Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический университет имени П. О. Сухого»

Кафедра «Промышленная электроника»

ПРОЕКТИРОВАНИЕ ИНТЕГРИРОВАННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

ПРАКТИКУМ
по выполнению лабораторных работ для студентов специальности 1-39 80 03 «Электронные системы и технологии» дневной и заочной форм обучения

Гомель 2023

УДК 004.65(075.8)
ББК 32.81я73
П79

Рекомендовано научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 4 от 06.12.2021 г.*)

Составители: *Н. А. Красовская*, *В. В. Щуплов*

Рецензент: доц. каф. «Информационные технологии» ГГТУ им. П. О. Сухого
канд. техн. наук, доц. *В. С. Захаренко*

Проектирование интегрированных информационных систем : практикум по выполнению лаборатор. работ для студентов специальности 1-39 80 03 «Электронные системы и технологии» днев. и заоч. форм обучения / сост.: Н. А. Красовская, В. В. Щуплов. – Гомель : ГГТУ им. П. О. Сухого, 2023. – 86 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

П79

Содержит восемь лабораторных работ с краткими теоретическими сведениями, порядком их выполнения, заданиями для самостоятельной работы и контрольными вопросами.

Для студентов специальности 1-39 80 03 «Электронные системы и технологии» дневной и заочной форм обучения.

УДК 004.65(075.8)
ББК 32.81я73

Лабораторная работа № 1

Принципы работы со встраиваемой СУБД SQLite

1. Цель работы

Изучить принципы работы со встраиваемой СУБД SQLite, с применением консольного тонкого клиента.

2. Основные теоретические сведения

2.1. Что такое SQLite

SQLite – компактная встраиваемая реляционная база данных. Исходный код библиотеки передан в общественное достояние.

Слово «встраиваемый» (embedded) означает, что SQLite не использует парадигму клиент-сервер, т. е. движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; ACID-функции достигаются в том числе за счёт создания файла журнала.

Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки. Другим вариантом развития событий является автоматическое повторение попыток записи в течение заданного интервала времени.

В комплекте поставки идёт также функциональная клиентская часть в виде исполняемого файла sqlite3, с помощью которого

демонстрируется реализация функций основной библиотеки. Клиентская часть работает из командной строки, позволяет обращаться к файлу БД на основе типовых функций ОС.

Благодаря архитектуре движка возможно использовать SQLite как на встраиваемых системах, так и на выделенных машинах с гигабайтными массивами данных.

2.2. Классы хранение и типы данных

В большинстве движков баз данных SQL используется строгая статическая типизация. При статической типизации, тип данных того или иного значения определяется его контейнером – конкретным столбцом, в котором это значение хранится.

SQLite использует более общую систему типизации – динамическую, когда тип данных значения связан с самим значением, а не с его контейнером. Динамическая система SQLite имеет обратную совместимость со статическими системами других СУБД. В том смысле, что SQL-запросы статически типизированных баз данных должны работать так же и с SQLite. Однако , динамическая типизация в SQLite позволяет выполнять операции, невозможные в традиционных жестко типизированных базах данных.

Каждое значение, хранящееся в базе данных SQLite (или обрабатываемое движком), имеет один из следующих классов хранения:

- NULL – пустое значение в таблице базы.
- INTEGER – целочисленное значение, хранящееся в 1, 2, 3, 4, 6 или 8 байтах, в зависимости от величины самого значения.
- REAL – числовое значение с плавающей точкой. Хранится в формате 8-байтного числа IEEE с плавающей точкой.
- TEXT – значение строки текста. Хранится с использованием кодировки базы данных (UTF-8, UTF-16BE или UTF-16LE).
- BLOB – значение бинарных данных, хранящихся точно в том же виде, в каком были введены.

Стоит отметить, что класс хранения – более широкое понятие, чем тип данных. К примеру, класс хранения INTEGER включает 6 различных типов целочисленных данных различной длины. На диске это записывается по-разному. Но как только целочисленные значения

считываются с диска и поступают для обработки в оперативную память, они преобразуются в наиболее общий тип данных (8-байтное целое число). Следовательно, хранение по системе класса данных в практическом плане неотличимо от хранения по типу данных, и они могут быть взаимозаменяемыми.

Любой столбец базы данных SQLite версии 3, за исключением столбцов INTEGER PRIMARY KEY, может быть использован для хранения значений любого класса.

Все значения в инструкциях SQL, являются ли они литералами или параметрами, встроенными в строку SQL-запроса в случае прекомпилируемых инструкций SQL, имеют неявный класс хранения. В условиях, описанных ниже, движок базы данных может конвертировать значения между числовыми классами хранения (INTEGER и REAL) и TEXT во время выполнения запроса.

2.2.1. *Логические типы данных*

SQLite не имеет отдельного логического класса хранения. Вместо этого, логические значения хранятся как целые числа 0 (false) и 1 (true).

