МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАН

НУКУССКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ИМЕНИ МУХАММАДА АЛ-ХОРАЗМИЙ

Кафедра «Информационные технологии»

направление Компьютерный инжиниринг(«Компьютерный инжиниринг»)

Допускается к защите
Заведующий кафедрой
Туреniязова А.
_____
«____» _____ 2019 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему

«Создание базы данных SQLite для учета ресурсов ZiyoNet в учебных заведениях»

Выполнил: Джумамуратов А.

Научный руководитель: Баекеев Т.

НУКУС - 2019 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время жизнь человека зависит от различного рода информации, для управления которой требуются создания огромного количества баз и банков данных. Одним из ключевых отраслей в области автоматизация управление данными с использованием современной информационных технологий является разработка баз данных. База данных может решить проблему хранения и систематизации информации согласно по различным требованиям. Основные идеи современной информационной технологии основывается на концепции, которой данные должны быть организованы в виде базы данных. Эти базы данных создаются и функционируют под управлением специальных программных комплексов, называемых системами управления базами данных (СУБД). С увеличением объёма и сложности данных стало расширяться круг пользователей информационных систем. Это привело к широкому распространению удобных и сравнительно простых для понимания реляционных СУБД. Для обеспечения одновременного доступа к данным множества пользователей, нередко расположенных достаточно далеко друг от друга и от места хранения баз данных, созданы сетевые мультипользовательские версии БД основанных на реляционной структуре. В них тем или иным путем решаются специфические проблемы параллельных процессов, целостности (правильности) и безопасности данных, а также санкционирования доступа.

Цель данной выпускной квалификационной работы – это проектирование и создание базы данных для ведение работы с учебно-образовательными ресурсами портала ZiyoNet в ВУЗе. В данной работе в качестве объекта выбран внутренний портал ZiyoNet Нукусский филиал Ташкентского университета информационных технологий имени Мухаммад ал-Хорезми.

Для достижения поставленной цели необходимо решить следующие задачи:

- Изучить сферу деятельности;
- Изучить положение ведение работы с ZiyoNet;
- Изучить необходимые данные ресурсов;
- Проектирование базы данных;
- Выбор СУБД для создание базы данных;
- Создание базы данных;
- Создание отчетов;
- Тестирование базы данных.

Материалы в выпускной квалификационной работе располагаются в той последовательности, в которой происходило его изучение и обработка. Работа состоит из введения, 3 глав, заключении, приложении и списков литературы.

В первой главе изучается сфера деятельности, дается информация о базы данных, изучается ведение работа с порталом ZiyoNet. Определяется необходимые данные и типы ресурсов. Изучается требование к базе данных.

Во второй главе изучается концепция базы данных и СУБД, дается информация о реляционной базы данных, правило Кодда, нормализация данных. Проектируется база данных. Определяется необходимые таблицы и их поля. Определяется связи между таблицами. Создается модель базы данных.

В третьей главе изучается и определяется инструменты для создание базы данных. Дается информация по выбранному СУБД. Создается база данных по проекту. Создается отчеты и тестируется база данных в целом.

В заключении приводятся результаты создание базы данных и достижения достигнутое в ходе выполнение выпускной квалификационной работы.

ГЛАВА I. ВАЖНЫЕ СВЕДЕНИЯ И НЕОБХОДИМЫЕ ДАННЫЕ

1.1 Описание сферы изучение

Развитие информационных технологий сопровождается еще двумя любопытными тенденциями в том, что касается терминологии. С одной стороны, мы наблюдаем постоянное обновление названий, в общем-то, одних и тех же вещей (конечно, технологии тоже развиваются, но темпы смены их названий намного выше). С другой - мы используем старые термины для понятия, смысл которого уже совсем не тот, что раньше.

В толковых словарях по вычислительной технике, выпущенных в 2002 году, встречается такое определение базы данных системы управления (как СУБД или системы управления базами данных): «экспонат, обеспечивающий создание, хранение, обновление и поиск информации в базе данных, а также управление безопасностью и целостность данных». В целом это толкование было верно 30 лет назад, но вся глубокая часть СУБД в наше время совсем иная, чем в те далекие времена.

В последнем десятилетнем событии мы наблюдаем ситуацию, когда СУБД превратилась из сугубо внутренних технологических дополнений в прикладную программу в самостоятельном продукте, вокруг которой строятся приложения для пользователей; другими словами, из одного компонента информационной системы - в платформу для построения таких систем.

В этой ситуации стоит обратить внимание на содержание «платформы Microsoft», чтобы измениться. Традиционно под этим термином понималась операционная система Windows. Однако применительно к серверной платформе все чаще встречается связка Windows Server + SQL Server. Более того, совершенно реально, что с выходом в начале следующего года к новой версии Microsoft SQL Server (рабочее имя Yukon) мы сталкиваемся с ситуацией, когда все остальные продукты Microsoft будут написаны уже не под Windows, а под Yukon. Хотя существует и будет существовать настольная

база данных: как ни странно, но Microsoft Access, судя по тому, что он представляет Microsoft, - тоже СУБД.

Исторически системы управления базой данных были ориентированы на решение задач, связанных, в первую очередь, с транзакционной обработкой для выделения информации. Безусловно, лучшим, проверенным временем решением здесь была и остается реляционная модель СУБД. Однако в последние годы база данных приложений неизменно расширялась. С одной стороны, необходимо контролировать более широкий набор форматов данных, идя на решение общих задач корпоративного управления информацией. С другой - именно СУБД берут на себя основные функции по интеграции данных и приложений корпоративных систем. (По данным Gartner Group, информационное подразделение предприятия тратит до 40% своего бюджета на решение задач, связанных с базой данных компонента интеграции). Именно этим объясняется активный интерес к обсуждению принципа архитектуры и возможностей реализации различных моделей базы данных - пост реляционной, объектно-ориентированной, реляционный, XML.

Если попытаться классифицировать существующее приложение как базу данных, но таким же образом оценить перспективы их развития в настоящее время, то можно получить приблизительный список наиболее распространенных классов, распространяемых и используемых во всей базе данных приложения. Этот список будет выглядеть следующим образом:

- Документальные и документальные материалы используются во всех базовых органах власти и управления.
- база данных по промышленной, строительной и сельскохозяйственной продукции (статистическая, кредитно-финансовая, внешняя торговля)
- фактографические основы социальных данных, в том числе информация о населении и о социальной среде
- база данных транспортной системы

- справка для населения и учреждений (энциклопедии и справочники, расписания самолетов и поездов, адреса и телефоны людей и организации)
- база данных ресурсов, включая фактографическую информацию о природных ресурсах (земля, вода, недра, биологические ресурсы, прогноз погоды и отходы, экологическая ситуация)
- фактографические базы и банки научных данных, обеспечивающие фундаментальные научные исследования
- фактографическая база данных в области культуры и искусства
- лингвистическая база данных, то есть машинный словарь разных видов и целей.

Экономические задачи, для решения которых необходимо использовать программные СУБД, более обширны и разнообразны. На его базе построены информационные системы предприятия разного уровня (от маленьких до больших). База данных приложений традиционно занимает ту область деятельности человека, где он сталкивается с большим объемом различной информации. Первые базы данных в основном использовались в таких фундаментальных науках, как физика ядра, химия, космонавтика и другие науки, требующие системного подхода к работе с данными. Дальнейшее развитие компьютерные технологии и компьютеризация общества привели к тому, что базы данных стали развиваться практически во всех сферах деятельности человека, и применяя предпринимательство в разных областях от сельского хозяйства до финансово-экономических систем. Последними нововведениями в использовании базы данных стала всемирная сеть Интернет, которая сама по себе является огромной базой данных. Соответственно для такого распространения базы данных требуются и новые программные средства управления ими.

Как уже упоминалось выше, мы можем сделать вывод, что для эффективного управления любой сферой использование информационных

технологий считается продуктивным способом. Кроме того, использование базы данных для работы с информацией может обеспечить эффективность и безопасное время.

Основная цель базы данных является повышение эффективности при работе с большими данными. В нашем работе мы рассмотрим создание базы данных для учета ресурсов портала ZiyoNet.

ZiyoNet является огромным порталом информационно-образовательным сети в Республике Узбекистан. Основной целью сети ZiyoNet является содействие внедрению в систему образования широкого комплекса информационно-коммуникационных услуг для учащихся и молодежи республики. Портал Информационной образовательной сети ZiyoNET включает в себя всю необходимую информацию для молодежи, преподавателей, а также других слоев населения и представляет собой комплекс удобных инструментов, необходимых для получения интересующей информации, приобщения к современным навыкам в сфере ИТ, общения и обмена опытом, и много другого.

С 2015 года каждый ВУЗ начал пополнят портал своими информационно-образовательными ресурсами. И в каждом ВУЗе должно содержится копии этих же ресурсов в виде базы данных. Поэтому мы выбрали это объект для разработки базы данных с использованием современных технологий.

Прежде нам надо определить данные, которые содержится в базе данных. Информационно-образовательные ресурсы разделяются на несколько категорий:

1. Лекции
2. Учебно-методический указание
3. Презентации
4. Курсовые работы
5. Рефераты и отчеты самостоятельных работ
6. Научные статьи

7. Выпускные квалификационные работы
8. Магистерские диссертации

После определение категорий материалов нам надо определять данные для каждого ресурса. Эти данные:

1. Название ресурса
2. Автор ресурса
3. Год печати
4. Количество страниц
5. Аннотация
6. Сфера образования
7. Ответственное лицо
8. Название файла

В каждом ВУЗе по несколько факультетов и кафедр. Поэтому надо определять каждого ресурса, в какой кафедре относиться. Для этого мы вводим базу данных новые поля кафедры. С помощью этой же поле, позже мы можем подготовить отчеты по каждой кафедре. Пополнение базы данных осуществляется в каждом месяце. Работа с базы данных осуществляется с одного место и со стороны одного ответственного лица. Поэтому не требуется работы базы данных в клиент-серверном режиме.

1.2 Требование к базе данных

Правильно спроектированная база данных должна удовлетворять следующим требованиям:

1. Минимальная избыточность. Согласованность.
2. Полнота данных.
3. Независимость данных.
4. Возможность ведения (сопровождения и удаления) и актуализации (корректировки, модификации) данных.
5. Безопасность и секретность.
6. Высокая эффективность. Минимальные затраты.

7. Соблюдение стандарта .

1. Минимальная избыточность означает, что данные в базе данных не должны дублироваться. Если существуют избыточные данные, это создает две опасности:

- неоправданно большие затраты на память и сокращение времени отклика системы при обработке чрезмерно больших объемов данных.
- нарушение согласованности данных, т. е. возникновение таких ситуаций, когда в разных местах памяти машины хранятся противоречивые данные. Происхождение последовательности чрезвычайно опасно для базы данных.

Противоречие может появиться в результате корректировки избыточных данных. При внесении изменений в логическую запись может произойти так, что отдельные копии этой записи, хранящиеся в разных местах памяти машины, окажутся неисправленными. Программист, оказывается, подчеркивает организациям процесс корректировки избыточных данных и разрабатывает специальные программы, предотвращающие появление противоречивых.

Противоречивость может появиться и при корректировке неустранимых данных. Хранение данных на сайте является причиной высокой вероятности того, что для двух или более пользователей одновременно необходимы одни и те же данные. Если один из пользователей обращается к данным, но другой в то же время вносит в них изменения, будут получены противоречивые данные. Объясняется это тем, что этот процесс обновления данных требует определенного времени, в течение которого одни и те же данные оказываются на разных этапах обновления. При ссылке на такие данные параллельные рабочие программы будут получены противоречивые сведения

В СУБД существуют сложные механизмы блокировки обновленных данных от доступа к нему других пользователей. Параллельные запросы к одному и тому же заданию обычно выполняются последовательно.

В строке СУБД есть средство, предотвращающее дублирование и возникновение согласованности данных. В противном случае такой объект разрабатывает слесарь

2. Полнота данных означает, что в базе данных должны храниться только правильные данные, то есть в них сохраняются логические условия, в соответствии с которыми данные считаются правильными. Разрушение и искажение данных возможно в результате неосторожных действий пользователей, в результате ошибки в программе и неисправностей оборудования.

Существуют особые методы и получение обеспечения целостности. Для обеспечения целостности данных, хранящихся в базе данных, накладывают ограничения. Являются ли они при этом определенным условием, которое должно соответствовать данным о важности.

Например, одна и та же порция не может иметь два разных года рождения и т. Д. Аналогичные ограничения определены в базе данных закона. Удовлетворенность базы данных законов проверяется сезонно СУБД.

