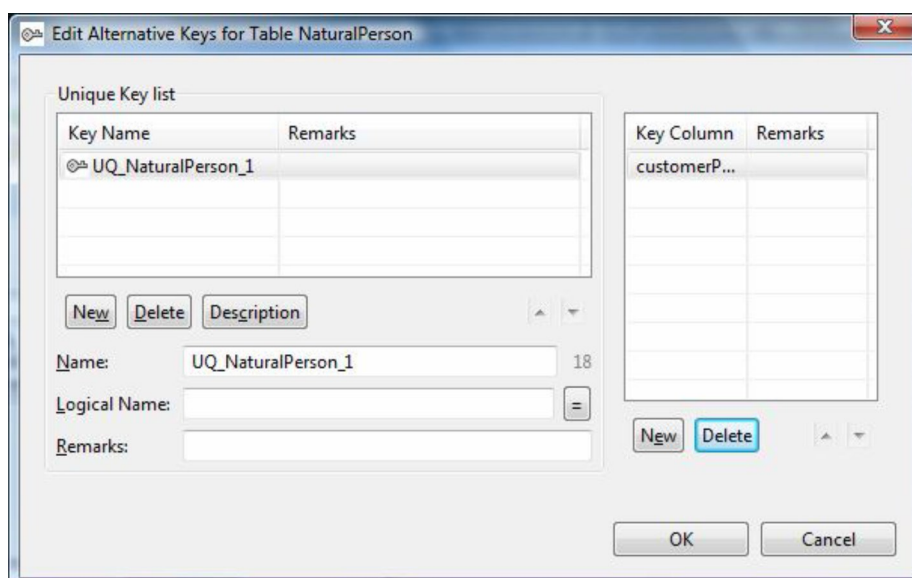


Рисунок 10.21 – Редактор уникальных полей таблицы

В окне редактора нужно ввести имя ограничения и указать колонки, значения в которых должны быть уникальными.

На этапе физического моделирования БД проектируются хранимые процедуры и триггеры, контролирующие ограничения целостности, накладываемые семантикой ПрО. Рассмотрим несколько примеров.

Пример 1. Триггер, который проверяет значения в полях PersonID и ProхуNum при добавлении или изменении данных в таблице Transaction. Если одно из них не пусто, ввод данных в другое недопустим (нельзя, чтобы одну сделку осуществляло 2 клиента). В противном случае добавление или изменение такой записи запрещается. Если оба этих поля пусты – добавление или изменение записи так же не допускается.

Пример 2. Триггер, проверяющий уникальные значения в поле CustomerPassportNum таблиц NaturalPerson и LegalEntity при добавлении или изменении. Если введенные значения этого поля совпадают со значениями, хранящимися в БД, то добавить или изменить такую запись нельзя.

Пример 3. Триггер, который не допускает ввод одинаковых значений в поля ShareCodeSold и ShareCodePurchased при добавлении или изменении записи в таблице Transaction. То есть нельзя в рамках одной сделки продать и

купить один тип акций.

Пример 4. Разработать хранимую процедуру, которая будет по фамилии имени и отчеству клиента выводить всю информацию о его сделках. Входными параметрами хранимой процедуры будут фамилия, имя и отчество клиента. Выходными – записи с полями: номер сделки; дата и время сделки; количество купленных и проданных акций; сумма покупки и продажи акций; ФИО кассира, осуществившего сделку.

Пример 5. Разработать хранимую процедуру, которая будет выводить итоговую информацию по обороту акций за заданное количество дней. Входными параметрами будут две даты, задающие интересующий промежуток времени. Выходными – записи с полями о количестве и сумме проданных и купленных акций.

Генерация схемы БД

Последним шагом является генерация схемы базы данных на основе физической модели. Для этого достаточно кликнуть правой кнопкой мыши на незанятой части диаграммы и из контекстного меню выбрать пункт

Generate SQL (CREATE TABLE) script или нажать кнопку ' на панели инструментов.

В результате будет открыто окно, в котором необходимо ввести имя sql-скрипта и указать папку проекта, куда он будет сохранен (рисунок 10.22)

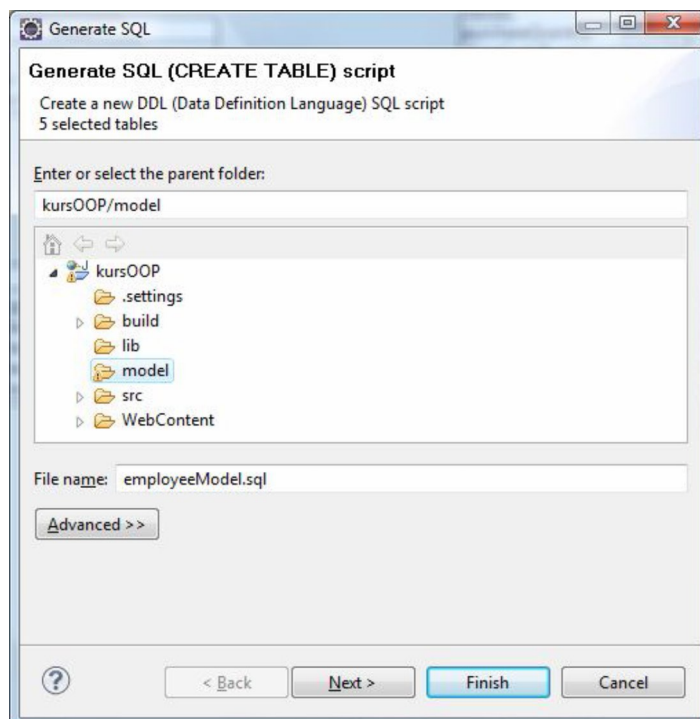


Рисунок 10.22 – Окно ввода имени sql-скрипта

После чего можно выбрать дополнительные параметры генерации схемы БД (рисунок 10.23).