2.2.2. *Типы данных даты и времени*

SQLite не имеют классов, предназначенных для хранения дат и/или времени. Вместо этого встроенные функции даты и времени в SQLite способны работать с датами и временем, сохраненными в виде значений TEXT, REAL и INTEGER в следующих форматах:

- TEXT как строка формата ISO8601 ("YYYY-MM-DD HH:MM:SS.SSS").
- REAL как числа юлианского календаря. То есть число дней с полудня 24 ноября 4714 г. до н. э. по Гринвичу в соответствии с ранним григорианским календарём.
- INTEGER как время Unix, – количество секунд с 1970-01-01 00:00:00 UTC.

В приложениях следует выбирать, в каком из этих форматов хранить даты и время, а затем можно свободно конвертировать из одного формата в другой с помощью встроенных функций даты и времени.

2.2.3. *Аффинированные типы*

В целях обеспечения максимальной совместимости между SQLite и другими СУБД, SQLite поддерживает концепцию аффинированных типов для столбцов таблиц. Аффинированный тип – это тот, который является рекомендуемым для сохраняемых в столбце значений. Важной идеей здесь является то, что тип рекомендуется, но не является обязательным. В любом столбце можно по-прежнему хранить данные любого типа. Просто в некоторых столбцах, при наличии выбора, одному классу хранения будет оказано предпочтение перед другим. Предпочтительный класс для хранения данных в столбце называется аффинированным.

Каждому столбцу в базе данных SQLite версии 3 может быть присвоен один из следующих аффинированных типов:

— TEXT
— NUMERIC
— INTEGER
— REAL
— NONE

Столбцы с аффинированным типом TEXT предназначены для хранения данных классов NULL, TEXT или BLOB. Если в столбец с аффинированным TEXT вставляются числовые данные, они перед сохранением конвертируются в текстовую форму.

Столбец с аффинированным NUMERIC может содержать значения с использованием всех пяти классов хранения. Когда в столбец с родством NUMERIC вставляются текстовые данные, текст конвертируется в INTEGER или REAL (в порядке предпочтения), если такое преобразование возможно без потерь и имеет обратимый характер. Касаемо преобразования между TEXT и REAL, SQLite считает, что конвертирование является обратимым и без потерь, если можно восстановить первые 15 значимых десятичных цифр числа. Если преобразование TEXT в INTEGER или REAL не представляется возможным без потерь, то значение сохраняется с помощью класса хранения TEXT. Попытки конвертирования в NULL или BLOB не предпринимаются.

2.3. Утилита sqlite3

Утилита sqlite3 представляет собой консольный тонкий клиент для баз SQLite.

Мета Команды - предназначены для формирования таблиц и других административных операций, описанные в табл. 1.

Таблица 1

Основные мета команды

Команда	**Описание**
.show	Показывает текущие настройки заданных параметров
.databases	Показывает название баз данных и файлов
.quit	Выход из sqlite3
.tables	Показывает текущие таблицы
.schema	Отражает структуру таблицы
.header	Отобразить или скрыть шапку таблицы
.mode	Выбор режима отображения данных таблицы
.dump	Сделать копию базы данных в текстовом формате

Также утилита поддерживает стандартные команды.

Язык описания данных DDL**:** команды для создания таблицы, изменения и удаления баз данных, таблиц и прочего.

CREATE
ALTER
DROP

Язык управления данными DML**:** позволяют пользователю манипулировать данными (добавлять/изменять/удалять).

INSERT
UPDATE
DELETE

Язык запросов DQL**:** позволяет осуществлять выборку данных командой:

SELECT

Для запуска утилиты необходимо вызвать командную строку (cmd – для Windwos или terminal – для linux) и запустить исполняемый файл sqlite3 указав в качестве параметра файл базы данных.

3. Порядок выполнения работы

3.1. Создание базы данных

В качестве примера, рассмотрим табл. 2 создадим базу данных test.db состоящую из одной таблицы следующего вида:

Таблица 2

Перечень компьютерных комплектующих

ID	Наименование	Цена	Количество
1	Жесткий диск USB 250 Gb G-Tech GDM4_250_EMEA	97	1
2	Видеокарта GeForce GTX1050	381	5
3	Колонки CBR CMS-580	28	2
4	Беспроводная клавиатура BTC 9039ARF III Black	92	8
5	Монитор 22" LG 22M38A-B Black	259	2
6	МФУ Ricoh SP150SU	239	5
7	UPS FSP FP-850 Black	164	7

Определим типы данных для каждого из столбцов табл. 2.

ID – Целочисленное значение, котороеявлется ключевым полем таблицы. Данное поле так же должно быть уникально для каждой записи. Тип данных выбираем целочисленный INTEGER. Так же необходимо указать флаг ключевого поля (PRIMARY KEY), флаг автоинкремента (AUTOINCREMENT) и флаг указывающий на то что значение не может быть нулевым (NOT NULL).

Наименование – текстовое поле, содержащее наименование товара. Тип данных выбираем TEXT. А также указываем флаг NOT NULL.

Цена – целочисленное значение, содержащее цену товара. Тип данных выберем целочисленный (INTEGER). А также указываем флаг NOT NULL.

Количество – также целочисленное поле, содержащее количество товаров на складе. Тип данных выберем целочисленный (INTEGER). А также указываем флаг NOT NULL.