Для предотвращения возможности ввода неверных данных разработана возможность проверки правильности введенных данных. Так, например, возможно использование процедур, проверяющих атрибуты введенных значений, определяемый диапазон возможных значений. Например, количество рабочих дней ограничено сверху количеством дней в текущем месяце.

Данные целостности могут быть нарушены при непреднамеренном прекращении транзакций. Сделки идентифицируются определенной неделимой оперативной процедурой по данным, выполненной по одному запросу в БД. Примером для транзакций является операция перевода денег с одного счета на другой в банковской системе. Здесь необходимо последовательное выполнение нескольких операций. Денежный отпуск с одним счетом, если данные исправлены, затем деньги добавляются к другому

счету, а данные вновь исправляются. Если хотя бы одно из действий не выполнено успешно, результат транзакций окажется недействительным. СУБД должна проследить перемещение выполнения к транзакциям от начала до ее завершения. Если по какой-либо причине одна из операций не была выполнена, то транзакции отменяются полностью. При этом «отдача» выполняется путем отмены всех уже выполненных изменений.

В БД должна быть предусмотрена возможность проведения данной реконструкции после программных сбоев и неисправностей оборудования. Существуют программы создания резервных копий и специальные программы, которые автоматически исправляют любые внесенные в БД изменения (в файле корректуры). Если работающая версия БД испорчена, то есть предыдущая версия, в нее вносится изменение, фиксируется в файле корректуры, и текущее (фактическое) состояние БД восстанавливается.

Различные СУБД в той или иной мере располагают возможностью предоставления данных целостности. В противном случае такие средства разрабатываются инструменталистами .

3. Независимость данных означает, что прикладные программы не должны зависеть от предварительно сохраненных данных, то есть от способа хранения данных в физической памяти. Это позволяет добавлять в БД новые данные, изменять структуры хранения данных, создавать на БД новые приложения. При этом ранее созданная программа не должна «чувствовать» эти изменения.

4. Структура БД должна позволять включать новые и удалять устаревшие данные, исправлять предварительно заданные данные без разрушения логических связей, установленных в схеме БД. По этой схеме БД должна быть правильно спроектирована, но операции по ведению БД не должны нарушать схему БД.

5. Безопасность и секретность означают защиту от несанкционированного доступа, преднамеренного и непреднамеренного

уничтожения данных, данных о незаконном присвоении. Система защиты БД призвана решать следующие задачи.

- Идентификация пользователей. Данные, хранящиеся в БД, должны использовать только лица, имеющие на это право и подтвержденные своими полномочиями. Самый распространенный способ решения этой задачи - это система паролей.
- Ограничение доступа к данным. Каждый пользователь должен работать только с теми данными, которые необходимы для решения его задач, остальные данные должны быть для него «не встречены».

Каждому пользователю предоставляются определенные полномочия (привилегии) для работы с данными. Ему могут быть предоставлены права только чтения из БД, права входа в БД или права обновления и т. Д. Все привилегии предоставляются только администратору БД

Защита конфиденциальности данных. Секретные данные необходимо защищать от доступа к системе специальными, достаточно сложными паролями. Мощно, уязвимые данные следует кодировать.

Значения защиты и обеспечения безопасности данных содержатся в СУБД или разрабатываются инструкторами .

6. Организация БД и методы доступа к данным должны обеспечивать высокую скорость обработки данных, чтобы пользователь мог работать с БД в диалоговых режимах. Стоимость обслуживания пользователей не должна быть высокой. Возможность выполнения этих требований определяется рядом факторов: объем предварительно хранимых данных, скорость технологии, а также способ организации данных в БД и во многом зависит от решений, принятых разработчиком на этапе создания БД. Например, возможно организовать режим местоположения, заданный на носителе, тем самым, что наиболее часто используемый заданный сохраняется в самой доступной области внешней памяти .

7. Представленная в БД презентация, сопроводительная документация, способ взаимодействия пользователя с БД должны соответствовать определенному стандарту. Стандарты могут быть корпоративными, ведомственными, отраслевыми, национальными и международными. Стандарт соблюдения абсолютно необходим для данных совместного использования и для организации обмена данными между отдельными системами.

ГЛАВА II. ПЛАНИРОВАНИЕ БАЗЫ ДАННЫХ

2.1 Концепция базы данных и СУБД

База данных (BD, база данных) - именованная совокупность общих данных, относящихся к определенному домену приложения. Домен приложения - определенная часть реально существующих систем, функционирующая как независимая единица. Полный домен приложения может представлять собой экономику страны или группы союзного государства. Однако на практике для информационных систем наибольшее значение имеет область применения масштаба отдельного предприятия или корпорации.

База данных системы управления (СУБД) - комплекс программных и языковых средств, необходимых для создания и модификации базы данных, сопровождения, модификации, удаления, поиска и выбора информации, представления информации на экране и в печатном виде, разграничения прав доступ к информации, выполнение других операций с базой.

Реляционная БД - основной тип современной базы данных. Состоит из таблиц, между которыми могут существовать отношения по ключевым значениям.

Таблица базы данных (таблица) - это регулярная структура, которая состоит из родственных строк (записи, записи), разделенных на столбцы (поля, поля). В теориях реляционной базы данных синоним таблицы - отношение (отношение), в строке которого указывается кортеж, а в столбце - атрибут.

В концептуальной модели реляционной БД аналогом таблицы является сущность (сущность), определяемая заданной характеристикой - атрибутом, способным принимать определенный знак (множество возможных значений - домен).

Ключевой элемент таблицы (ключ, обычный ключ) - такое ее поле или строковое выражение, образованное из значений нескольких полей (ключ компонента), по которым можно определить значения другого поля для одного

или несколько записей таблицы. На практике индексы для использования ключей - это служебная информация, содержащая ранжированную информацию о ключевых значениях. В реляционной теории и концептуальной модели понятие «ключ» используется для атрибутных отношений или сущности.

Первичный ключ (первичный ключ) - элемент основного ключа, однозначно идентифицирующий строку в таблице. Также могут существовать альтернативные (ключ-кандидат) и уникальные (уникальный ключ) ключи, служащие также для идентификации строк в таблице.

В теории отношений первичный ключ - минимальный набор атрибутов, однозначно идентифицирующий кортеж в отношении. В концептуальной модели первичный ключ - минимальный набор атрибутов сущности, однозначно идентифицирующий копию сущности.

Связь (отношение) - это функциональная зависимость между объектом. В реляционной базе данных фиксируются отношения между таблицей по ключу, одна из которых в основной (родительской, родительской) таблице - основной, вторая - во внешнем ключе - во внешней (дочерней, дочерней) к таблице, как правило, основной нет и формирует отношения «один ко многим» (1: N). В случае первичного внешнего ключа связь между таблицами имеет тип «один к одному» (1:1). Информация о взаимосвязях сохраняется в базе данных. Внешний ключ (внешний ключ) - ключевой элемент, подчиненный (внешний, дочерний) таблицы, важность которого соответствует важности первичного ключа main (родительского) таблицы. Заданная целостность ссылки (ссылочная целостность) - набор правил, обеспечивающих соответствие ключевых значений в связанной таблице.

Хранимые процедуры (хранимые процедуры) - это программные модули, сохраненные в базе данных для выполнения определенной операции с информацией базы.

Триггеры (триггеры) - предварительно созданные процедуры, обеспечивающие условия для ссылки на целостность, заданную в операции

изменения первичных ключей (возможное изменение каскада), удаление записи из основной таблицы (удаление каскада в дочерней таблице) и сопровождение записи или изменение дано в дочерней таблице.

Объект (объект) - это элемент информационной системы, обладающий определенной характеристикой (свойствами) и определяемой образом на внешних событиях (событиях).

Система - это совокупность, взаимодействующая между собой и с внешним объектом окружения.

Репликация базы данных - это создание базы данных (примечания), которая может быть изменена обновленными данными или реплицированными формами, отчетом или другим объектом в результате выполнения процесса синхронизации.

Транзакции - это изменение информации в базе в результате выполнения одной или их последовательности, которая должна быть выполнена полностью или не выполнена вообще. В СУБД существуют специальные механизмы обеспечения транзакций.

Язык SQL (Structured Query Language) - универсальный язык работы с базой данных, включающий в себя возможность ее создания, изменения структуры, выбора по запросу, изменения информации в базе и других операций манипулирования базы данных.

Null - значение поля таблицы, показывающее, что информация в данном поле отсутствует. Разрешение на возможность существования важности Null можно назначить для отдельного лоскута стола.

Реляционная база данных

Реляционные системы далеко не сразу получили широкое распространение. Хотя основной теоретический результат в этой области был получен еще в 70-х годах, а затем появились первые прототипы реляционных СУБД, долгое время считалось невозможным получить эффективную реализацию таких систем. Однако постепенное накопление методов и

алгоритмов для организации реляционных баз данных и управления ими привело к тому, что уже в средних 80-х реляционные системы практически вытеснили с мирового рынка ранние СУБД.

Данная реляционная модель основана на математическом принципе, вытекающем непосредственно из теории множеств и логического предиката. Эти принципы впервые применялись в области моделирования, приведенной в конце 1960-х гг. Доктор Э. Ф. Коддом в то время работал в IBM, но впервые опубликован в 1970 году.

Техническая статья «Реляционная модель, приведенная для больших разделенных банков данных» доктора Э. Ф. Кодда, опубликованная в 1970 году, является прародителем современной теории реляционных БД. Доктор Кодд определил 13 правил для реляционной модели (которые называют 13 правил Кодда).

13 правил Кодда

1. Реляционная СУБД должна быть способна полностью контролировать базу данных через ее реляционные возможности.
2. Правило информации - вся информация в реляционной БД (включая имена таблиц и столбцов) должна определяться строго как значения в таблице.
3. Гарантированный доступ - любая важность в реляционной БД должна быть гарантированно доступна для использования через комбинацию имени таблицы, важности первичного ключа и имени столбца.
4. Поддержка пустых значений (нулевое значение) - СУБД должна знать, как работать с пустыми значениями (неизвестные или неиспользуемые значения), в отличие от значений по умолчанию и для любых доменов независимо.
5. Онлайн реляционный каталог - описание БД и ее содержимого должно быть представлено на логическом уровне в виде таблиц, к которым можно обращаться с запросами, используя языковую базу данных.

6. Предоставлен всеобъемлющий язык управления - по крайней мере, один из поддерживаемых языков должен иметь четко определенный синтаксис и быть всеобъемлющим. Он должен поддерживать описание структуры данных и манипулирование ими, правила полноты, авторизации и транзакций.
7. Правило обновления презентаций (представлений) - все презентации, теоретически обновленные, могут обновляться через систему.
8. Вставка, обновление и удаление - СУБД поддерживает не только запрос данных для выбора, но также вставку, обновление и удаление.
9. Физическая независимость дана - на программных приложениях и специальных программах логически не влияют изменения физического метода доступа к данным и структурам хранилища данных.
10. Приведенная логическая независимость - на программные приложения и специальные программы логически не влияют, в пределах разумного, на изменение структуры таблиц.
11. Независимость от целостности - языковая БД должна быть способна определять правила целостности. Они должны быть сохранены в онлайн-справочнике и не должны существовать таким образом, чтобы их избежать.
12. Независимость от дистрибутива - на программные приложения и специальные программы логически не влияет, сначала используются однажды данные или снова.
13. Непрерывность - невозможность избежать правил целостности, определенных посредством языковой базы данных, использования языков низкого уровня.

Реляционная алгебра

Основная идея реляционной алгебры состоит в том, что, если вскоре отношения становятся ансамблем, возможность манипулирования отношениями может основываться на традиционной теории множественного

числа, дополненной некоторой специальной операцией, специфичной для реляционной базы данных.

Существует много подходов к определению реляционной алгебры, которые отличаются набором операции и способом их интерпретации, но, в принципе, более или менее равносильны. Расширенный начальный вариант алгебры, который был предложен Коддом, идентифицируется алгеброй Кодда.

В этом варианте множество основных алгебраических операций состоит из восьми операций, которые делятся на два класса - операция теоретического множественного числа и специальные реляционные операции. В состав теоретико-множественной операции входят операции:

- ассоциации отношений;
- пересечение отношений;
- принимая во внимание различия отношений;
- беря декартово построение отношений.

Специальные реляционные операции включают в себя:

- ограничительные отношения;
- проекционные отношения;
- присоединение к отношениям;
- деление отношений.

Кроме того, в состав алгебры включена операция присваивания, позволяющая сохранять в базе данных результаты расчетов алгебраических выражений, и операция переименования атрибута, позволяющая корректно формировать заголовок (схему) результирующих отношений.

При выполнении операций объединения (UNION) в двух отношениях с одинаковым заголовком вырабатывается отношение, включающее все кортежи, которые входят хотя бы в одно из отношений - операнд.