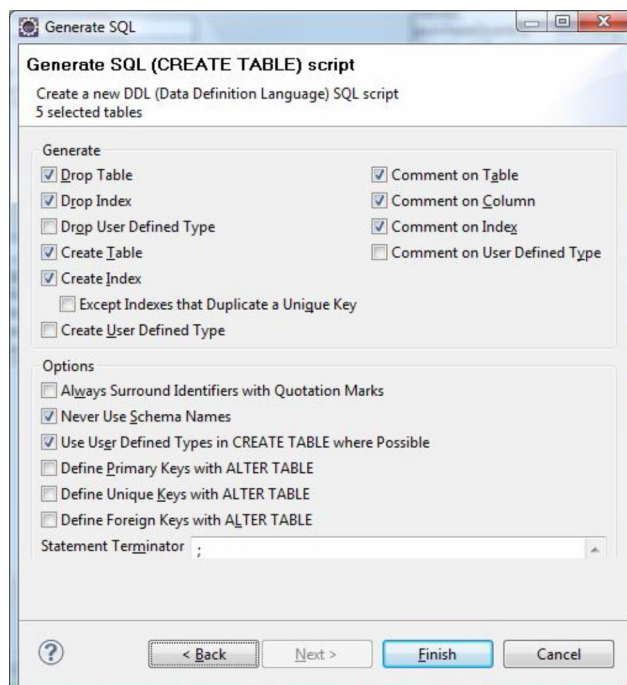


Рисунок 10.23 – Параметры генерации SQL-скрипта БД В результате будет сгенерирован sql-скрипт схемы БД для конкретной СУБД.

Реверсное проектирование

Для запуска процедуры реверсного проектирования необходимо воспользоваться командой Reverse Engineer Database из контекстного меню модели или одноименной кнопкой на панели инструментов.

Рассмотрим процесс реверсного проектирования на примере учебной базы данных Deanery, СУБД PostgreSQL.

1. В открывшемся окне после вызова команды Reverse Engineer Database нажимаем кнопку Add New и создаем новое подключение к БД. В окне редактирования параметров подключения (рисунок 6.24) задаются следующие параметры:

- шаблон подключения (Connection Template) – PostgreSQL;
- алиас подключения (Alias) – dbDeanery;
- URL источника данных, в формате
`jdbc:<имя драйвера>:[//<хост>[:<порт>/]]<имя БД>`
- имя пользователя (User), пароль (Password), имя JDBC драйвера;
- путь к JDBC драйверу (Driver Location)

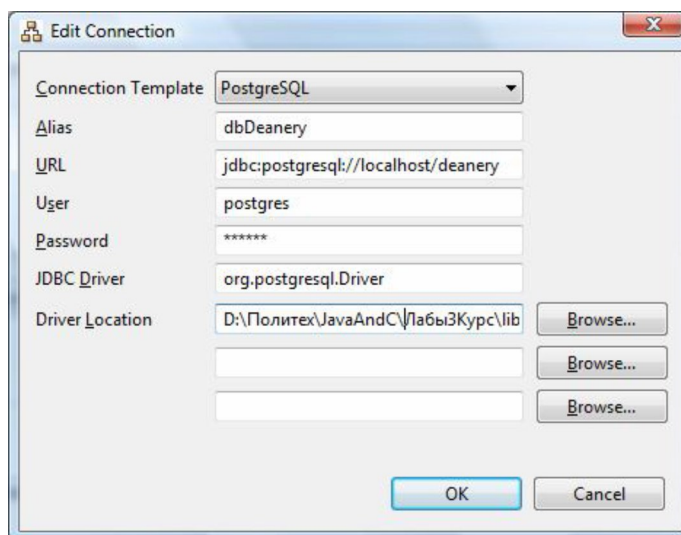


Рисунок 10.24 – Редактор параметров подключения к БД

Нажав кнопку Ok соединение будет создано. После этого нужно нажать кнопку Next и в открывшемся окне выбрать схему с БД (рисунок 10.25).

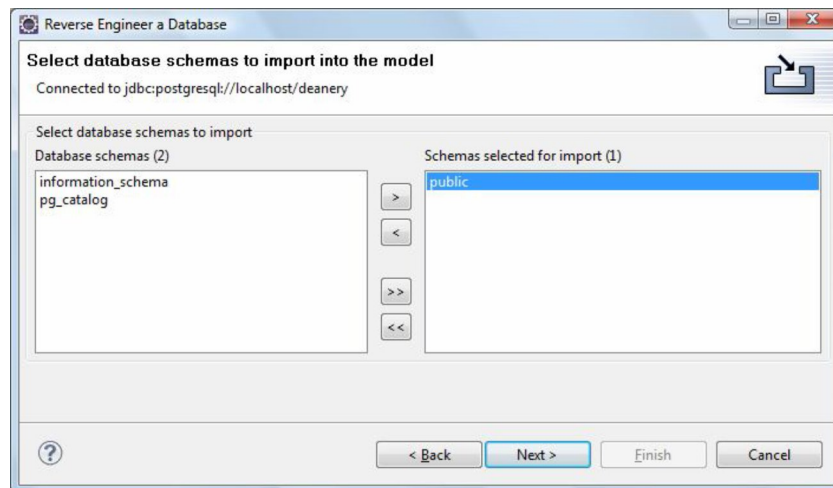


Рисунок 10.25 – Выбор схемы, импортируемой БД

Нажав кнопку Next откроется окно (рисунок 10.26) где нужно выбрать таблицы и нажать кнопку Finish.