Для создания таблиц применяется команда CREATE TABLE. Формат команды:

CREATE TABLE <имя таблицы> (поле 1 тип данных флаги, поле 2 тип данных флаги, …).

Если неизвестно создана таблица или нет используется второй враиант команды:

CREATE TABLE IF NOT EXISTS <имя таблицы> (поле 1 тип данных флаги, поле 2 тип данных флаги, …);

Данная команда создаст новую таблицу, с указанными параметрами, если не существует таблицы с таким же именем.

Команда для создания таблицы товаров, с учётом выбранных типов данных и флагов, будет выглядеть следующим образом:

```
CREATE TABLE product (
id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
name TEXT NOT NULL,
cost INTEGER NOT NULL,
quantityINTEGERNOTNULL);
```

После предыдущей команды можно проверить создалась ли таблица. Для этого выполним мета команду .tables. Для просмотра структуры таблицы можно ввести мета команду .schema.

Теперь необходимо занести данные в таблицу. Для вставки данных в таблицу используется команда insert. Формат команды:

```
INSERT INTO <название таблицы> ([<Имя столбца>, ... ])
VALUES (<Значение>,...);
```

Заполним данными нашу таблицу, для этого выполним следующую последовательность команд:

```
INSERT INTO product (name,cost,quantity)
VALUES ('Жесткий диск USB 250 Gb G-Tech GDM4_250_EMEA','97','1');

INSERT INTO product (name,cost,quantity)
VALUES ('Видеокарта GeForce GTX1050','381','5');

INSERT INTO product (name,cost,quantity)
VALUES ('Колонки CBR CMS-580','28','2');

INSERT INTO product (name,cost,quantity)
```

```
VALUES ('Беспроводная клавиатура BTC 9039ARF III Black','92','8');

INSERT INTO product (name,cost,quantity)
VALUES ('Монитор 22" LG 22M38A-B Black','259','2');

INSERT INTO product (name,cost,quantity)
VALUES ('МФУ Ricoh SP150SU','239','5');

INSERT INTO product (name,cost,quantity)
VALUES ('UPS FSP FP-850 Black','164','7');
```

Для просмотра содержимого таблицы можно вести команду выбора без условия:

```
SELECT * FROM product;
```

Если все предыдущие команды были выполнены правильно на экране вы увидите содержания таблицы. Обратите внимание что поле id мы явным образом не задавали при внесении данных, однако каждой записи присвоено значение. Данную функцию обеспечивает флаг AUTOINCREMENT в свойствах поля.

3.2. Задание для самостоятельного выполнения

Разработайте и создайте базу данных, содержащую список группы (имена, отчества и фамилии должны располагаться в разных столбцах) и экзаменационные оценки предыдущей сессии по всем предметам.

4. Содержание отчета

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а также снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаний после выполнения работы.

5. Контрольные вопросы

1. Что такое База данных?
2. Что такое SQLite?
3. Какие типы данных поддерживает SQLite?
4. С помощью какой мета команды можно просмотреть перечень таблиц в базе данных?
5. С помощью какой мета команды можно просмотреть структуру таблицы?
6. С помощью какой команды выполняется создание таблицы?
7. С помощью какой команды вносятся данные в таблицу?
8. С помощью какой команды можно просмотреть содержимое таблицы?

Лабораторная работа № 2

Принципы работы со встраиваемой СУБД FireBird

1. Цель работы

Изучить принципы работы со встраиваемой СУБД FireBird, с применением консольного тонкого клиента.

2. Основные теоретические сведения

2.1. Что такое FireBird

Firebird (FirebirdSQL) – кроссплатформенная система управления базами данных (СУБД), работающая на Mac OS X, Linux, MicrosoftWindows и разнообразных Unix платформах.

Firebird используется в различных промышленных системах (складские и хозяйственные, финансовый и государственный сектора) с 2001 г. Это коммерчески независимый проект C и C++ программистов, технических советников.

Firebird является сервером баз данных. Один сервер Firebird может обрабатывать несколько сотен независимых баз данных, каждую с множеством пользовательских соединений. Он является полностью свободным от лицензионных отчислений даже для коммерческого использования.

Основные характеристики:

1. Соответствие требованиям ACID: Firebird сделан специально, чтобы удовлетворять требованиям «атомарности, целостности, изоляции и надёжности» транзакций («Atomicity, Consistency, IsolationandDurability»).

2. Версионная архитектура: основная особенность Firebird – версионная архитектура, позволяющая серверу обрабатывать различные версии одной и той же записи в любое время таким образом, что каждая транзакция видит свою версию данных, не мешая соседним («читающие транзакции не блокируют пишущих, а пишущие – не блокируют читающих»). Это позволяет использовать одновременно OLTP и OLAP запросы.

3. Хранимые процедуры: используя язык PSQL (процедурный SQL) Firebird, можно создавать сложные хранимые процедуры для обработки данных полностью на стороне сервера. Для генерации отчётов особенно удобны хранимые процедуры с возможностью выборки, возвращающие данные в виде набора записей. Такие процедуры можно использовать в запросах точно так же, как и обычные таблицы.