Операция пересечения (INTERSECT) двух отношений с одинаковым заголовком производит отношение, включая все кортежи, которые попадают в оба отношения-операнда.

Отношение, будучи разницей (MINUS) двух отношений с одинаковым заголовком, включает в себя все кортежи, попадающие в отношение - первый операнд, такой, что ни один из них не попадает в отношение, которое является вторым операндом.

При выполнении декартового произведения (ВРЕМЕНИ) два отношения, заголовок пересечения которых пусто, создается отношение, какие кортежи создаются ассоциациями первого и второго операндов кортежей.

Результатом отношений ограничения (WHERE) на некоторые условия являются отношения, в том числе кортежи отношений-операндов, удовлетворяющие этому условию.

При выполнении проекционных (PROJECT) отношений на заданном подмножестве ансамбля его атрибутом вырабатывается отношение, кортежи которого соответствуют подмножеству кортежей отношений-операндов.

При объединении (JOIN) двух отношений по некоторому условию формируется результирующее отношение, кортежи которого создаются ассоциациями кортежей первого и второго отношений и удовлетворяют этому условию.

Помимо операций реляционного деления (DIVIDE BY) два операнда - бинарные и монадические отношения. Результирующее отношение состоит из монадических кортежей, включая значения первого атрибута кортежей первого операнда, так что ансамбль значений второго атрибута (при фиксированной важности первого атрибута) включает в себя ансамбль значений второго операнда.

Операция переименования (RENAME) создает отношение, тело которого соответствует телу операнда, но атрибут имени изменяется.

Операция присваивания (: =) позволяет сохранить результат вычисления реляционного выражения в существующей базе данных отношений.

Кодд предложил использовать реляционную алгебру в СУБД для расчленения, заданного в ограниченных множествах. Он организовал свою системную базу данных вокруг концепции, основанной на множестве данных.

В реляционной модели данные разбиты на множества, которые образуют табличную структуру. Эта структура таблиц состоит из данных отдельных элементов, названных полями. Одиночный сет или группа по лоскуту, известному как запись.

Данные модели или концептуальное описание предметной области, - самый абстрактный уровень проектирования базы данных.

С точки зрения теории реляционных БД, основные принципы реляционной модели на концептуальном уровне можно сформулировать следующим образом:

- все данные вводятся в порядке ранжированной структуры, определяются в виде линий и столбцов и именуются по отношению;
- Все значения являются скалярами. Это означает, что для любой строки и столбца любых отношений существует одна и только одна важность;
- Все операции выполняются по целому числу по отношению, и результатом их выполнения также является целочисленное отношение. Этот принцип определяется закрытием.

Формулируя принципы реляционной модели, доктор Кодд выбрал термин «отношение» (отношение), поскольку, по его мнению, этот термин однозначен (в то время как, например, термин «таблица» имеет ансамбль другого типа - таблица в тексте)., таблицы и пр.). Более распространенная следующая ошибка: реляционная модель названа так, что она определяет отношения между таблицами. Действительно, название этой модели происходит от отношений (таблиц базы данных), лежащих в ее основе.

Каждая строка, содержащая данные, идентифицируется кортежем, каждый столбец отношения идентифицируется атрибутом (в темпе

практической работы с современными реляционными БД используются термины «запись» и «поле»).

Элементом описания реляционной модели, приведенным на концептуальном уровне, являются сущность, атрибуты, домены и отношения.

Сущность - некий отдельный объект или событие, информацию о котором необходимо сохранить в базе данных, определив заданную характеристику - атрибут. Сущность может быть как физическая (реально существующие объекты: например, СТУДЕНТ, атрибуты - число зачетной книги, фамилия, его факультет, профессия, номер группы и т. Д.), Так и абстрактная (например, EXAM, Атрибуты - дисциплина, дата, учитель, аудитория и пр.). Ее тип и копию различают по сущности. Тип характеризуется именем и списком свойств, а копия - конкретной характеристикой.

Атрибуты сущности быть:

Идентифицирующая и описательная. Атрибуты идентификации имеют уникальную важность для данного типа сущности и являются потенциальным ключом. Они позволяют однозначно распознать копии, по сути. Из возможных ключей выбирается один первичный ключ (ПК). В качестве ПК обычно выбирается потенциальная клавиша, на которой чаще всего встречается адрес для копирования записи. ПК должен включать в свой состав минимальные потребности для идентификации количества атрибута. Остальные атрибуты обозначены описательными.

Простые и составные. Атрибут Simple состоит из одного компонента, его важность неделима. Атрибут Компонент представляет собой комбинацию нескольких возможных компонентов, принадлежащих к различным заданным типам (например, адрес). Решение о том, использовать атрибут компонента или разделить его на компоненты, зависит от особенностей процессов его использования и может быть связано с обеспечением высокой скорости работы с большей базой данных.

Однозначный и неоднозначный - может иметь, соответственно, одно или много значений для каждой копии сущности.

Основное и производное. Важность основного атрибута не зависит от другого атрибута. Важность производного атрибута рассчитывается на основе значения другого атрибута (например, возраст человека рассчитывается на основе даты его рождения и текущей даты).

Спецификация атрибута состоит из его имен, инструкций типа данных и описаний ограничений - ансамбля значений (или домена), который может принимать данный атрибут.

Домен - набор всех возможных значений, которые могут содержать атрибут. Понятие «домен» часто путается с понятием «тип дан». Необходимо различать эти два понятия. Тип данных - физическая концепция, а предметная - логическая. Например, «целое число» - это тип данных, а «возраст» - домен.

Отношения - на концептуальном уровне представляют собой простые ассоциации между сущностями. Например, утверждение «Покупатели приобретают продукты» указывает, что между сущностью «Покупатели» и «Продукты» существует связь, и такая сущность идентифицируется участником этой связи.

Существует несколько типов отношений между двумя сущностями: это отношения «один к одному», «один ко многим» и «многие ко многим».
Каждое отношение в реляционной модели характеризуется именем, обязательностью, типом и степенью. Различают необязательные и обязательные отношения. Если сущность одного типа оказывается связанной с сущностью другого типа, то между этими типами объектов существует обязательная связь (отмечена двойная строка). В противном случае отношения не являются обязательными.

Отношение степени определяется количеством сущности, которые охватываются данным отношением. Пример бинарных отношений - это отношения между подразделением и сотрудником, в котором работает.

2.2 Проектирование базы данных

Один из важных этапов при разработке информационной системы это проектирование базы данных. Основные задачи проектирования базы данных:

- Обеспечение хранения в БД всей необходимой информации
- Обеспечение возможности приема дано по всем необходимым запросам
- Сокращение до избыточности и дублирования данных
- Предоставление базы данных целостности

Этапы проектирования базы данных:

- Концептуальное (инфологическое) проектирование
- Логическое (датологическое) проектирование
- Физическое проектирование
- Концептуальное (инфологическое) проектирование

Концептуальное (инфологическое) проектирование - построение семантической модели предметной области, то есть информационной модели высочайшего уровня абстракций. Такие модели без ориентации на некоторые конкретные СУБД и модельные данные. Термины «семантическая модель», «концептуальная модель» и «инфологическая модель» являются синонимами. Кроме того, в этом контексте может использоваться слово «база данных модели» и «модель предметной области» (например, «база данных концептуальной модели» и «концептуальная модель предметной области»), поскольку такая модель является изображением реальности. Поэтому и имидж разработан для этой реальности.

Конкретный тип и содержание базы данных концептуальной модели определены для этого формального устройства. Они обычно используют графические обозначения, такие как ER-диаграмма?

Чаще всего база концептуальных моделей включает в себя:

- Описание информационного объекта или понятия предметной области и взаимосвязи между ними.

- Описание ограничений, т. Е. Требований к возможным данным и взаимосвязи между ними.

Логическое (даталогическое) проектирование

Логическое (даталогическое) проектирование - создание базы данных схемы на основе данных конкретной модели, например данных реляционной модели. Для реляционной модели дана даталогическая модель - совокупность схем отношений, обычно с указанием первичных ключей, а также «отношений» между отношениями, представляющих собой внешние ключи.

Преобразование в концептуальную модель в логической модели, как правило, осуществляется по формальному правилу. Этот этап может быть в значительной степени автоматизирован.

На этапе логического проектирования учитывается специфика конкретных данных модели, но может не учитываться специфика конкретной СУБД.

Физическое проектирование

Физическое проектирование - создание схемы базы данных для конкретной СУБД. Специфика конкретной СУБД может включать в себя ограничения на именование объектной базы данных, ограничения на поддерживаемые типы данных и т. Д. Кроме того, специфика конкретной СУБД при физическом проектировании включает выбор решений в соответствии с физической средой хранения данных (выбор методов дисковой памяти). Управление, деление БД на файлы и устройства, способ доступа к данным, создание индекса и т. д.

Модели «сущность-отношения»

Модель "сущность-связь" (англ. "Entity-Relationshipmodel"), или ER-модель, предложенная П. Ченом в 1976 г., является наиболее известным представителем класса семантических (концептуальных, инфологических) моделей домен приложения. ER-модель обычно вводится в графической

форме, с использованием исходной записи П. Чена, называемой ER-диаграммой, или с использованием другой графической записи (Crow's Foot, Information Engineering и др.).

Основное преимущество ER-моделей:

- наглядность;
- модели позволяют проектировать базу данных с большим количеством объекта и атрибута;

ER-модели продаются во многих базах данных систем автоматизированного проектирования (например, ERWin) .

Основные элементы ER-моделей:

- объекты (сущность);
- объектные инструменты;
- отношения между объектом.

Сущность - это объект предметной области, имеющий атрибуты.

Отношения между сущностями характеризуются:

- тип отношения (1: 1, 1: N, N: M);
- Классные аксессуары. Класс может быть обязательным и ненужным. Если каждая копия сущности участвует во взаимоотношениях, то классовые принадлежности - обязательны, в противном случае - не нужны.

Семантические модели

Семантическая модель (концептуальная модель, инфологическая модель) - модель предметной области, предназначенная для представления семантики предметной области на самом высоком уровне абстракций. Это означает, что устраняется или сводится к минимуму необходимость использовать понятия «низкий уровень», связанные со спецификой физического представления и хранения данных.

Нормализация данных

Важный этап при проектировании базы данных - это нормализация данных, то есть приведение таблиц в нормальную форму. Нормальная форма - это характерная связь в данных реляционной модели, характеризующая его с точки зрения избыточности, потенциально может привести к логически неверному результату выборки или изменению данных. Нормальная форма определяется как совокупность требований, которые должны удовлетворять отношению.

Процесс преобразования базы данных отношений в тип, отвечающий нормальным формам, идентифицирован нормализацией. Нормализация предназначена для приведения структуры БД к типу, обеспечивая минимальную логическую избыточность, и не направлена на уменьшение или увеличение производительности работы или уменьшение или увеличение базы данных физического тома. Долгосрочной целью нормализации является сокращение потенциальной согласованности, которая сохраняется в базе данных. Общая цель процесса нормализации заключается в следующем:

- исключение некоторых типов для резервирования ;
- устранение некоторых аномальных ремонтов;
- разработка базы данных проекта, которая является достаточно «качественной» презентацией реального мира, интуитивно понятной и может послужить хорошей основой для последующего расширения;
- упрощение процедуры использования необходимых ограничений.

Удаление к избыточности производится, как правило, для подсчета разложений отношений так, чтобы в каждом отношении сохранялись только первичные факты (то есть факты, не вынутые из других предварительно установленных фактов).

Первая нормальная форма (1НФ)

Переменные отношения находятся в первой нормальной форме (1 НФ) тогда и только тогда, когда в любых возможных отношениях важности каждый его кортеж содержит только одну важность для каждого из атрибутов.

В реляционной модели отношение всегда находится в первой нормальной форме при определении понятия отношения. Что касается разных таблиц, то, что они могут быть не правильными представлениями отношений и, соответственно, могут быть не в 1 НФ.

Вторая нормальная форма (2NF)

Переменные отношения обнаруживаются во второй нормальной форме, если только она находится в первой нормальной форме и каждый не ключевой атрибут неприводимо (упакованная функция) зависит от ее потенциального ключа.

Третья нормальная форма (3NF)

Переменные отношения находятся в третьей нормальной форме, если она находится во второй нормальной форме, и отсутствуют транзитивно для функциональных зависимостей, а не ключевой атрибут из ключа.

При проектирование базы данных самым первым этапом является определить необходимые данные для базы данных. В этой выпуксной квалификационно работе на первой главе мы определяли необходимые данные. Из этих данных мы определяли несколько полей:

1. Название ресурса
2. Автор ресурса
3. Год печати
4. Количество страниц
5. Категория ресурса
6. Кафедра
7. Аннотация
8. Сфера образования
9. Ответственное лицо
10. Название файла

На рисунке 2.1 приведен представление модели базы данных.