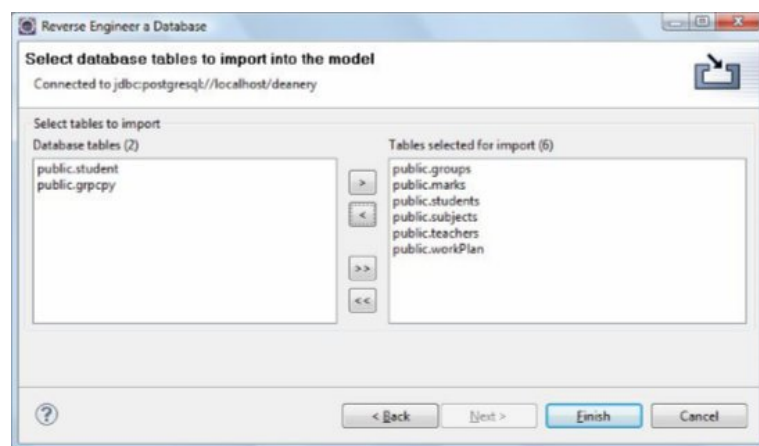


Рисунок 10.26 – Выбор импортируемых таблиц

После нажатия на кнопку Finish запустится процесс генерации схемы БД, получаем ER-диаграмму БД Deanery (рисунок 10.27).

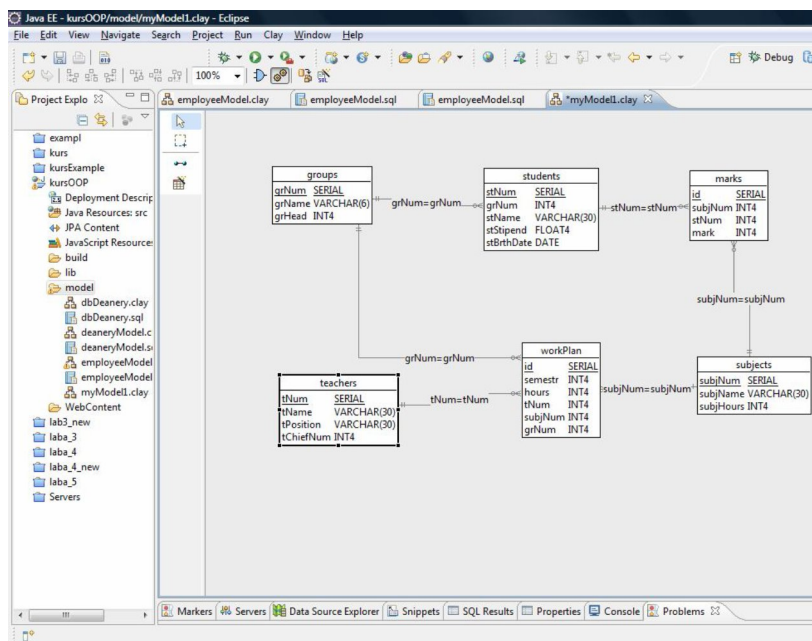


Рисунок 10.27 – Схема БД Deanery

Оборудование и материалы:

Для изучения материалов этой лабораторной работы необходимо:

1. Компьютер под управлением операционной системы Windows 8/10 с параметрами по умолчанию.
2. Установленные на компьютер программы Microsoft Visio, PostgreSQL, MySQL, VMWare/VirtualBox.

3. Текстовый процессор Microsoft Word 2010/2013/2016.

Указания по технике безопасности:

К выполнению лабораторных работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж по технике безопасности.

Задания:

1. Ознакомиться с теоретическими сведениями к лабораторной работе.
2. Скачать и распаковать Eclipse IDE for Java EE Developers (Eclipse Juno 4.2) (ссылка для скачивания <http://www.eclipse.org/downloads/>).
3. Установить плагин Clay Mark II к Eclipse (<http://marketplace.eclipse.org/search/site/clay>).

4. Установить драйвер JDBC для PostgreSQL (драйвер можно скачать по адресу: <http://jdbc.postgresql.org/download.html>).
5. В Eclipse создать новый проект типа Dynamic Web project и в нем создать папку, которая будет содержать модель разрабатываемой БД. Структура проекта приведена на рисунке 10.28.

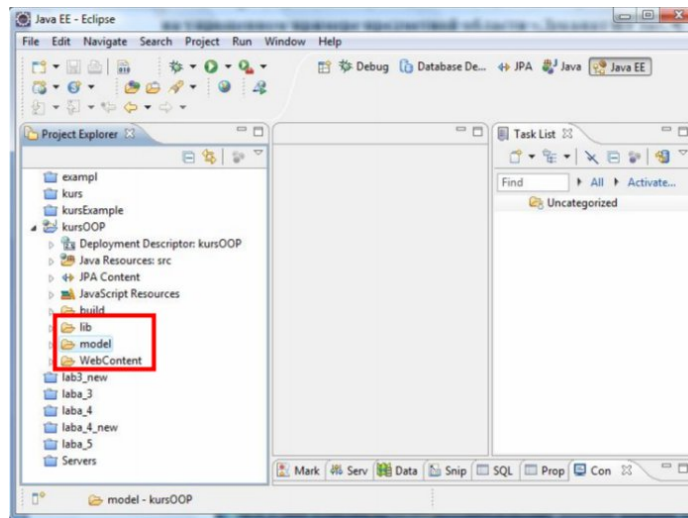


Рисунок 10.28 – Структура проекта

6. Добавить в проект модель: File→New→Database Modeling→Azzurri Clay Database Design Diagram и поместить ее в папку, созданную на предыдущем шаге.

Примечание. Поскольку БД будет храниться на сервере Postgres, то в окошке создания модели для параметра SQL Dialect следует указать PostgreSQL 7.3/7.4 (рисунок 10.29).