4. События: хранимые процедуры и триггеры могут генерировать события, на которые может подписаться клиент. После успешного завершения транзакции (COMMIT) он будет извещён о произошедших событиях и их количестве.

5. Генераторы: идея генераторов (последовательностей) делает возможной простую реализацию автоинкрементных полей, и не только их. Генераторы являются 64 битными, хранимыми в базе данных счётчиками, работающими независимо от транзакций. Они могут быть использованы для различных целей, таких как генерация первичных ключей, управление длительными запросами в соседних транзакциях и т. д.

6. Базы данных только для чтения: позволяют распространять базы данных, к примеру, на CD-ROM. Особенно упрощает распространение данных их использование в комбинации со встраиваемой версией сервера Firebird (FirebirdEmbedded).

7. Полный контроль за транзакциями: одно клиентское приложение может выполнять множество одновременных транзакций. В разных транзакциях могут быть использованы разные уровни изоляции. Протокол двухфазного подтверждения транзакций обеспечивает гарантированную устойчивость при работе с несколькими базами данных. Также доступны оптимистическое блокирование данных и точки сохранения транзакций.

8. Резервное копирование на лету: для резервного копирования нет надобности останавливать сервер. Процесс резервного копирования сохраняет состояние базы данных на момент своего старта, не мешая при этом работе с базой. Кроме того, существует возможность производить инкрементальное резервное копирование БД.

9. Триггеры: для каждой таблицы возможно назначение нескольких триггеров, срабатывающих до или после вставки, обновления или удаления записей. Для триггеров используется язык PSQL, позволяя вносить начальные значения, проверять целостность

данных, вызывать исключения и т. д. В Firebird 1.5 появились «универсальные» триггеры, позволяющие в одном триггере обрабатывать вставки, обновления и удаления записей таблицы.

10. Внешние функции: библиотеки с UDF (UserDefinedFunction) могут быть написаны на любом языке и легко подключены к серверу в виде DLL/SO, позволяя расширять возможности сервера «изнутри».

11. Декларативное описание ссылочной целостности: Обеспечивает непротиворечивость и целостность многоуровневых отношений «master-detail» между таблицами.

12. Наборы символов: Firebird поддерживает множество международных наборов символов (включая Unicode) с множеством вариантов сортировки.

Firebird работает на аппаратных платформах x86, x64 и PowerPC, Sparc и многих других, и поддерживает легкий переход между этими платформами. Может использоваться даже не очень мощное оборудование, особенно под Linux. И, как в любой СУБД, на производительность влияют: количество памяти, скорость работы дисковой подсистемы и т. д. Рекомендации для выбора аппаратного обеспечения зависят от требования к системе, прогнозируемого размера базы данных, количества пользователей и т. п. Допустимо начинать с минимальной конфигурации, расширяя её по мере надобности.

Существует четыре различных взаимозаменяемых архитектуры сервера:

1. ClassicServer – один процесс на одно соединение; поддержка многопроцессорных машин.
2. SuperServer – все соединения используют один процесс, меньшие требования к памяти при большем быстродействии; для многопроцессорных машин (до 3.0 для однопроцессорных).
3. SuperClassicServer – один процесс, но свой поток на каждое соединение. Данная архитектура введена в версии 2.5.
4. Embedded (встраиваемая) версия – весь движок содержится в одной библиотеке с именем клиентской библиотеки сервера, идеально подходит для однопользовательских систем, не требует инсталляции в Windows.

Все архитектуры используют одинаковый формат файла базы данных, таким образом, в любой момент можно переключиться на другую архитектуру.

Firebird выпускается под условиями IPL (InterBasePublicLicense) и IDPL (InitialDeveloper'sPublicLicense), которые совпадают с условиями MozillaPublicLicense 1.1. Firebird полностью бесплатен для использования и распространения (в том числе и коммерческого). Раскрытие исходного кода вашего продукта не требуется, вне зависимости от используемой модели лицензирования. Однако, в случае, если вы модифицировали исходный код сервера, то необходимо сделать доступным исходный код ваших модификаций.

2.2. Типы данных

Перечень основных типов данных, поддерживаемых FireBird 3.0, описанн в табл. 1

Просмотрев табл. 1 может возникнуть вопрос: а зачем столько много типов данных, ведь можно обойтись и меньшим количество? Если вспомнить основные задачи СУБД, то можно понять, чем точнее выбран тип данных для конкретной задачи – тем более компактной будет база данных, а следовательно, и обработка информации будет происходить намного быстрее.