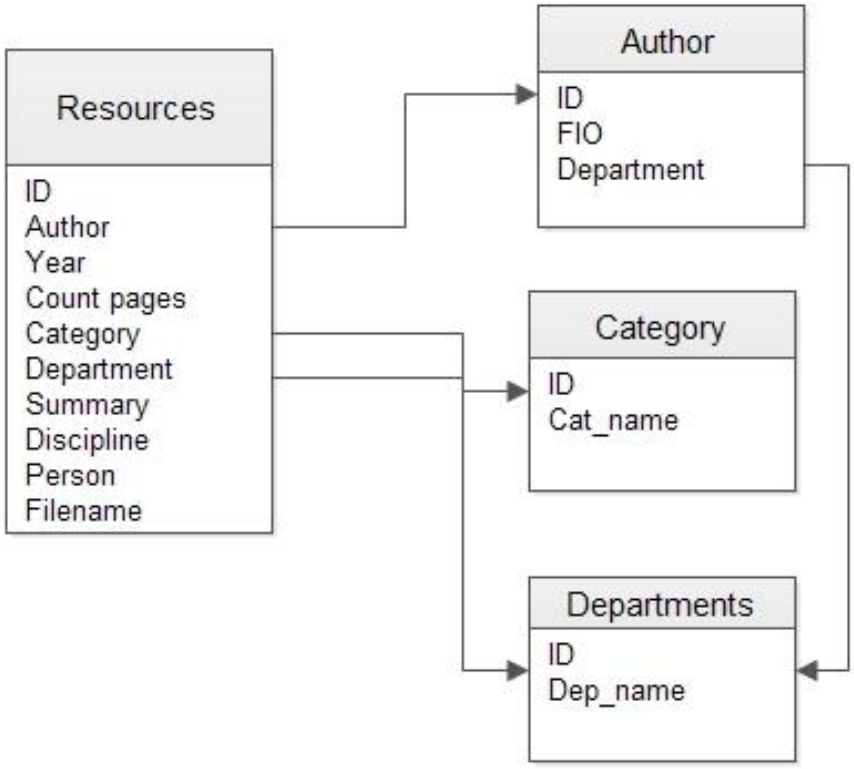


Рис 2.1 – Модель базы данных ZiyoNet

Здесь мы определили 4 таблицы:

- Resources – Это таблица для хранение данные всех ресурсов, это основное таблица
- Author – Таблица для хранение информации об авторах, их фамилия, имя, в каком кафедре работает и т.д.
- Category – Таблица для определении категорий ресурсов. На первой главе мы определили 10 категорий ресурсов.
- Departments – Это таблица для хранение название всех кафедр.

ГЛАВА III. СОЗДАНИЕ БАЗЫ ДАННЫХ

3.1 Выбор инструментов создание базы данных

Встроенная база данных SQLite

Автор SQLite - Д. Ричард Хипп. Изначально программа работала на Hewlett-Packard Unix (HPUX) и использовала базу данных Informix в качестве серверной части. Для их конкретного применения Informix был несколько излишним. Для опытного администратора базы данных (DBA) установка или обновление может занять почти целый день. Для непосвященного прикладного программиста это может занять вечность. Что действительно было необходимо, так это автономная база данных, которая была бы проста в использовании и могла путешествовать с программой и работать где угодно, независимо от того, какое другое программное обеспечение было или не было установлено в системе.

В январе 2000 года Хипп и его коллега обсудили идею создания простой встроенной базы данных SQL, которая использовала бы библиотеку B-Tree GNU DBM (gdbm) в качестве серверной части, которая не требовала бы никакой установки или административной поддержки. Позже, когда появилось свободное время, Хипп начал работу над проектом, и в августе 2000 года был выпущен SQLite 1.0. Как и планировалось, SQLite 1.0 использовал gdbm в качестве менеджера хранилища. Однако вскоре Хипп заменил его собственной реализацией B-дерева, которая поддерживала транзакции и сохраняла записи в ключевом порядке. С первым серьезным обновлением

SQLite начал устойчивую эволюцию, расширяясь как по функциям, так и по пользователям. К середине 2001 года многие проекты - как с открытым исходным кодом, так и коммерческие - начали использовать его. В последующие годы другие члены сообщества открытого исходного кода начали писать расширения SQLite для своих любимых языков сценариев и библиотек. Одно за другим, новые расширения - интерфейс Open Database Connectivity (ODBC), за которым следовали расширения для Perl, Python, Ruby, Java и других систем поддержки - стали на свои места и свидетельствовали о широком применении и полезности SQLite.

SQLite начал серьезное обновление с версии 2 до 3 в 2004 году. Его основной целью была расширенная интернационализация, поддерживающая текст UTF-8 и UTF-16, а также определяемые пользователем последовательности сортировки текста. Хотя 3.0 изначально планировалось выпустить летом 2005 года, America Online предоставила необходимое финансирование, чтобы убедиться, что оно было завершено к июлю 2004 года. Помимо интернационализации, версия 3 принесла много других новых функций, таких как обновленный C API, более компактный формат для файлов базы данных (уменьшение на 25 процентов), манифестная типизация, поддержка больших двоичных объектов (BLOB), 64-битные ROWID, автоочистка и улучшенный параллелизм. Несмотря на множество новых функций, общий объем библиотеки по-прежнему составлял менее 240 килобайт. Еще одно улучшение в 3 -й версии был хороший код очистки, и переписывание Пересмотр или иначе выбрасывая посторонние вещи, накопленную в 2 х серий.

SQLite продолжает развиваться по функциональным возможностям, оставаясь верным своим первоначальным целям разработки:

- Простота
- гибкость
- Компактность

- Скорость

Встроенная база данных

SQLite - это встроенная база данных. Вместо того, чтобы работать независимо как автономный процесс, он симбиотически сосуществует внутри приложения, которое он обслуживает, - внутри своего пространства процессов. Его код переплетается или внедряется как часть программы, в которой он находится. Для стороннего наблюдателя никогда не будет очевидно, что такая программа имеет встроенную систему управления реляционными базами данных (RDBMS). Программа просто выполняет свою работу и каким-то образом управляет своими данными, не обращая внимания на то, как это происходит. Но внутри, есть полный, автономный движок базы данных в работе.

Одним из преимуществ наличия сервера базы данных в вашей программе является то, что не требуется настройка сети или администрирование. И клиент, и сервер работают вместе в одном процессе. Это снижает накладные расходы, связанные с сетевыми вызовами, упрощает администрирование базы данных и упрощает развертывание приложения. Все, что вам нужно, скомпилировано прямо в вашу программу.

Рассмотрим процессы, показанные на рисунке 3.1. Один - это скрипт на Perl, другой - стандартная программа на C / C ++, а другой - процесс Apache с PHP, все с использованием SQLite. Сценарий Perl импортирует модуль DBI :: SQLite, который, в свою очередь, связан с API SQLite C, извлекая библиотеку SQLite. Библиотека PHP работает аналогично программе C ++. В конечном итоге все три процесса взаимодействуют с API-интерфейсом SQLite C. Поэтому все три имеют встроенный SQLite в свои пространства процессов, и все три являются независимыми серверами баз данных сами по себе. Более того, даже если каждый процесс представляет собой независимый сервер, он все равно может работать с одним и тем же файлом (файлами) базы данных, поскольку SQLite использует операционную систему для управления синхронизацией и блокировкой.

Для программистов SQLite похож на цифровую клейкую ленту, предоставляя простой способ привязки приложений и их данных. Как и клейкая лента, нет конца ее потенциальному использованию. В веб-среде SQLite может помочь в управлении сложной информацией о сеансе. Вместо того, чтобы сериализовать данные сеанса в один большой двоичный объект, отдельные фрагменты можно выборочно записывать и считывать из отдельных баз данных сеансов.

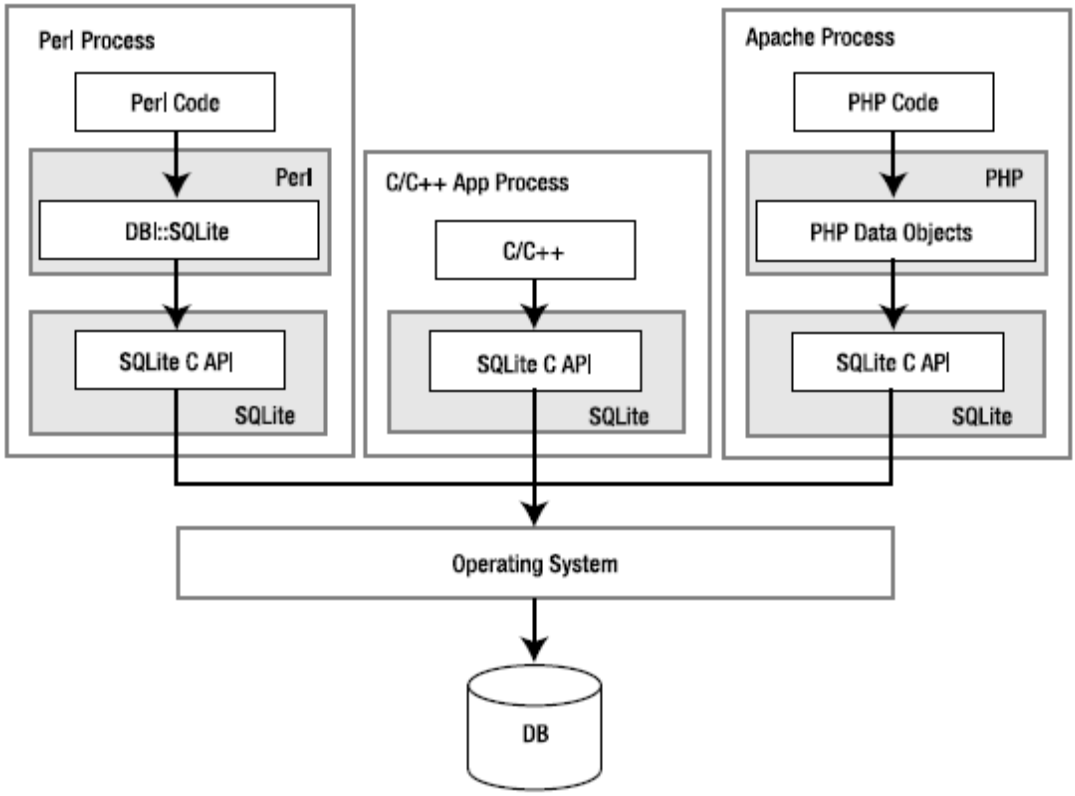


Рис. 3.1 – Процесс работы SQLite

SQLite также служит хорошей резервной реляционной базой данных для разработки и тестирования: нет внешних СУБД или сетей для настройки, а также имен пользователей и паролей. SQLite также может служить кешем, хранить конфигурационные данные или из-за своей двоичной совместимости между платформами даже работать как формат файла приложения.

Помимо того, что SQLite является просто хранилищем памяти, он может служить как чисто функциональным инструментом, так и для общей

обработки данных. В зависимости от размера и сложности, может быть проще представить некоторые структуры данных приложения в виде таблицы или таблиц в базе данных в памяти. Таким образом, вы можете работать с данными реляционно, используя SQLite для выполнения тяжелой работы, вместо того, чтобы написать свои собственные алгоритмы для манипулирования и сортировки структур данных.

Если вы уже знакомы с SQL, представьте себе кодирование эквивалента подзапроса, составного запроса, предложения GROUP BY или многоканального соединения - в C. SQLite встраивает все эти функции в ваше приложение с минимальными затратами. С ядром базы данных, интегрированным непосредственно в ваш код, вы можете начать думать о SQL как о предметно-ориентированном языке для реализации сложных алгоритмов сортировки в вашей программе. Этот подход становится более привлекательным по мере роста размера вашего набора данных или по мере усложнения ваших алгоритмов. Более того, SQLite можно настроить на использование фиксированного объема ОЗУ, а затем выгружать данные на диск, если он превышает указанный лимит. Это еще сложнее сделать, если вы пишете свои собственные алгоритмы. В SQLite это ограничение устанавливается одной командой SQL.