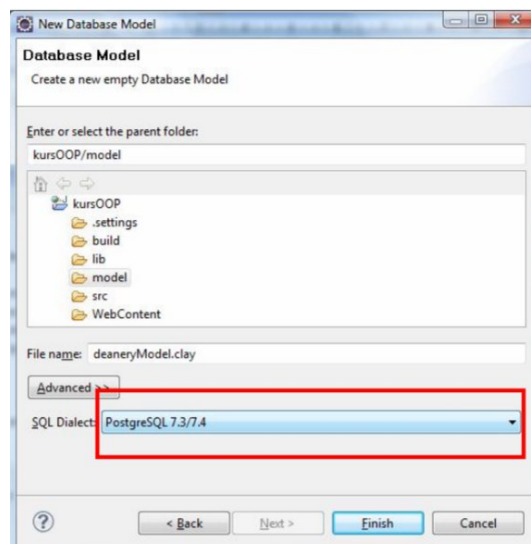


Рисунок 10.29 – Ввод параметров модели

В результате выполнения этих действий откроется редактор модели БД.

7. Согласно заданному варианту, выполнить анализ ПрО, приведенной в таблице 10.2.

Таблица 10.2 – Варианты заданий на выполнения лабораторной работы

Название ПрО	Описание
1. Университет	Объекты: группа; студент; преподаватель; предмет; оценка. Бизнес-логика: подсчет среднего бала для студента при вводе. При отчислении студента отправлять ему сообщение на почту. Отчисление производится автоматически, когда у студента четыре неудовлетворительных оценки. Формы: Список студентов по среднему баллу.
2. Футбольный чемпионат	Объекты: страна; клуб; игроки; матчи. Бизнес-логика: после каждого матча проставлять очки клубу. Отсылать уведомление на почту президенту клуба. В случае проигрыша – отсылка всем игрокам уведомления об уменьшении зарплаты в данном месяце. Формы: таблица чемпионата.
3. Всеукраинская сеть супермаркетов	Объекты: город; супермаркет; продавцы; продукты; поставщики. Бизнес-логика: при уменьшении количества продуктов меньше критической массы автоматически отсылается заказ (почтой) поставщику продукта. Формы: поставщики продукта по увеличению цены.
4. Фирма экстремального спуска с гор	Объекты: гора; средство спуска (сноуборд, лыжи...); мероприятия после спуска(душ, чай, успокоительное); клиент, счет. Бизнес-логика: подсчет общей стоимости спуска с горы, снятие этих денег со счета, а отправка уведомления клиенту. Формы: TOP10 активных клиентов за последний месяц.
5. Корпорация по продаже земельных участков на планетах солнечной системы	Объекты: планета; клиент; участок; расписание рейсов на планету; средство передвижения. Бизнес-логика: подсчитывать остаток свободного места на планете после отнимания площади всех участков. При уничтожении планеты астероидом или захвате внеземными цивилизациями (изменение статуса планеты) – отсылка сообщения всем владельцам участком на планете. Формы: вывод клиентов планеты по убыванию площади участков.
6. Сеть зоопарков	Объекты: город; зоопарк; животное; рабочий; услуги, которые рабочий может оказывать животному (чистка, мойка, расчесывание, кормление). Бизнес-логика: при добавлении нового животного связывать его с работником, у которого наименьшее количество животных такого вида. Также отсылать работнику сообщение на почту. Формы: TOP10 работников месяца.

7. Форум	Объекты: рубрика; тема; сообщение; пользователи, список некультурных слов. Бизнес-логика: фильтрация добавленных сообщений согласно справочнику некультурных слов. Отправка сообщения пользователю о неприятии сообщения. Позволять создавать тему пользователям, если у него более 10 сообщений. Формы: TOP10 активных пользователей.
8. Международная компания «Апельсин+»	Объекты: страна; фрукт; получатель; способ доставки (стоимость за километр); доставка (расстояние). Бизнес-логика: не добавлять информацию о доставке, если время хранения фруктов меньше чем время доставки. Расчет стоимости доставки. Отправлять сообщение получателю о подтверждении доставки. Формы: TOP10 популярных фруктов. TOP10 стран, в которые поставляется наибольшее количество фруктов.
9. Национальная компания «Пилорама»	Объекты: лесник (поставляет дерево); тип дерева; поставка (лесником); склад; выпускаемая продукция; получатель товара. Бизнес-логика: не позволять смешивать в поставках хвойные в лиственные породы древесины. На каждые 10 кубометров древесины лесник добавляет 11-ый кубометр бесплатно. Отправлять сообщение о подтверждении получателю. Формы: наиболее используемая древесина, наиболее активные лесники.
10. Столовая «Советская»	Объекты: блюда; напитки; продукты; меню на день; клиент; заказ. Бизнес-логика: не позволять смешивать мясные и рыбные блюда в одном заказе. В одном заказе не брать блюда из селедки и молочные блюда. После формирования меню отправлять информацию о меню клиентам по почте. Формы: наиболее активные клиенты, наиболее заказываемое блюдо.
11. Фирма по продаже легких наземных экологически-чистых средств передвижения «Машина будущего»	Объекты: средство передвижения (велосипед, самокат, ...); деталь; поставщик; заказ транспортного средства; клиент. Бизнес-логика: подсчет стоимости транспортного средства(стоимость деталей + стоимость сборки), вывод количества транспортных средств, которые могут быть произведены из доступных деталей. Отправлять уведомление клиенту о заказе. Формы: таблица велосипедов, которые можно собрать из доступных деталей, с описанием деталей.
12. Программа по контролю деятельности секретных агентов в зарубежных странах	Объекты: страна; агент; документы, которые переданы; завербованный агентом персонал. Бизнес-логика: не позволять вербовать агенту более восьми людей (за ними становится трудно следить). При переводе кого-либо из персонала в статус «рассекречен» отправлять сообщение (если можно – зашифрованное) в штаб-квартиру об угрозе раскрытия агента. Формы: TOP10 агентов по количеству переданных документов.
13. Программа для контроля МЧС рыбаков, дрейфующих на льдинах	Объекты: море; льдина; рыбак; корабль; мореходство. Бизнес-логика: подсчет количества рыбаков, которые остаются на льдине после спасения некоторого количества рыбаков кораблем. При добавлении информации о дрейфующей льдине отсылать сообщение в мореходство с информацией о количестве рыбаков. Формы: TOP10 льдин, с наибольшим количеством рыбаков.