Таблица 1

Типы данных FireBird 3.0

Название	Размер	Диапазон и точность	Описание
BIGINT	64 бита		Тип данных доступен только в 3 диалекте
BOOLEAN	8 бит	false, true, unknown	Логический тип данных
BLOB	Переменный	Нет. Размер сегмента BLOB ограничивается 64К. Максимальный размер поля BLOB 4 Гб. Для размера страницы 4096 максимальный размер BLOB поля несколько ниже 2 Гб	Тип данных с динамически изменяемым размером для хранения больших данных, таких как графика, тексты, оцифрованные звуки. Базовая структурная единица – сегмент. Подтип Blob описывает содержимое

Продолжение табл. 1

Название	Размер	Диапазон и точность	Описание
CHAR(n) CHARACTER(n)	n символов (размер в байтах зависит от кодировки, кол-во байт на символ)	от 1 до 32 767 байт	Символьный тип данных фиксированной длины. При отображении данных, строка дополняется пробелами справа до указанной длины. Если количество символов n не указано, то по умолчанию принимается 1.
DATE	32 бита	От 01.01.0001 н.э. до 31.12.9999 н.э.	ISC_DATE
DECIMAL (precision, scale)	Переменный (16, 32 или 64 бита)	precision = от 1 до 18, указывает, по меньшей мере, количество цифр для хранения; scale = от 0 до 18. Задаёт количество знаков после разделителя	scale должно быть меньше или равно precision. Число с десятичной точкой, имеющей после точки scale разрядов. Пример: DECIMAL(10,3) содержит число точно в следующем формате: ppppppp.sss.
DOUBLE PRECISION	64 бита		IEEE двойной точности, 15 цифр, размер зависит от платформы
FLOAT	32 бита		IEEE одинарной точности, 7 цифр
INTEGER INT	32 бита	–2 147 483 648 .. 2 147 483 647	signedlong
NUMERIC (precision, scale)	Переменный (16, 32 или 64 бита)	precision = от 1 до 18, указывает, по меньшей мере, количество цифр для хранения; scale = от 0 до 18. Задаёт количество знаков после разделителя.	scale должно быть меньше или равно precision. Число с десятичной точкой, имеющей после точки scale разрядов. Пример: NUMERIC(10,3) содержит число точно в следующем формате: ppppppp.sss.
SMALLINT	16 бит	–32 768 .. 32 767	signedshort (word)

Окончание табл. 1

Название	Размер	Диапазон и точность	Описание
TIME	32 бита	От 0:00 до 23:59:59.9999	ISC_TIME
TIMESTAMP	64 бита	От 01.01.0001 н.э. до 31.12.9999 н.э.	Включает информацию и о времени
VARCHAR(n) CHAR VARYING CHARACTER VARYING	n символов (размер в байтах зависит от кодировки, кол-ва байт на символ)	От 1 до 32 765 байтов	Размер символов в байтах с учётом их кодировки не может быть больше 32765. Для этого типа данных, в отличие от CHAR (где по умолчанию предполагается количество символов 1),количество символов n обязательно должно быть указано

2.2.1. *Целочисленные типы данных*

Для целых чисел используют целочисленные типы данных SMALLINT, INTEGER и BIGINT (в 3 диалекте). Firebird не поддерживает беззнаковый целочисленный тип данных.

SMALLINT

Тип данных SMALLINT представляет собой 16-битное целое. Он применяется в случае, когда не требуется широкий диапазон возможных значений для хранения данных. Числа типа SMALLINT находятся в диапазоне, или –32 768 .. 32 767.

INTEGER

Тип данных INTEGER представляет собой 32-битное целое. Сокращённый вариант записи типа данных INT. Числа типа INTEGER находятся в диапазоне, или –2 147 483 648 .. 2 147 483 647.

BIGINT

BIGINT – это SQL-99-совместимый 64 битный целочисленный тип данных. Он доступен только в 3-м диалекте. При использовании клиентом диалекта 1, передаваемое сервером значение генератора

усекается до 32-х битного целого (INTEGER). При подключении в 3-м диалекте значение генератора имеет тип BIGINT. Числа типа BIGINT находятся в диапазоне , или -9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807.

Числа типа BIGINT могут быть заданы в шестнадцатеричном виде с 9 – 6 шестнадцатеричными цифрами. Более короткие шестнадцатеричные числа интерпретируются как тип данных INTEGER.

2.2.2. *Типы данных с плавающей точкой*

Типы данных с плавающей точкой хранятся в двоичном формате IEEE 745, который включает в себя знак, показатель степени и мантиссу. Точность этого типа является динамической, что соответствует физическому формату хранения, который составляет 4 байта для типа FLOAT и 8 байт для типа DOUBLEPRECISION.

Учитывая особенности хранения чисел с плавающей точкой, этот тип данных не рекомендуется использовать для хранения денежных данных. По тем же причинам не рекомендуется использовать столбцы с данными такого типа в качестве ключей и применять к ним ограничения уникальности.

При использовании таких типов данных в выражениях рекомендуется крайне внимательно и серьёзно подойти к вопросу округления результатов расчётов.

FLOAT

Данный тип данных обладает приблизительной точностью 7 цифр после запятой. Для обеспечения надёжности хранения полагайтесь на 6 цифр.

DOUBLE PRECISION

При хранении данных, предполагается приблизительная точность 15 цифр.

2.2.3. *Типы данных с фиксированной точкой*

Данные типы данных позволяют применять их для хранения денежных значений и обеспечивают предсказуемость операций умножения и деления.

Firebird предлагает два типа данных с фиксированной точкой: NUMERIC и DECIMAL. В соответствии со стандартом оба типа ограничивают хранимое число объявленным масштабом (количеством чисел после запятой). При этом подход к тому, как ограничивается точность для типов разный: для столбцов NUMERIC точность является такой, «как объявлено», в то время как DECIMAL столбцы могут получать числа, чья точность, по меньшей мере, равна тому, что было объявлено.