Но SQLite - это не просто база данных программиста. Это полезный инструмент для системных администраторов. Он небольшой, компактный и элегантный, как регулярное выражение или утилита Unix, такая как find, rsync или grep. SQLite имеет утилиту командной строки, которую можно использовать в сценариях оболочки. Тем не менее, он работает еще лучше с большим разнообразием языков сценариев, таких как Perl, Python и Ruby. Вместе они могут помочь в решении самых разнообразных задач, таких как объединение данных файла журнала, мониторинг дисковых квот или учет пропускной способности в межсетевых экранах с отслеживанием состояния. Кроме того, поскольку базы данных SQLite представляют собой обычные файлы операционной системы, с ними легко работать, переносить и создавать

резервные копии. Кроме того, SQLite является удобным инструментом обучения. Это идеальная база данных для начинающих, с которой можно узнать о реляционных концепциях. Он может быть быстро и легко установлен практически на любой платформе, и его файлы базы данных свободно обмениваются между собой без необходимости преобразования. Это полнофункциональный, но не пугающий. И это - и программа, и база данных - можно носить с собой на дискете или на универсальной последовательной шине (USB).

В отличие от большинства продуктов RDBMS, SQLite не имеет клиент-серверной архитектуры. В большинстве крупных систем баз данных имеется большой серверный пакет, который составляет базу данных. Сервер базы данных часто состоит из нескольких процессов, которые работают совместно для управления клиентскими подключениями, файловым вводом-выводом, кэшированием, оптимизацией запросов и обработкой запросов. Экземпляр базы данных обычно состоит из большого количества файлов, организованных в одно или несколько деревьев каталогов на сервере. Для доступа к базе данных все файлы должны присутствовать и быть правильными. Это может затруднить перемещение или надежное резервное копирование экземпляра базы данных.

Все эти компоненты требуют ресурсов и поддержки со стороны главного компьютера. Bestpractices также предписывают, чтобы хост-система была настроена с выделенными учетными записями пользователей, сценариями запуска и выделенным хранилищем, что делает сервер баз данных очень интрузивным программным обеспечением. По этой причине и из-за проблем с производительностью принято выделять компьютерную систему исключительно для программного обеспечения сервера базы данных.

Для доступа к базе данных клиентские библиотеки программного обеспечения обычно предоставляются поставщиком базы данных. Эти библиотеки должны быть интегрированы в любое клиентское приложение, которое хочет получить доступ к серверу базы данных. Эти клиентские

библиотеки предоставляют API для поиска и подключения к серверу базы данных, а также для настройки и выполнения запросов и команд базы данных. На рис. 3-2 показано, как все сочетается в типичной клиент-серверной СУБД.

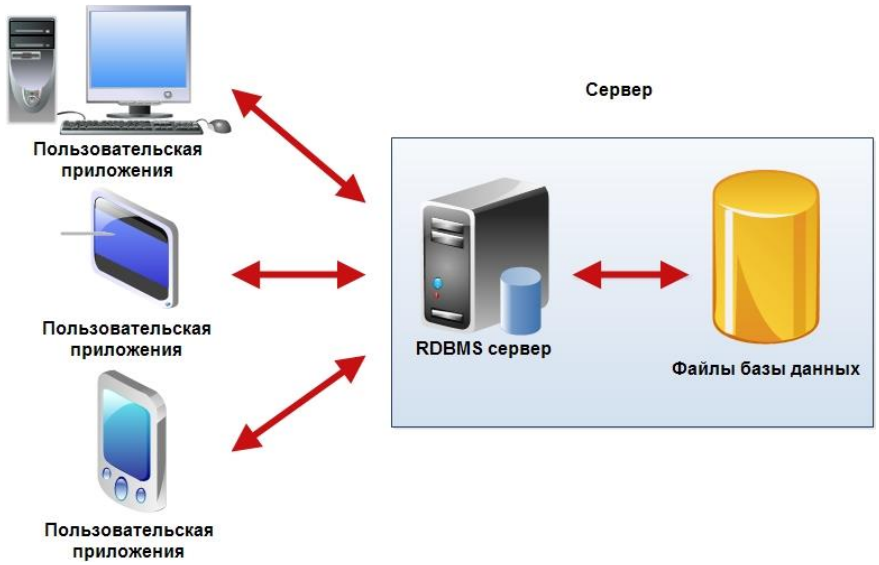


Рис. 3.2 – Традиционная клиент-серверная архитектура СУБД.

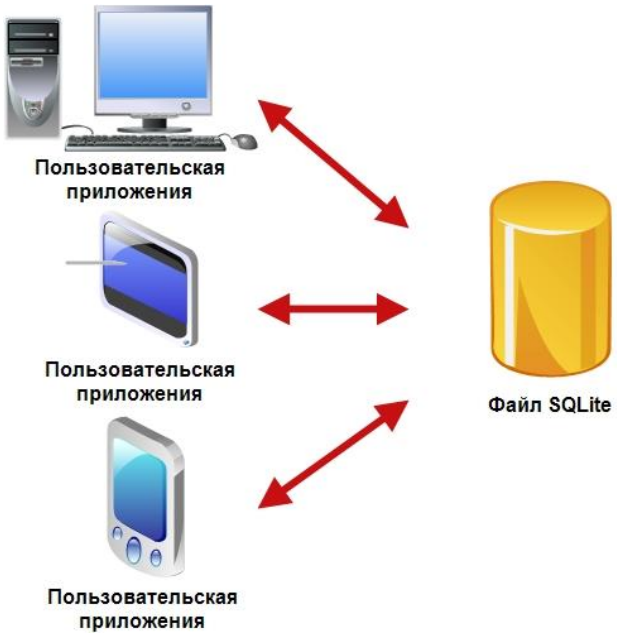


Рис. 3.3 – Архитектура SQLite без сервера.

В отличие от SQLite нет отдельного сервера. Весь движок базы данных интегрирован в любое приложение, необходимое для доступа к базе данных. Единственный общий ресурс среди приложений - это единственный файл базы

данных, который находится на диске. Если вам нужно переместить или создать резервную копию базы данных, вы можете просто скопировать файл. Рисунок 3-3 показывает инфраструктуру SQLite.

Особенности SQLite

SQLite предлагает удивительный набор функций и возможностей, несмотря на его небольшой размер. Он поддерживает большое подмножество ANSI SQL92 (транзакции, представления, проверочные ограничения, коррелированные подзапросы и составные запросы), а также многие другие функции, имеющиеся в реляционных базах данных, такие как триггеры, индексы, столбцы автоинкремента и предложение LIMIT / OFFSET. У него также есть много уникальных особенностей, таких как базы данных в памяти, динамическая типизация и то, что называется разрешением конфликтов (объяснено ниже).

Нулевая конфигурация

Исходя из своей первоначальной концепции, SQLite был разработан с учетом особого отсутствия администратора баз данных. Настройка и администрирование SQLite настолько просты, насколько это возможно. SQLite содержит достаточно функций, чтобы поместиться в мозгу одного программиста, и, как и его библиотека, требует такой же небольшой площади в сером веществе, как и в оперативной памяти.

Портативность

SQLite был разработан специально для переносимости. Он компилируется и работает в Windows, Linux, BSD, Mac OS X, коммерческих системах Unix, таких как Solaris, HPUX и AIX, а также на многих встроенных платформах, таких как QNX, VxWorks, Symbian, Palm OS и Windows CE. Он без проблем работает на 16-, 32- и 64-разрядных архитектурах с порядком байтов, как в старшем, так и в младшем порядке. Портативность не ограничивается и программным обеспечением: файлы базы данных SQLite так же переносимы, как и его код. Формат файла базы данных является двоично-

совместимым для всех поддерживаемых операционных систем, аппаратных архитектур и порядков байтов.

Компактность

SQLite был спроектирован так, чтобы быть легковесным и автономным: один заголовочный файл, одна библиотека, и вы - реляционный, внешний сервер базы данных не требуется. Все - клиент, сервер, виртуальная машина - упаковывается в аккуратный четверть мегабайта, который на данный момент меньше домашней страницы издателей этой книги: www.apress.com (домашняя страница весит около 260 килобайт). Кроме того, существует проприетарная версия SQLite, которая имеет небольшой размер - 69 килобайт и может работать на смарт-картах (более подробную информацию см. В разделе «Дополнительная информация»). Столь же компактны базы данных SQLite. Это обычные файлы операционной системы. Независимо от вашей системы все объекты в вашей базе данных SQLite - таблицы, триггеры, схемы, индексы и представления - содержатся в одном файле операционной системы. Кроме того, SQLite использует записи переменной длины, выделение только минимального объема данных, необходимых для хранения каждого поля. 2-байтовое поле в столбце varchar (100) занимает всего 3 байта, а не 100 (дополнительный байт используется для записи информации об его типе).

Простота

Как библиотека программирования, API-интерфейс SQLite является одним из самых простых и простых в использовании. API хорошо документирован и интуитивно понятен. Он предназначен для того, чтобы помочь вам настроить SQLite разными способами, например, реализовать свои собственные пользовательские функции SQL в C. Более того, сообщество open source создало огромное количество языковых и библиотечных интерфейсов для использования SQLite. Существуют расширения для Perl, Python, Ruby, Tcl / Tk, Java, PHP, Visual Basic, ODBC, Delphi, Microsoft .NET, Smalltalk, Ada, Objective C, Eiffel, Rexx, Lisp, Scheme, Lua, Pike, Objective Camel, Qt, WxWindows, REALBASIC и другие. SQLite имеет модульную конструкцию.

Этот дизайн включает в себя множество инновационных идей, которые позволяют ему быть полнофункциональным и расширяемым, в то же время сохраняя высокую степень простоты всей своей кодовой базы. Каждый модуль является специализированным, независимая система, которая выполняет определенную задачу. Эта модульность значительно упрощает разработку каждой системы и отладку запросов при их передаче из одной системы модуля к следующему - от компиляции и планирования до исполнения и материализации. Несколько факторов работают вместе, чтобы сделать SQLite очень гибкой базой данных. Как встроенная база данных, она предлагает лучшее из обоих миров: мощь и гибкость внешнего интерфейса реляционной базы данных, а также простоту и компактность внутреннего интерфейса B-дерева. Благодаря этому не нужно настраивать большие серверы баз данных, не нужно беспокоиться о проблемах с сетью или подключением, нет ограничений на платформу, не нужно изучать API в стиле барокко, а также не нужно платить лицензионные сборы или роялти.

Надежность

Но исходный код больше, чем просто бесплатный. Он также написан хорошо. Базовый код SQLite содержит около 30 000 строк стандартного ANSI C, который является чистым, модульным и хорошо прокомментированным. Он разработан так, чтобы быть доступным, простым для понимания, легко настраиваемым и в целом очень доступным. Компетентный программист C легко может следовать любой части SQLite или всему этому с достаточным временем и обучением. Кроме того, код SQLite предлагает полнофункциональный API, специально предназначенный для настройки расширяемого SQLite посредством добавления пользовательских функций, агрегирование и сопоставление последовательностей наряду с поддержкой эксплуатационной безопасности. Хотя модульная конструкция SQLite значительно повышает его общую надежность, ее исходный код также хорошо протестирован.В то время как основное программное обеспечение (библиотека и утилиты) состоит из около 30 000 строк кода, в дистрибутив

также входит обширный набор тестов, состоящий из более чем 30 000 строк кода регрессионного теста, который охватывает более 97 процентов кода ядра. Таким образом, более половины всего кода проекта SQLite посвящено исключительно регрессионному тестированию. Иными словами, это касается каждой строки написанного кода базы данных, также написана примерно одна строка кода тестирования. Также написана примерно одна строка кода для тестирования. Также написана примерно одна строка кода для тестирования.

3.2. Реализация и тестирование базы данных

На первой главе мы изучили необходимые данные для базы данных. На второй главе мы проектировали базы данных из этих информации. Следующий этап это создание базы данных. Для базы данных мы выбрали компактная СУБД SQLite. Потому что его можно использовать на различных платформах с легкостью.

Можно создать базу данных с помощью консоли или с помощью менеджеров. Такие как Db Browser for SQLite, SQLite Studio и т.д. Из этиж менеджеров мы выбрали SQLite Studio. Потому что его удобно использовать и с помощью менеджера можно быстро создать базы данные с различными сложностями. На рисунке 3.5 приведен интерфейс SQLite Studio.

По плану мы должны создать 4 таблицы:

1. Resources
2. Author
3. Category
4. Departments

Первая таблица у нас это Resources, в нем храняться основные информации о ресурсах. В таблице 3.1 приведен поля и их типы.