14. Фирма проката свадебных платьев и аксессуаров	Объекты: платье; фата; цветы; заказ; клиент. Бизнес-логика: не объединять в одном заказе желтый и красный цвет. Если клиент из Японии – не предлагать белые цветы. Отсылать подтверждение на почту клиенту с информацией о заказе. Формы: TOP10 наиболее используемых платьев.
15. Туристическая фирма	Объекты: страна; отель; номер; дополнительные услуги(экскурсия, сафари, посещение театра, луна-парк); клиент; заказ. Бизнес-логика: клиентам из США не предлагать туры в исламские страны. Подсчет стоимости полного заказа. Отсылать подтверждение о заказе клиенту. Формы: наиболее посещаемые отели.
16. Парикмахерская	Объекты: парикмахер; прическа; дополнительные услуги; клиент; заказ. Бизнес-логика: не предлагать дополнительных услуг «бритье» женщинам. При добавлении заказа отсылать сообщение парикмахеру. Формы: список заказов на день.
17. Дендропарк	Объекты: лес (может быть несколько в одном дендропарке), дерево, лесник, школа. Бизнес-логика: если деревьев какого-то вида становится меньше критического количества, отправлять сообщение школе о необходимости предоставления учеников для посадки деревьев. Формы: наиболее полезные школы (выполнили наибольшее количество заказов), состояние лесов с информацией о деревьях.
18. Фирма по разведению аквариумных рыбок	Объекты: аквариум; рыба; рыба-экземпляр (с информацией о весе, длине, массе); корм; заказ на рыбу; клиент. Бизнес-логика: не позволять держать в одном аквариуме хищных и не хищных рыб. Отправлять сообщение клиентам о появлении нового вида рыб. Формы: наличие рыбок по аквариумам, TOP10 заказываемых видов рыб.
19. Общественная организация «Тимуровец»	Объекты: город; агент-тимуровец; потенциальный клиент; вид помощи; помощь(место, время, вид помощи). Бизнес-логика: клиентам, возраст которых менее 60 лет не оказывать вид помощи «перевести через дорогу». Не выполнять определенные виды работ в некоторые времена года (сбор яблок зимой). Если за день выполнено 70 более 10 добрых дел – отправлять сообщение на центральный офис города о выполнении плана. Формы: TOP10 активных агентов-тимуровцев.
20. Служба спасения домашних животных	Объекты: город; животное; заявление об исчезновении; находка животного; клиент. Бизнес-логика: при нахождении животного, всем кто давал заявление об исчезновении такого вида животного отсылать сообщение на почту. Удалять животное из базы, если ее не забрали в течение 30 дней. Формы: поиск в найденных животных экземпляров по каким-либо параметрам.
21. Электронный песенник	Объекты: исполнители; песни; комментарии подружек; пользователи; стиль песни(справочник). Бизнес-логика: изменять цвет фона страницы при каждом добавлении комментария. При добавлении комментария – уведомление пользователю. Формы: TOP10 подружек по количеству комментариев.

22. Корпорация по борьбе с полтергейстом	Объекты: агент; вид полтергейста; город; заказ на расследование. Бизнес-логика: не выбирать агентов-новичков на сложные виды полтергейстов. Не позволять агентам выполнять более 3 заказов в неделю. При добавлении заявки на расследование полтергейста отправлять сообщение всем агентам, которые специализируются по этому виду полтергейста. Формы: статистика выполненных/невыполненных заданий по сотрудникам, статистика заказов и выполненных заданий по городам.
23. Программа по контролю за случаями встречи с НЛО	Объекты: город, участник события; тип НЛО; событие. Бизнес-логика: при поступлении заявки о событии в одинаковое время и в том же месте – создавать одну заявку, присоединяя участников события. При добавлении события на почтовый адрес центрального офиса города отсылать сообщение. Формы: наиболее активные очевидцы НЛО. Наиболее часто появляющееся НЛО.
24. Аукцион	Объекты: товар; клиент, которые покупает; лот; рубрика; проданный товар. Бизнес-логика: не позволять выставлять цену за лот меньше той, которая уже есть сейчас. При закрытии аукциона автоматически добавлять товар в проданный и отправлять почтовое сообщение пользователю. Формы: самые дорогие проданные товары.
25. Доставка пиццы	Объекты: пиццерия, пицца, продукты, заказ, машина, дополнительные блюда. Бизнес-логика: Не позволять создавать заказ на время, которое уже занято. При заказе 10 пицц, 11 бесплатно. Отправлять сообщение пиццерии при оформлении заказа. Формы: Самые популярные пиццы, текущее состояние машин доставки (свободна, занята).
26. Детский оздоровительный лагерь "Отдохни-ка!"	Таблица: заезд детей, отряды, дети, вожатые, мероприятия. Бизнес-логика: В отряде не должно быть больше 30 детей. Для каждого отряда есть ограничение в возрасте. За победу в каждом мероприятии отряд получает очки, и в конце заезда определяется самый лучший отряд. При смене статуса ребенка (прибыл, выехал), отправлять сообщение родителям на почту. Формы: Список мероприятий на заезд, Список очков каждого отряда.
27. Военкомат	Объекты: военкомат (количество требуемых призывников), работник военкомата; бригада по поиску призывников; студент; заявка на выезд к студенту. Бизнес-логика: не позволять добавлять в бригаду работников, которые являются однофамильцами студентов. Не позволять формировать бригады зимой и летом (отсутствие призыва). При поимке студента бригадой, отправлять сообщение на почтовый адрес родителей. Формы: Отслеживание набора призывников (сколько осталось).
28. Авиакомпания "Взлет-посадка"	Объекты: самолеты, города, пилоты, заказ, рейс. Бизнес-логика: Не позволять оформлять заказ при отсутствии свободных мест. В рейсе задается стоимость полета и пилоты. Не позволять пилотам летать чаще 4 раз в неделю. При отмене рейса, отправлять сообщение клиентам, адреса которых указаны в заказах. Формы: наиболее часто посещаемые города.