Например, NUMERIC(4, 2) описывает число, состоящее в общей сложности из четырёх цифр, включая 2 цифры после запятой; итого 2 цифры до запятой, 2 после. При записи в столбец с этим типом данных значений 3,1415 в столбце NUMERIC(4, 2) будет сохранено значение 3,14.

Для данных с фиксированной точкой общим является форма декларации, например NUMERIC(p, s). Здесь важно понять, что в этой записи s – это масштаб, а не интуитивно предсказываемое «количество знаков после запятой». Для «визуализации» механизма хранения данных запомните для себя процедуру:

1. При сохранении в базу данных число умножается на 10s (10 в степени s), превращаясь в целое;
2. При чтении данных происходит обратное преобразование числа.

Способ физического хранения данных в СУБД зависит от нескольких факторов: декларируемой точности, диалекта базы данных, типа объявления.

NUMERIC

Формат объявления данных:

NUMERIC(p, s)

В зависимости от точности p и масштаба s СУБД хранит данные по-разному. Рассмотрим примеры того, как СУБД хранит данные в зависимости от формы их объявления, описанные в табл. 2.

Таблица 2

Хранение типа данных NUMERIC в СУБД

NUMERIC(4)	SMALLINT
NUMERIC(4,2)	SMALLINT (data *
NUMERIC(10,4)	DOUBLE PRECISION в 1-омдиалекте BIGINT в 3-ем диалекте (data *

Всегда надо помнить, что формат хранения данных зависит от точности. Например, вы задали тип столбца NUMERIC(2, 2), предполагая, что диапазон значений в нем будет – 0.99...0.99. Однако в действительности диапазон значений в столбце будет -327.68..327.68, что объясняется хранением типа данных NUMERIC(2, 2) в формате SMALLINT. Фактически типы данных NUMERIC(4, 2), NUMERIC(3, 2) и NUMERIC(2, 2) являются одинаковыми. Т.е. Для реального хранения данных в столбце с типом данных NUMERIC(2, 2) в диапазоне -0.99...0.99 для него надо создавать ограничение.

DECIMAL

Формат объявления данных:

DECIMAL(p, s)

Формат хранения данных в базе во многом аналогичен NUMERIC, хотя существуют некоторые особенности, которые проще всего пояснить на примере.

Рассмотрим примеры того, как СУБД хранит данные в зависимости от формы их объявления, описанные в табл. 3.

Таблица 3

Хранение данных типа DECIMAL в СУБД

DECIMAL(4)	INTEGER
DECIMAL(4,2)	INTEGER (data *)
DECIMAL(10,4)	DOUBLE PRECISION в 1-омдиалекте BIGINT в 3-ем диалекте (data *)

2.2.4. *Типы данных для работы с датой и временем*

В СУБД Firebird для работы с данными, содержащими дату и время, используются типы данных DATE, TIME, TIMESTAMP. В 3-м диалекте присутствуют все три вышеназванных типа данных, а в 1-м для операций с датой и временем доступен только тип данных DATE, который не тождественен типу данных DATE 3-го диалекта, а напоминает тип данных TIMESTAMP из 3-го диалекта.

В типах DATETIME и TIME Firebird хранит секунды с точностью до десятитысячных долей. Если вам необходима более низкая гранулярность, то точность может быть указана явно в виде

тысячных, сотых или десятых долей секунды в базах данных в 3 диалекте и ODS 11 и выше.

DATE

В 3-м диалекте тип данных DATE, как это и следует предположить из названия, хранит только одну дату без времени. В 1-м диалекте нет типа данных "только дата".

Допустимый диапазон хранения от 1 января 1 г. н. э. до 31 декабря 9999 года.

TIME

Этот тип данных доступен только в 3-м диалекте. Позволяет хранить время дня в диапазоне от 00:00:00.0000 до 23:59:59.9999. При необходимости получения времени из DATE. в 1-м диалекте можно использовать функцию EXTRACT.

Примериспользования EXTRACT:

EXTRACT (HOUR FROM DATE_FIELD)
EXTRACT (MINUTE FROM DATE_FIELD)
EXTRACT (SECOND FROM DATE_FIELD)

TIMESTAMP

Этот тип данных доступен только в 3-м диалекте, состоит из двух 32-битных слов и хранит дату со временем. Такое хранение эквивалентно типу DATE 1-го диалекта.

2.2.5. *Символьные типы данных*

В СУБД Firebird для работы с символьными данными есть тип данных фиксированной длины CHAR и строковый тип данных VARCHAR переменной длины. Максимальный размер текстовых данных, хранящийся в этих типах данных, составляет 32767 байт для типа CHAR и 32765 байт для типа VARCHAR. Максимальное количество символов, которое поместится в этот объём, зависит от используемого набора символов CHARACTER SET и/или заданного порядка сортировки, который для символьных данных задаётся предложением COLLATE.

В случае отсутствия явного указания набора символов при описании текстового объекта базы данных будет использоваться набор символов по умолчанию, заданный при создании базы данных. При отсутствии явного указания набора символов, а также отсутствия набора символов по умолчанию в базе данных, поле получает набор символов CHARACTER SET NONE.