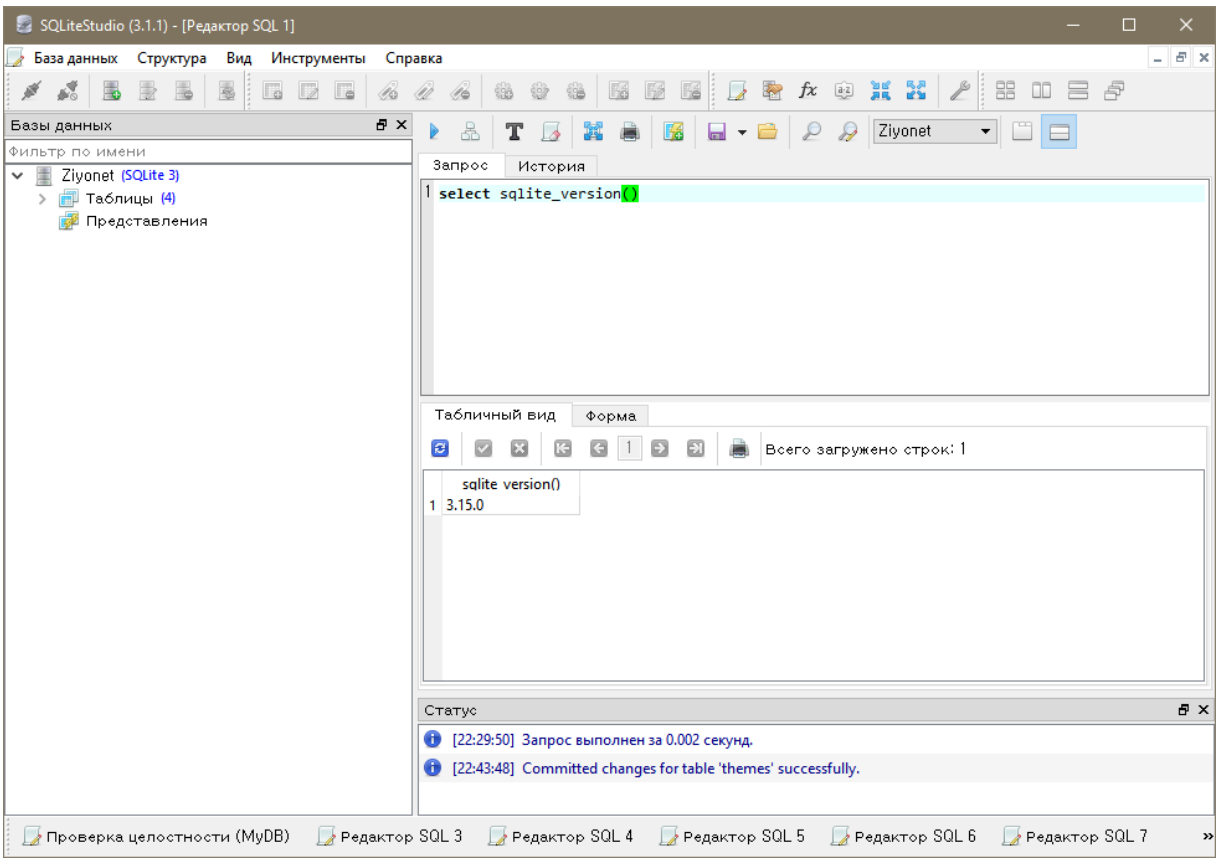


Рис. 3.4 – Интерфейс SQLite Studio

Таблица 3.1

№	Название поля	Тип	Описание
1.	ID	INTEGER	Уникальный ID номер для каждого ресурса
2.	Name_res	VARCHAR	Название ресурса
3.	Author	INTEGER	ID номер автора из таблицы Author ФИО Автора
4.	Year	INTEGER	Год
5.	Count_pages	INTEGER	Количество страниц материала
6.	Category	INTEGER	ID номер категорий из таблицы Category Назваие категорий

7.	Department	INTEGER	ID номер кафедры из таблицы Departments Название кафедры
8.	Summary	TEXT	Краткая аннотация материала
9.	Discipline	VARCHAR	Сфера образования, название предмета которое относиться этот материал
10.	Person	VARCHAR	Ответственное лицо
11.	Filename	VARCHAR	Название файла

Для создание таблицы можно использовать графический интерфейс программы SQLite Studio (рис 3.5) или же sql запросы.

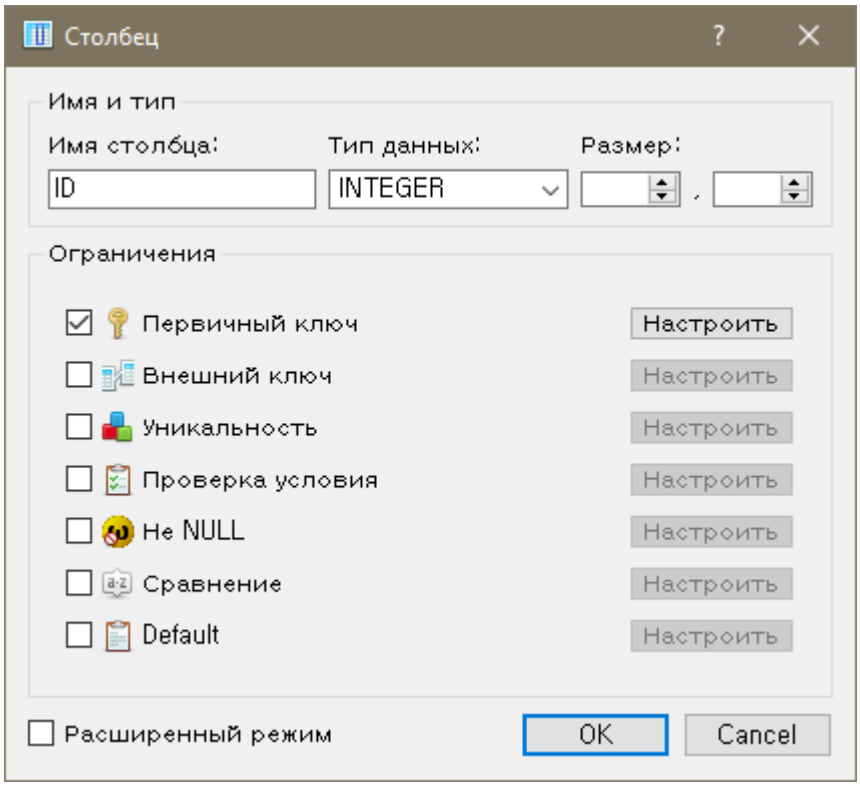


Рис. 3.5 – Создание таблицы с помощью графического интерфейса

Для создание таблицы resources мы использовали следующий sql запрос:

```
CREATE TABLE resources (
ID INTEGER PRIMARY KEY,
```

```
author INTEGER,
year INTEGER,
count_pages INTEGER,
category INTEGER,
department INTEGER,
summary TEXT,
discipline VARCHAR,
person VARCHAR,
filename VARCHAR
);
```

Sql запрос создание таблицы author:

```
CREATE TABLE author (
ID INTEGER PRIMARY KEY,
FIO VARCHAR,
department INTEGER
);
```

На таблице 3.2 приведен данные полей таблицы author.

Таблица 3.2 Поля таблицы author.

№	Название поля	Тип	Описание
1.	ID	INTEGER	Уникальный ID номер для каждого автора
2.	FIO	VARCHAR	ФИО Автора
3.	department	INTEGER	Кафедра автора

Sql запрос создание таблицы category:

```
CREATE TABLE category (
ID INTEGER PRIMARY KEY,
cat_name VARCHAR
);
```

На таблице 3.3 приведен данные полей таблицы category.

Таблица 3.3 Поля таблицы category.

№	Название поля	Тип	Описание
1.	ID	INTEGER	Уникальный ID номер для каждой категорий
2.	cat_name	VARCHAR	Название категорий

Sql запрос создание таблицы departments:

```
CREATE TABLE departments (
ID INTEGER PRIMARY KEY,
dep_name VARCHAR
);
```

На таблице 3.4 приведен данные полей таблицы departments.

Таблица 3.4. Поля таблицы departments.

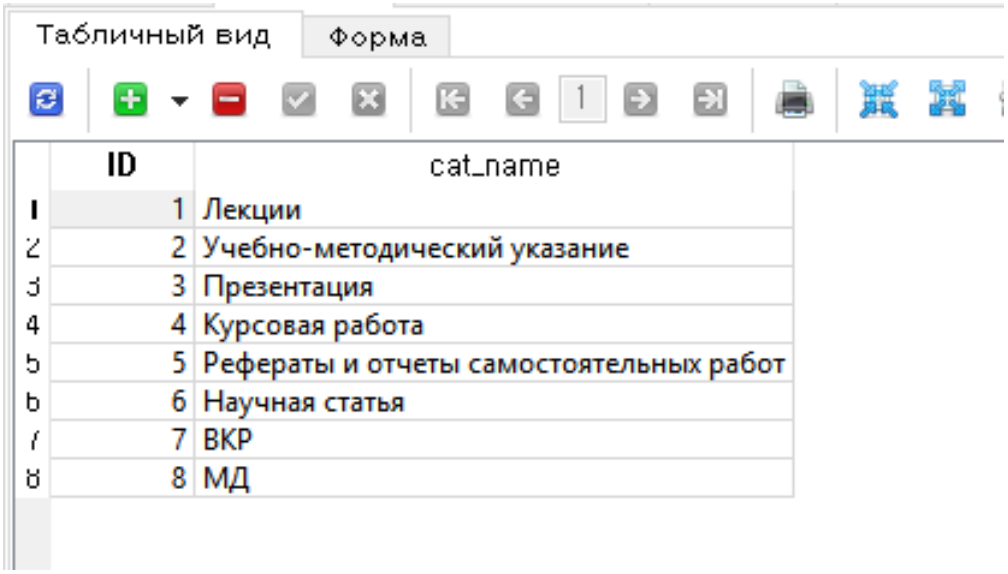
№	Название поля	Тип	Описание
1.	ID	INTEGER	Уникальный ID номер для каждой кафедры
2.	dep_name	VARCHAR	Название кафедры

После создание базы всех таблиц нам надо осуществить связи между таблицами. Все связи приведены в таблице 3.5.

Таблица 3.4 – свзязи между таблицами.

№	Название таблицы	Название поля	Связь	Название таблицы	Название поля
1.	resources	author	-	author	ID
2.	resources	category	-	category	ID
3.	resources	department	-	departments	ID
4.	author	department	-	departments	ID

Что бы работать с базы данных мы добавили в таблицы начальные данные. Для таблицы category мы добавили следующие начальные данные (Рис. 3.6).



	ID	cat_name
1	1	Лекции
2	2	Учебно-методический указание
3	3	Презентация
4	4	Курсовая работа
5	5	Рефераты и отчеты самостоятельных работ
6	6	Научная статья
7	7	ВКР
8	8	МД

Рис. 3.6 – Данные таблицы category

SQL запрос добавление данных в таблицу category:

```
INSERT INTO category('cat_name') VALUES ('Лекции')
INSERT INTO category('cat_name') VALUES ('Учебно-
методический указание')
INSERT INTO category('cat_name') VALUES ('Презентация')
INSERT INTO category('cat_name') VALUES ('Курсовая работа')
INSERT INTO category('cat_name') VALUES ('Рефераты и отчеты
самостоятельных работ')
INSERT INTO category('cat_name') VALUES ('Научная статья')
INSERT INTO category('cat_name') VALUES ('ВКР')
INSERT INTO category('cat_name') VALUES ('МД')
```

Для таблицы departments мы добавили следующие начальные данные (Рис. 3.7).

SQL запрос добавление данных в таблицу category:

```
INSERT INTO departments('dep_name') VALUES ('Информационные
технологий')
```

```
INSERT INTO departments('dep_name') VALUES ('Программный
инжиниринг')
INSERT INTO departments('dep_name') VALUES
('Телекоммуникационные технолгий')
```

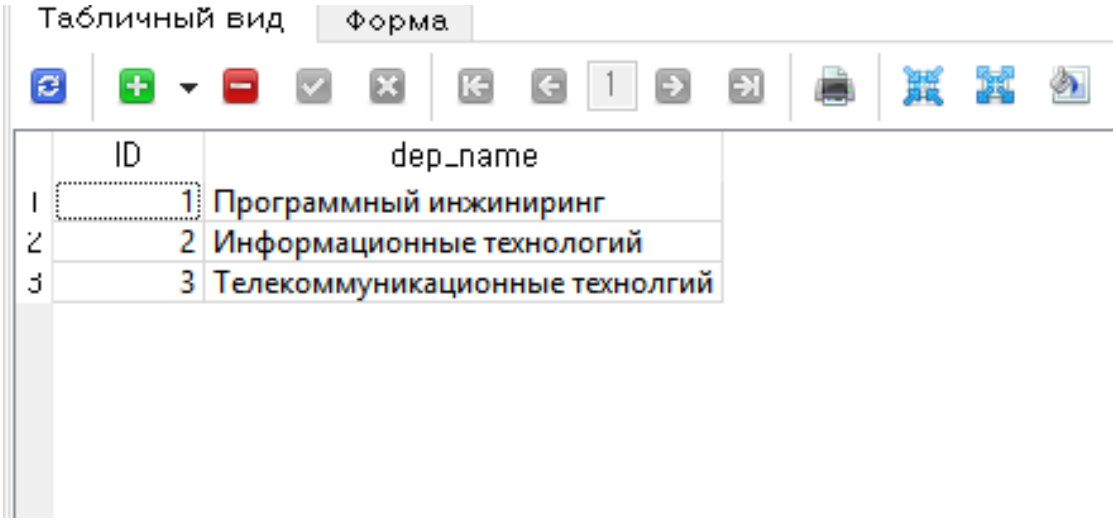


Рис. 3.7 – Данные таблицы departments

Добавление данные в таблицы author отличается от двух предыдущих. Потому что в этой талице есть связь с таблицы departments. По этому надо установить внешний ключ. Для этого мы использовали интерфейс менеджера SQLite Studio (Рис. 3.8 и 3.9).