29. Рыбнадзор	Объекты: рыб-инспектор, лодка рыб-инспектора, справочники рыб и снастей, потенциальные браконьеры, акт о поимке. Бизнес-логика: Одну лодку разрешается комплектовать 2-мя – 4-мя рыб-инспекторами. Автоматически рассчитывать стоимость убытка природе, исходя из массы и вида пойманной рыбы, и используемых снастей. При составлении акта отправлять копию его городской администрации, суду, родственникам браконьера. Формы: топ 10 рыб-инспекторов и браконьеров.
30. Агентство по продаже квартир	Объекты: заявки на продажу, заявки на покупку, агенты по продаже, нотариусы, сделки. Бизнес-логика: При составлении сделки агентом автоматически назначать свободного нотариуса для оформления документов. При поступлении заявки на продажу квартиры отправлять сообщения всем клиентам, которые подали заявку на покупку квартиры с похожими характеристиками, и наоборот. Формы: список продаваемых квартир, составленный на основе заявок на продажу; Топ10 агентов, которые заключили больше всего сделок.

8. Разработать концептуальную (логическую) модель БД в виде ER-диаграммы.

9. Определить ограничения целостности БД.

10. Спроектировать не менее 3-х триггеров (по одному на каждое из событий Insert, Update, Delete). Для триггера обязательно указать событие, на которое он срабатывает, время инициирования, дать описание.

11. Спроектировать не менее 2-х хранимых процедур. Определить входные и выходные параметры, их типы. Дать словесное описание действий, выполняемых хранимой процедурой.

12. Разработать физическую модель БД.

Примечание. Особенность Clay Mark II состоит в том, что этап логического и физического моделирования БД можно выполнять параллельно. При этом необходимо одновременно указывать логические имена сущностей, атрибутов, связей и т.д. и соответствующие им имена таблиц, колонок, связей и т.д.

13. Сгенерировать схему БД.

Содержание отчета:

Отчет о выполнении лабораторной работы должен содержать:

- название и тему лабораторной работы;

- цель лабораторной работы;
- краткие теоретические сведения;
- ход выполнения работы;
- выводы.

Раздел «Ход выполнения работы» должен содержать: анализ ПрО в виде текста (не менее 1 стр.); рисунок с концептуальной моделью ПрО (ER-диаграмма); описание ограничений целостности, накладываемых на данные (можно в виде таблицы); схему БД, полученную по созданной ER-диаграмме в Clay Mark II (физическая модель); сгенерированный SQL-скрипт схемы БД. При описании ограничений обязательно указать:

- уникальность и обязательность ввода первичных ключей;
- поддержка целостности для внешних ключей (каскадное удаление и обновление);
- значения атрибутов по умолчанию и обязательность ввода значений атрибутов;
- ограничения на значения данных атрибутов вида: "интервал", "перечислимое значение" и "сравнение значений двух атрибутов одной таблицы".

Контрольные вопросы:

В данном разделе приведено лишь несколько примеров контрольных вопросов, остальные будут приблизительно такого же содержания.

1. Суть концептуального проектирования БД.
2. В чем суть логического проектирования БД?
3. В чем суть физического проектирования БД?
4. Модель «сущность-связь». Основные компоненты.
5. Процесс построения информационной модели БД.
6. Суть прямого и реверсного проектирования БД.

Список литературы, рекомендуемый к использованию по данной теме:

1. Братченко, Н. Ю. (Северо-Кавказский федеральный университет). Базы данных. Основы работы С СУБД ORACLE 10G EXPRESS EDITION: учеб. пособие: Направление подготовки 230100 – Информатика и

вычислительная техника / Н. Ю. Братченко; Сев.-Кав. федер. ун-т. – Ставрополь: СКФУ, 2013. – 202 с.: ил. – Гриф: Доп. УМО. – ISBN 978-5-88648-877-7

2. Разработка приложений на С с использованием СУБД PostgreSQL / И.А. Васюткина. – Новосибирск: НГТУ, 2015. – 143 с. – ISBN 978-5-7782-2699-9

3. Шацков, В.В. Программирование приложений баз данных с использованием СУБД MS SQL Server / В.В. Шацков: учебное пособие. Электронный ресурс: Санкт-Петербургский государственный архитектурно-строительный университет, ЭБС АСВ; Санкт-Петербург, 2015. – 80 с. – Книга находится в базовой версии ЭБС IPRbooks. – ISBN 978-5-9227-0607-0

4. Голицина, О. Л. Системы управления базами данных: [учеб. пособие] / О.Л. Голицина, Т.Л. Партыка, И.И. Попов. – М.: ФОРУМ, 2011. – 432 с.: ил. – (Профессиональное образование). – На учебнике гриф: Рек.МО. – Библиогр.: с. 315. – ISBN 5-8199-0251-3

5. Гудов, А. М. Базы данных и системы управления базами данных. Программирование на языке PL/SQL / А.М. Гудов; С.Ю. Завозкин; Т.С. Рейн. – Кемерово: Кемеровский государственный университет, 2010. – 134 с. – ISBN 978-5-8353-1005-0

6. Харченко М. В. Организация баз данных. Методические указания к выполнению лабораторных работ для студентов направления 6.050102 – «Компьютерная инженерия». – Чернигов, ЧГТУ. 2013 – 141 с

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации самостоятельной работы по дисциплине
«ОСНОВЫ ПОСТРОЕНИЯ ЗАЩИЩЕННЫХ БАЗ ДАННЫХ» для
студентов специальности 10.05.01 «Компьютерная безопасность»
специализация «Информационно- аналитическая и техническая
экспертиза компьютерных систем»

**Ставрополь
2026**

СОДЕРЖАНИЕ

Введение.....	94
1. Общая характеристика самостоятельной работы студента.....	94
2. Технологическая карта самостоятельной работы студента.....	96
3. Самостоятельное изучение тем лекций.....	96
4. Паспорт фонда оценочных средств для проверки самостоятельной работы.....	97
5. Вопросы для собеседования по самостоятельному изучению тем дисциплины.....	97
6. Список рекомендуемой литературы.....	98

Введение

Целью изучения дисциплины «Основы построения защищенных баз данных» является обучение студентов принципам обеспечения безопасности информации в автоматизированных информационных системах (АИС), основу которых составляют базы данных (БД), навыкам работы со встроенными в системы управления базами данных (СУБД) средствами защиты.