В настоящее время все современные средства разработки поддерживают Unicode. При возникновении необходимости использования восточноевропейских текстов в строковых полях базы данных или для более экзотических алфавитов, рекомендуется работать с набором символов UTF8. При этом следует иметь в виду, что на один символ в данном наборе приходится до 4 байт. Следовательно, максимальный размер символов в символьных полях составит 32676/4 (8192) байта на символ. При этом следует обратить внимание, что фактически значение параметра «байт на символ» зависит от диапазона, к которому принадлежит символ: английские буквы занимают 1 байт, русские буквы кодировки WIN1251 – 2 байта, остальные символы – могут занимать до 4-х байт.

CHAR

CHAR является типом данных фиксированной длины. Если введённое количество символом меньше объявленной длины, то поле дополнится концевыми пробелами. В общем случае символ заполнитель может и не являться пробелом, он зависит от набора символов, так например, для набора символов OCTETS – это ноль.

В случае если не указана длина, то считается, что она равна единице.

Данный тип символьных данных можно использовать для хранения в справочниках кодов,длина которых стандартна и определённой «ширины». Примером такого может служить почтовый индекс в России – 6 символов.

VARCHAR

Является базовым строковым типом для хранения текстов переменной длины, поэтому реальный размер хранимой структуры равен фактическому размеру данных плюс 2 байта, в которых задана длина поля.

Все символы, которые передаются с клиентского приложения в базу данных, считаются как значимые, включая начальные и конечные пробельные символы.

Полное название CHARACTER VARYING. Имеется и сокращённый вариант записи CHAR VARYING.

NCHAR

Представляет собой символьный тип данных фиксированной длины с предопределённым набором символов ISO8859_1.

Синонимом является написание NATIONAL CHAR.

Аналогичный тип данных доступен для строкового типа переменной длины: NATIONAL CHARACTER VARYING.

2.2.6. *Логический тип данных*

SQL-2008 совместимый тип данных BOOLEAN (8 бит) включает различные значения истинности TRUE и FALSE. Если не установлено ограничение NOT NULL, то тип данных BOOLEAN поддерживает также значение истинности UNKNOWN как NULL значение.

Спецификация не делает различия между значением NULL этого типа и значением истинности UNKNOWN, которое является результатом SQL предиката, поискового условия или выражения логического типа. Эти значения взаимозаменяемы и обозначают одно и то же.

2.2.7 *Бинарный тип данных*

BLOB

BLOB (BinaryLargeObjects, большие двоичные объекты) представляют собой сложные структуры, предназначенные для хранения текстовых и двоичных данных неопределённой длины, зачастую очень большого объёма.

Синтаксис:

BLOB [SUB_TYPE <subtype>]
[SEGMENT SIZE <seg_length>]
[CHARACTER SET <charset>]

Сокращённый синтаксис:

BLOB (<seg_length>)
BLOB (<seg_length>, <subtype>)
BLOB (, <subtype>)

Размер сегмента: Указание размера сегмента BLOB является некоторым атавизмом, оно идёт с тех времён, когда приложения для работы с данными BLOB писались на C (Embedded SQL) при помощи GPRE. В настоящий момент размер сегмента при работе с данными BLOB определяется клиентской частью, причём размер сегмента может превышать размер страницы данных.

Подтип BLOB отражает природу данных, записанную в столбце. Firebird предоставляет два предопределённых подтипа для сохранения пользовательских данных:

1. Подтип 0 (BINARY): Если подтип не указан, то данные считаются не типизированными и значение подтипа принимается равным 0. Псевдоним подтипа 0 – BINARY. Этот подтип указывает, что данные имеют форму бинарного файла или потока (изображение, звук, видео, файлы текстового процессора, PDF и т.д.).
2. Подтип 1 (TEXT): Подтип 1 имеет псевдоним TEXT, который может быть использован вместо указания номера подтипа. Например, BLOB SUBTYPE TEXT. Это специализированный подтип, который используется для хранения текстовых данных большого объёма. Для текстового подтипа BLOB может быть указан набор символов и порядок сортировки COLLATE, аналогично символьному полю.

Максимальный размер поля BLOB ограничен 4 Гб и не зависит от варианта сервера, 32 битный или 64 битный (во внутренних структурах, связанных с BLOB присутствуют 4-х байтные счётчики). Для размера страницы 4096 максимальный размер BLOB поля несколько ниже 2 Гб.