Ниже приведен SQL запрос установление внешнего ключа для таблицы author с departments.

```
PRAGMA foreign_keys = 0;
CREATE TABLE sqlitestudio_temp_table AS SELECT *
FROM author;
DROP TABLE author;

CREATE TABLE author (
ID INTEGER PRIMARY KEY,
FIO VARCHAR,
department INTEGER REFERENCES departments (ID)
);
INSERT INTO author (
ID,
```

```
FIO,
department
)
SELECT ID,
FIO,
department
FROM sqlitestudio_temp_table;
DROP TABLE sqlitestudio_temp_table;
PRAGMA foreign_keys = 1;
```

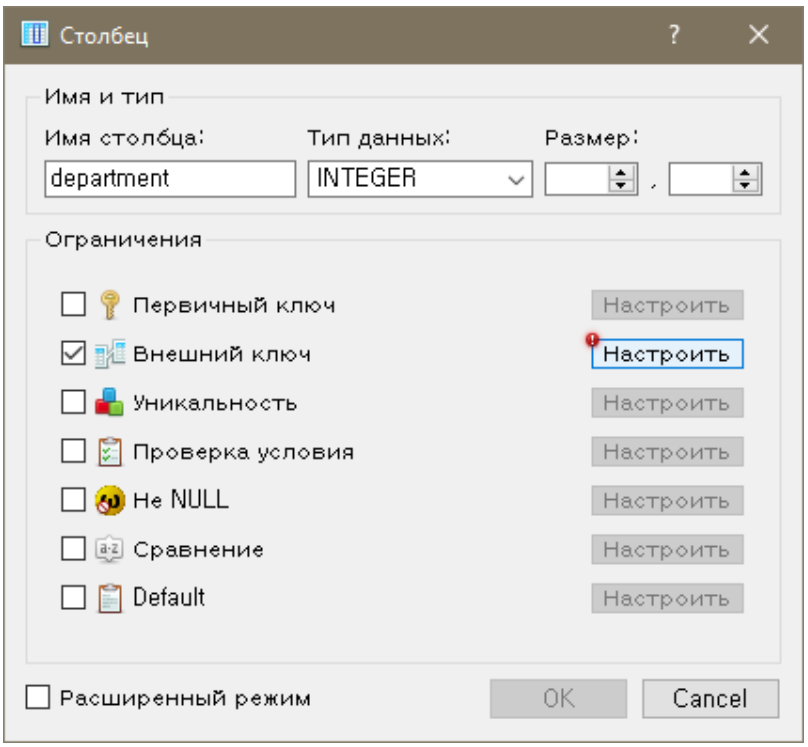


Рис. 3.8 – Установление внешнего ключа

Для добавление автором в таблицу author мы использовали графический интерфейс менеджера(Рис 3.10), потому что графический интерфейс является удобным средством.

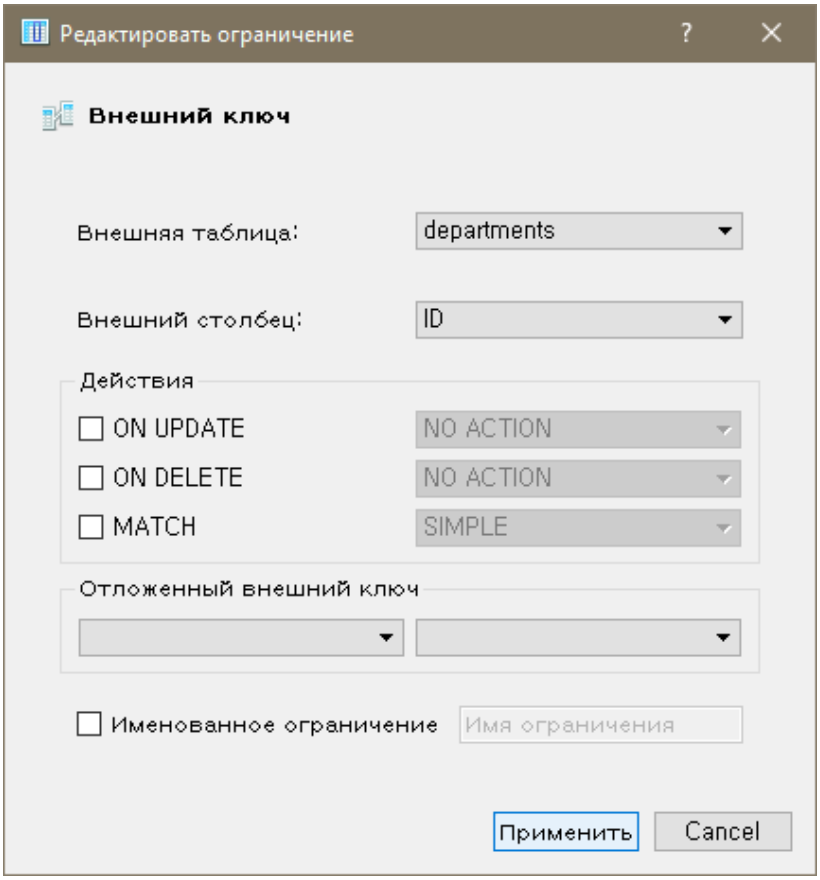


Рис. 3.9 - Установление внешнего ключа

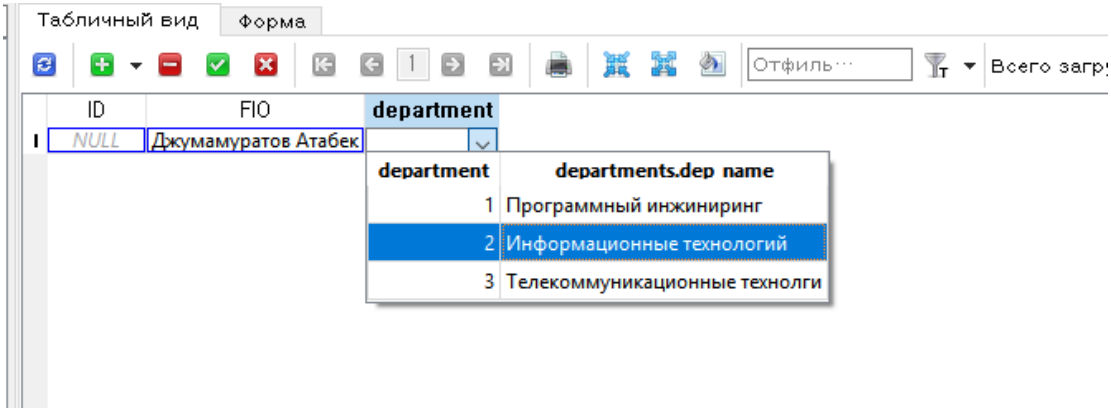
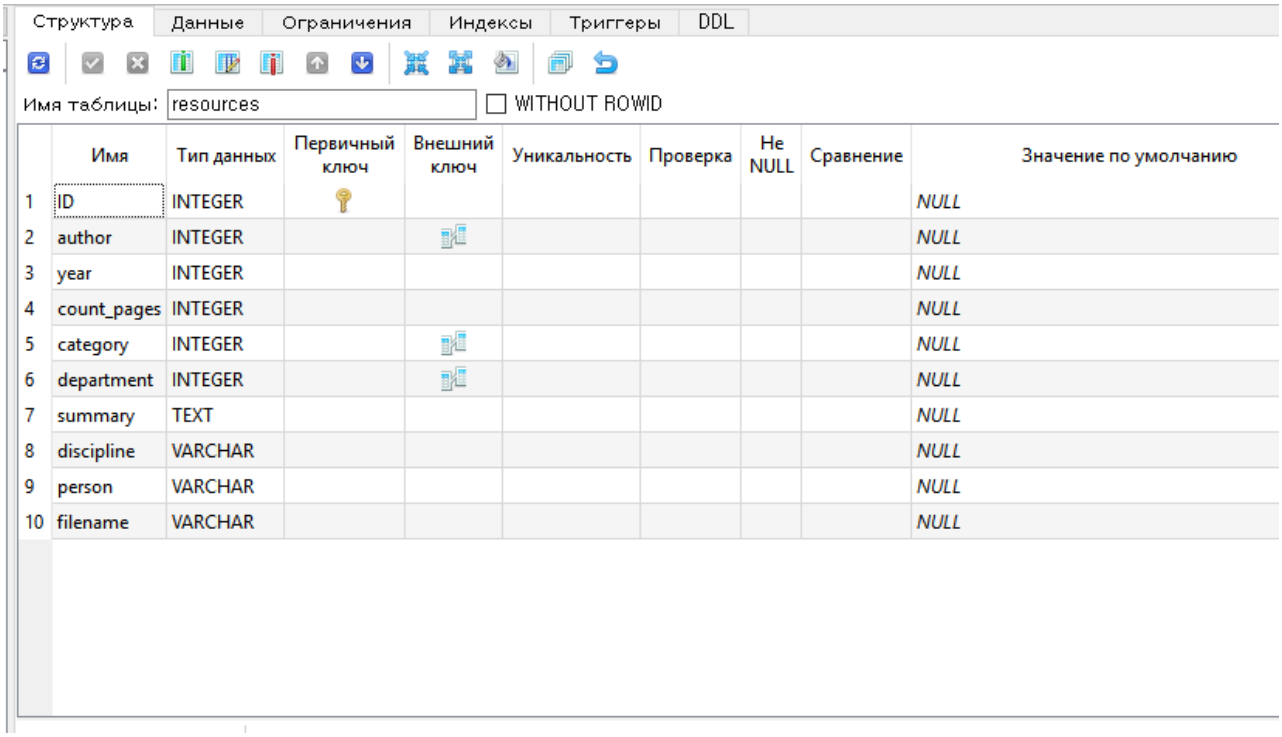


Рис. 3.10 – Добавление автора в таблицу author

Как и таблица author мы установили связи в таблице resources. (Рис. 3.11). Значок в поля «Внешний ключ» означает что у это поля есть связь с другой таблицей.



	Имя	Тип данных	Первичный ключ	Внешний ключ	Уникальность	Проверка	Не NULL	Сравнение	Значение по умолчанию
1	ID	INTEGER							NULL
2	author	INTEGER							NULL
3	year	INTEGER							NULL
4	count_pages	INTEGER							NULL
5	category	INTEGER							NULL
6	department	INTEGER							NULL
7	summary	TEXT							NULL
8	discipline	VARCHAR							NULL
9	person	VARCHAR							NULL
10	filename	VARCHAR							NULL

Рис. 3.11 – Результат установление связи в таблице resources.

Ниже приведен sql запрос установление связи между таблицами.

```
PRAGMA foreign_keys = 0;
CREATE TABLE sqlitestudio_temp_table AS SELECT *
FROM resources;
DROP TABLE resources;
CREATE TABLE resources (
ID INTEGER PRIMARY KEY,
author INTEGER REFERENCES author (ID),
year INTEGER,
count_pages INTEGER,
category INTEGER REFERENCES category (ID),
department INTEGER REFERENCES departments (ID),
summary TEXT,
discipline VARCHAR,
person VARCHAR,
filename VARCHAR
);
INSERT INTO resources (
ID,
```

```
author,
year,
count_pages,
category,
department,
summary,
discipline,
person,
filename
)
SELECT ID,
author,
year,
count_pages,
category,
department,
summary,
discipline,
person,
filename
FROM sqlitestudio_temp_table;
DROP TABLE sqlitestudio_temp_table;
PRAGMA foreign_keys = 1;
```

После установление необходимые связи и пополнение начальными данными таблиц, нам необходимо тестировать базы данных. Для этого мы добавим несколько ресурсов в базу данных.

Например, чтобы добавить автора сначала надо занести его в базу. То есть надо добавить авторов в таблица author. После это нам надо выбрать автра при добавление ресурса в базу (Рис.3.12).