Задачи курса:

- приобретение системного подхода к проблеме защиты информации в СУБД;
- изучение моделей и механизмов защиты в СУБД;
- приобретение практических навыков организации защиты БД.

Компетенции обучающегося, формируемые в результате освоения дисциплины:

1. Общая характеристика самостоятельной работы студента

Основными видами учебной работы по достижению результатов освоения дисциплины являются лекции, лабораторные работы, практические занятия и самостоятельная работа студентов (СРС).

На лекциях раскрываются основные положения и понятия курса, формируются знания в области теории сложности комбинаторных алгоритмов. На лабораторных работах и практических занятиях формируются умения и навыки, необходимые для решения задач профессиональной деятельности.

Приступая к изучению учебной дисциплины, необходимо ознакомиться с учебной программой, получить в библиотеке рекомендованные учебные пособия, а также получить у ведущего преподавателя в электронном виде конспекты лекций, методические рекомендации к лабораторным занятиям и тестовые задания, завести новую тетрадь для конспектирования лекций и выполнения практических заданий.

Для изучения дисциплины предлагается список основной и дополнительной литературы. Основная литература предназначена для обязательного изучения, дополнительная – поможет более глубоко освоить отдельные вопросы, подготовить исследовательские задания и выполнить задания для самостоятельной работы.

В ходе лекционных занятий студент обязан осуществлять конспектирование учебного материала, особое внимание, обращая на категории, формулировки, раскрывающие содержание тех или иных понятий, физических явлений, процессов, эффектов. В рабочих конспектах желательно оставлять поля, на которых следует делать пометки из рекомендованной литературы, дополнять материал прослушанной лекции, формулировать научные выводы и практические рекомендации. Студент имеет право задавать преподавателю уточняющие вопросы с целью уяснения теоретических положений, разрешения спорных ситуаций.

Самостоятельная работа студентов над материалом учебной дисциплины является неотъемлемой частью учебного процесса и должна предполагать углубление знания учебного материала, излагаемого на аудиторных занятиях, и приобретение дополнительных знаний по отдельным вопросам самостоятельно.

Основными видами самостоятельной работы студентов по учебной дисциплине являются:

- самостоятельное изучение учебного материала по заданным темам;
- подготовка к лабораторным занятиям, оформление отчета по выполненному лабораторной работе и подготовка к собеседованию по результатам работы.

Основными методами самостоятельной работы студентов являются:

- 1) для овладения знаниями:
 - чтение текста (конспекта лекций, учебника, дополнительной литературы) и конспектирование текста;
 - работа с нормативными документами;
 - поиск информации в Интернет;
- 2) для закрепления и систематизации знаний:
 - работа с конспектом лекции (обработка текста);
 - повторная работа над учебным материалом (учебника, дополнительной литературы, слайдами);
 - составление плана и тезисов ответа или графическое изображение структуры ответа;
 - составление таблиц для систематизации учебного материала;
 - изучение нормативных документов;
 - ответы на контрольные вопросы;
 - тестирование в программе компьютерного тестирования «Iren»;
- 3) для формирования умений:
 - подготовка к лабораторным занятиям;
 - решение задач и практических заданий;
 - подготовка отчетов по лабораторным занятиям;
 - рефлексивный анализ профессиональных умений.

В случае пропуска учебного занятия студент может воспользоваться содержанием различных блоков учебно-методического комплекса (лекции, практические занятия, контрольные вопросы и тесты) для самоподготовки и освоения темы. Для самоконтроля необходимо использовать вопросы и задания, предлагаемые к лабораторным занятиям.

2. Технологическая карта самостоятельной работы студента

2. Технологическая карта самостоятельной работы студента					
Коды реализуемых компетенций, индикатора(ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателем	Всего
9 семестр					
ИД-3 ОПК-14 ИД-2 ОПК-14 ИД-1 ОПК-14 ИД-1 ОПК-11	Подготовка к практическому занятию	Собеседование, проверка конспектов	15	15	30
ИД-3 ОПК-14 ИД-2 ОПК-14 ИД-1 ОПК-14 ИД-1 ОПК-11	Подготовка к лабораторной работе	Собеседование, проверка конспектов	14	10	24
Итого за 9 семестр			29	25	54
Итого			29	25	54

3. Самостоятельное изучение тем лекций

№ п/п	Темы для самостоятельного изучения	Рекомендуемые источники информации (№ источника)			
		Основная	Дополнительная	Методическая	Интернет-ресурсы
1.	Тема: Механизмы обеспечения конфиденциальности в СУБД	2	1,2	1, 2	1, 2,3
2.	Тема: СУБД MariaDB (MySQL).	1	1, 2, 4	1, 2	1, 2,3

4. Паспорт фонда оценочных средств для проверки самостоятельной работы

Код оцениваемой компетенции (или её части)	Этап формирования компетенции (№ темы)	Тип контроля	Вид контроля	Количество элементов, шт.	
				Базовый	Повышенный
ИД-3 ОПК-14 ИД-2 ОПК-14 ИД-1 ОПК-14 ИД-1 ОПК-11	1-2	текущий	Вопросы для собеседования по самостоятельному изучению тем дисциплины	4	2

5. Вопросы для собеседования по самостоятельному изучению тем дисциплины Тема 1. Механизмы обеспечения конфиденциальности в СУБД

Базовый уровень:

1. Что такое СУБД?
2. Как обеспечить конфиденциальность информации в СУБД?