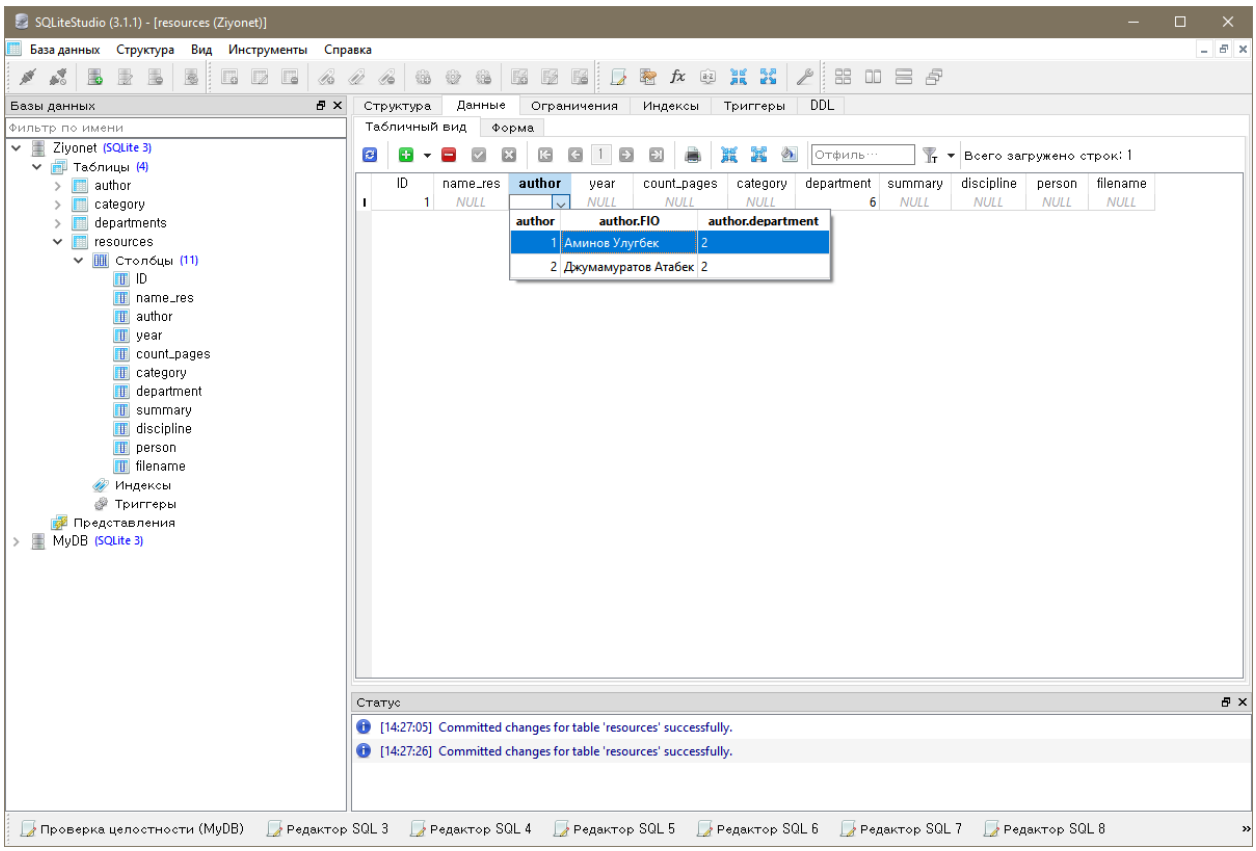


Рис.3.12 – Процесс добавление ресурса в базу.

Как и авторов надо сначала добавить всех категорий и название кафедры. После мы можем выбрать этот же записей (Рис 3.13 и 3.14). После этого мы добавили в базу несколько записей.

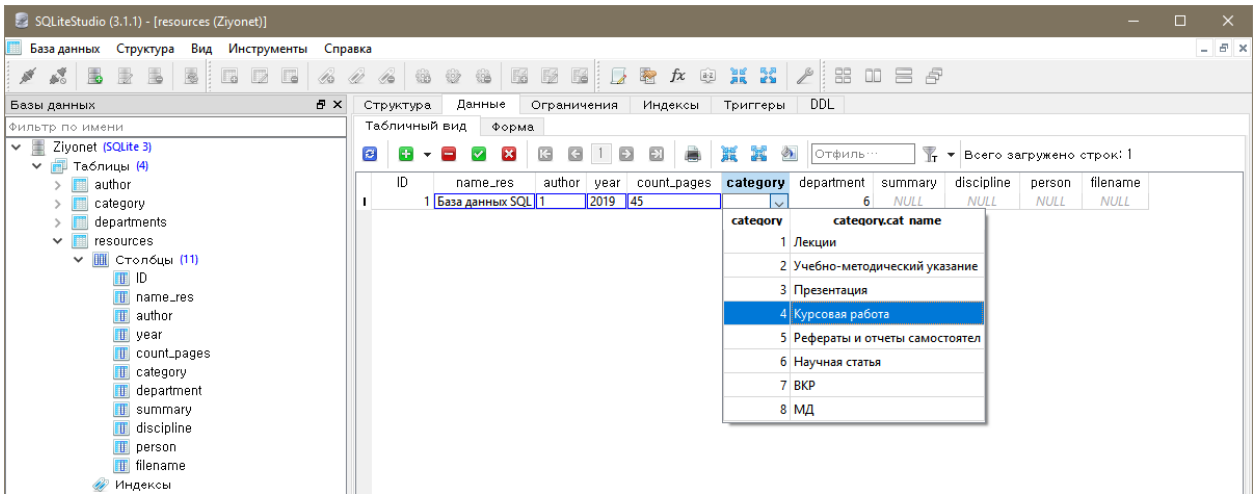


Рис. 3.13 – Выбор категорий

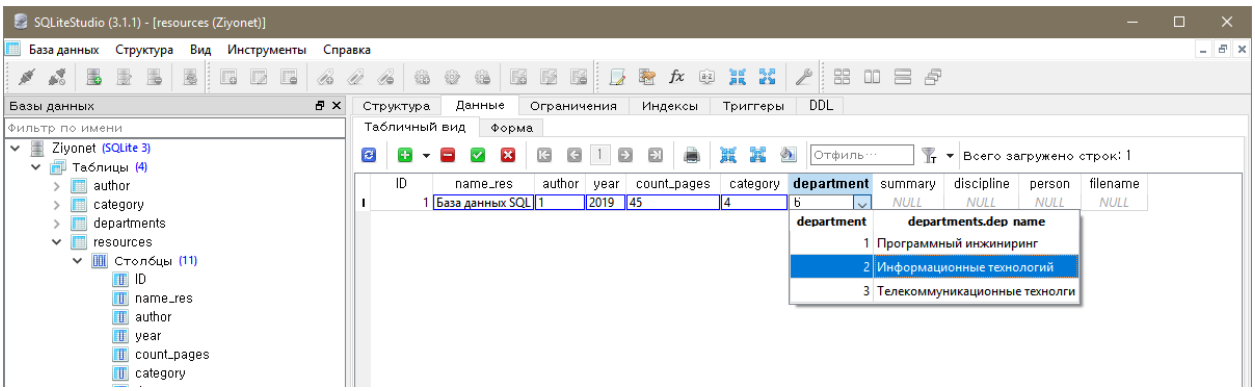


Рис. 3.14 – Выбор кафедры

3.3. Создание отчетов

После добавление в базу несколько записей мы можем создать разные отчеты. Например для содание отчета ресурсов по кафедре мы можем использовать следующий sql запрос:

```
SELECT resources.name_res AS Название, departments.dep_name AS Кафедра FROM resources, departments WHERE resources.department = departments.ID
```

С помощью этого запроса мы можем извлечь из базы название ресурсов и название кафедры. Результат запроса приведен ниже:

Название	**Кафедра**
База данных SQL	Программный инжиниринг
Язык программирование Java	Информационные технологий

Если нам потребуется искать в базе ресурсы конкретный кафедры, тогда можно использовать следующий sql запрос:

Отчёт №1. Список ресурсов по искомой кафедре

```
SELECT resources.name_res AS Название, departments.dep_name AS Кафедра FROM resources, departments WHERE resources.department = departments.ID AND departments.dep_name = "Информационные технологий"
```

В пункте `departments.dep_name = "Информационные технологий"` можно написать название кафедры для поиска. Результат запроса приведен ниже:

Название	**Кафедра**
Язык программирование Java	Информационные технологий

Например нам потребуеться узнать количество ресурсов по каждый кафедре. Тогда с помощью команды GROUP BY и COUNT можно получить количество ресурсов по кафедре.

Отчёт №2. Количество ресурсов по каждой кафедре

```
SELECT departments.dep_name AS Кафедра, COUNT(*) AS Количество FROM resources, departments WHERE resources.department = departments.ID GROUP BY resources.department
```

Результат:

Кафедра	**Количество**
Программный инжиниринг	1
Информационные технологий	1

Теперь можно создать несколько отчетов от предудущих пример. Например, узнать количество ресурсов по типам.

Отчёт №3. Количество ресурсов по типам ресурсов.

```
SELECT category.cat_name AS Категория, COUNT(*) AS Количесвто FROM resources, category WHERE resources.category= category.ID GROUP BY resources.category
```

Результат:

Категория	**Количество**
Курсовая работа	2

Возможно потребуется список ресурсов сортированной по годам. Тогда можно использовать команду ORDER BY. Это команда сортирует записей по выбранному полю.

Отчёт №4. Список ресурсов по годам

```
SELECT name_res AS Название, author.FIO AS Автор, year AS Год FROM resources, author WHERE resources.author = author.ID ORDER BY year
```

Результат:

Название	**Автор**	**Год**
Язык программирование Java	Джумамуратов Атабек	2018
База данных SQL	Аминов Улугбек	2019

Если потребуется искать ресурсов конкретного автора, можно использовать следующий sql запрос.

Отчёт №5. Поиск ресурсов конкретного автора

```
SELECT name_res AS Название, author.FIO AS Автор FROM resources, author WHERE resources.author = author.ID AND author.FIO = "Аминов Улугбек"
```

Результат:

Название	**Автор**
База данных SQL	Аминов Улугбек

Ниже приведен несколько видом отчета.

Отчёт №6. Поиск ресурсов по конкретной категорий.

```
SELECT resources.name_res AS Название, category.cat_name AS Категория FROM resources, category WHERE resources.category= category.ID AND category.cat_name = "Курсовая работа"
```

Результат:

Название	**Категория**
База данных SQL	Курсовая работа
Язык программирование Java	Курсовая работа

Отчёт №7. Поиск ресурсов по годам.

```
SELECT resources.name_res AS Название, year AS Год FROM resources WHERE year=2019
```

Результат:

Название	**Год**
База данных SQL	2019

Потребуется список ресурсов по фразы. Например надо извлечь из базы ресурсов относящие на язык java. Тогда можно искать эту фразу на поле краткая описание. Для это используется команда LIKE.

Отчёт №8. Поиск ресурсов по фразе.

```
SELECT resources.name_res AS Название, author.FIO AS Автор, resources.summary AS Описание FROM resources, author WHERE author.id = resources.author AND resources.summary LIKE "%java%"
```

Результат:

Название	**Автор**	**Описание**
Язык программирование Java	Джумамуратов Атабек	Курсовая работа по язык программирование java

ЗАКЛЮЧЕНИЕ

Целью данной выпускной квалификационной работы заключалась в проектировании и созданий базы данных для ведение работы с учебно-образовательными ресурсами портала ZiyoNet в ВУЗе.

В ходе выполнения работы автором были решены следующие задачи:

- Изучение сферы деятельности;
- Изучение положение ведение работы с ZiyoNet;
- Изучение необходимые данные ресурсов;
- Проектирование база данных;
- Выбрано СУБД SQLite и программное обеспечение SQLite Studio для создание базы данных;
- Создание база данных;
- Создание несколько отчетов;
- Тестировалась база данных.

На стадии обследования существенную помощь оказал руководитель по выпускной квалификационной работе.

Разработанная база данных продемонстрирована в кафедре. База данных имеет большой потенциал дальнейшего развития. Можно использовать созданной базы данных в работе с порталом ZiyoNet в ВУЗе. Связи с тем, что база данных создано в компактной SQLite, можно реализовать интерфейсы для работы в различных платформах, такие как android, desktop или серверное приложениях.

Результаты использование база данных показали, что работа с такими объемами данных значительно увеличивает эффективность работы и сократить времени на обработку данных.

Таким образом, в ходе выполнения выпускной квалификационной работы все поставленные задачи были решены, цель выпускной квалификационной работы достигнута.

СПИСОК ЛИТЕРАТУРЫ