Повышенный уровень:

3. Основные принципы защиты информации в СУБД.

Тема 2. СУБД MariaDB (MySQL) Базовый уровень:

1. Что такое СУБД MariaDB?
2. Чем СУБД MariaDB отличается от обычной СУБД?

Повышенный уровень:

3. Как обеспечить конфиденциальность информации в СУБД MariaDB?

Критерии оценки для проверки уровня обученности:

Оценка «отлично» выставляется студенту, если студент показал глубокое, прочное и аргументированное освоение программного учебного материала, при этом поставленный вопрос раскрыт последовательно, четко и логически стройно, в полном исчерпывающем объеме, основные категории, понятия и термины учебного курса формулировались правильно, не допущено при ответе ошибок.

Оценка «хорошо» выставляется студенту, если студент показал твердое знание программного учебного материала, при этом поставленный вопрос раскрыт грамотно и по существу, в достаточно полном объеме, основные категории, понятия и термины учебного курса формулировались правильно, допущены при ответе отдельные неточности или одна ошибка.

Оценка «удовлетворительно» выставляется студенту, если студент

показал знание только основной части учебного материала без его частных деталей, при этом поставленный вопрос раскрыт с нарушением логической последовательности, не в полном объеме; были допущены неточные формулировки основных категорий, понятий и терминов учебного курса, а также ошибки (не более двух) или ряд незначительных неточностей, не искаживших существенно суть ответа.

Оценка «неудовлетворительно» выставляется студенту, который не знает значительной части программного материала, допускает существенные ошибки (более двух), существенно искажившие его суть. Оценка неудовлетворительно выставляется также, если отсутствует ответ на вопрос, либо студент отказался его сдавать.

6. Список рекомендуемой литературы Основная литература:

1. Колисниченко Д. PHP и MySQL. Разработка Web-приложений.- БХВ-Петербург,
2. Бекаревич Ю. Access 2010. - БХВ-Петербург, 2013.
3. Иванова И, М.М. Корниенко. Базы данных. Системы управления базами данных .-БХВ-Петербург, 2011.
4. Колисниченко Д.Н. PHP 5/6 и MySQL 6. Разработка Web-приложений.- БХВ- Петербург, 2013.
5. Новиков Б.А. Настройка приложений баз данных. Гриф УМО МО РФ. -БХВ- Петербург, 2012.
6. Дунаев В.В. Базы данных. Язык SQL для студента.- БХВ-Петербург, 2012.
7. Колисниченко Д. Н. PHP и MySQL. Разработка веб-приложений.- БНВ, 2015.
8. Благодаров А. В. Алгоритмы категорирования персональных данных для систем автоматизированного проектирования баз данных информационных систем Издатель - БНВ, 2015.

Дополнительная литература:

1. Кузнецов С.Д. Основы современных баз данных. Учебник. - <http://www.citforum.ru/database/osbd/contents.shtml>
2. Кириллов В.В. Основы проектирования реляционных баз данных. Учебное пособие. – <http://www.citforum.ru/database/dbguide/index.shtml>
3. Носков Ю.М. Система программирования Delphi. - раздел «Основы работы с базами данных». – <http://www.mgoru.ru/PVU/2.1/Delphi/index.html>
4. Пушкинов А.Ю. Введение в системы управления базами данных. Учебное пособие/Изд-е Башкирского ун-та. - Уфа, 1999. - 246 с. - <http://www.citforum.ru/database/dblearn/index.shtml>
5. Козленко Л. Информационная безопасность в современных системах управления базами данных. - КомпьютерПресс 3'2002. - http://www.citforum.ru/security/articles/safe_db/
6. Гайдамакин Н.А. Автоматизированные информационные системы, базы и банки данных. Вводный курс: Учебное пособие. – М.: Гелиос АРВ,

2002.

7. Гайдамакин Н.А. Разграничение доступа к информации в компьютерных системах. – Екатеринбург: Изд-во Урал. Ун-та, 2003. – 328 с.

8. Базы данных. Учебник для вузов под ред. А. Хомоненко. – М.: Корона Принт.

9. Вьюкова Н.И., Галатенко В.А. Информационная безопасность систем управления базами данных. СУБД, 1996, № 1.

10. Белкин П.Ю., Михальский О.О., Першаков А.С. и др. Защита программ и данных: Учебное пособие для ВУЗов. – М.: Радио и связь, 1999.

11. Кузнецов С.Д. СУБД и файловые системы. – М.: Майор, 2001.

12. Вендров А.М. CASE-технологии. Современные методы и средства проектирования информационных систем. - <http://www.citforum.ru/database/case/index.shtml>.

Методическая литература:

1. Электронный курс лекций.
2. Электронные методические рекомендации для выполнения лабораторных заданий.

Интернет-ресурсы:

1. Курсы информационных технологий Яндекса (КИТ) <https://academy.yandex.ru/events/kit/5/>
2. Базы данных ("обзорно-понятийный курс", создатели: Яндекс, CS клуб при ПОМИ РАН и т.д.): (2012г.) - <http://www.lektorium.tv/course/?id=22894>

Программное обеспечение:

1. Microsoft Visio 2012
2. Операционная система Linux
3. (PostgreSQL)
4. MariaDB (MySQL).

Материально-техническое обеспечение дисциплины

1. Мультимедийная лекционная аудитория № 431.
2. Аудитории, оснащённые компьютерами с установленным требуемым программным обеспечением, для проведения лабораторных работ.