3. Порядок выполнения работы

3.1. Создание базы данных

В качестве примера создадим базу данных test.db, состоящую из одной таблицы следующего вида (табл. 4):

Таблица 4

База данных

ID	Название фильма	Название эпизода	Бюджет	Сборы	Дата выхода
1	Звёздные войны	Эпизод IV: Новая надежда	11 000 000	775 398 007	25.05.1977
2	Звёздные войны	Эпизод V: Империя наносит ответный удар	16 000 000	538 375 067	17.05.1980
3	Звёздные войны	Эпизод VI: Возвращение джедая	32 500 000	475 106 177	25.05.1983
4	Звёздные войны	Эпизод I: Скрытая угроза	115 000 000	474 544 677	16.05.1999
5	Звёздныевойны	Эпизод II: Атака клонов	115 000 000	649 398 328	16.05.2002
6	Звёздные войны	Эпизод III: Месть ситхов	113 000 000	848 754 768	17.05.2005
7	Звёздные войны	Эпизод VII: Пробуждение Силы	245 000 000	2 068 223 624	14.12.2015
8	Звёздные войны	Звёздные войны: Истории	200 000 000	1 035 462 946	14.12.2016
9	Пираты карибского моря	Проклятие Чёрной жемчужины	140 000 000	654 264 015	23.06.2003
10	Пираты карибского моря	Сундук мертвеца	225 000 000	1 066 179 725	24.06.2006
11	Пираты карибского моря	На краю света	300 000 000	963 420 425	19.05.2007
12	Пираты карибского моря	На странных берегах	350 000 000	1 045 713 802	18.05.2011

В табл. 4 могут содержаться как текстовые, так и цифровые данные. Определим типы данных для каждого из столбцов таблицы:

ID – Целочисленное значение, которое является ключевым полем таблицы. Данное поле так же должно быть уникально для каждой записи. Тип данных выбираем целочисленный INTEGER. Так же необходимо указать флаг ключевого поля (PRIMARY KEY) и флаг указывающий на то что значение не может быть нулевым (NOT NULL). Стоит отметить что FireBird не имеет флага автоинкремента. Как реализовать данную функцию – рассмотрим позже.

Название фильма – текстовое поле, содержащее название саги. Так как дляна названия может быть различной, выберем тип данных VARCHAR. А также указываем флаг NOT NULL.

Название эпизода – это так же текстовое значение и содержит название определенной части фильма. По своим параметрам данное поле аналогично предыдущему значении.

Бюджет – содержит целочисленное значение, и отображает изначальный бюджет эпизода. Тип данных выберем целочисленный (INTEGER). А также указываем флаг NOT NULL.

Сборы – содержит так же целочисленное значение и отображает сборы эпизода в прокате. По своим параметрам данное поле аналогично предыдущей записи.

Дата выхода – содержит дату выхода эпизода в прокат. Тип данных выберем DATE, т.к. нам необходимо хранить дату без указания конкретного времени. А также указываем флаг NOT NULL.

После определения параметров каждого поля приступим к созданию базы данных. Для работы с базами данных FireBird имеет встроенную консольную утилиту isql. Данная утилита находится в папке с установленной СУБД. По умолчанию c:\Program Files\Firebird\Firebird_3_0\ isql.exe. Но данный путь зависит от системы и версии СУБД. Так же, при установке FireBird, создается набор ярлыков в системном меню. Для запуска консольного консольного клиента необходимо открыть системное меню (пуск), затем выбрать «Программы», затем папку с необходимой версией FireBird (напримерFirebird 3.0) и запустить Firebird ISQL Tool.

После запуска на экране появится окно консольного клиента с приглашением. Приглашение можно увидеть на рис. 1

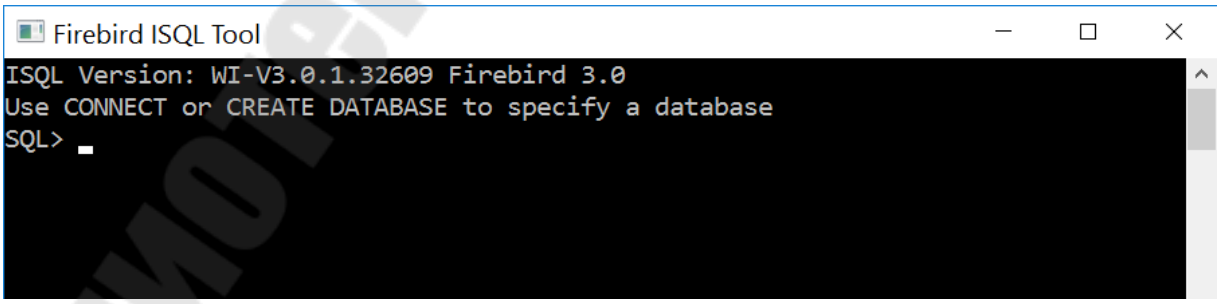


Рис. 1. Окно консольного клиента

В приглашении сказано: Используйте CONNECT или CREATE DATABASE для доступа к базе данных. Т.к. СУБД FireBird

поддерживает большое количество баз данных, сначала необходимо создать необходимую нам базу данных.

Для создания баз данных используется команда CREATE DATABASE. Синтаксис команды:

CREATE DATABASE ‘<путь и имя базы данных>’
user '<имя пользователя>' password '<пароль>';

Т.к. FireBird поддерживает многопользовательский режим, то при создании базы данных неоходимо указать имя пользователя и пароль. В данном случае имя пользователя SYSDBA , а пароль user.

Создание базы данных:

CREATE DATABASE ‘f:/users/films.fdb’
user 'SYSDBA' password 'user';

После создания базы данных необзодимо создать таблицу. Для создания таблиц применяется команда CREATE TABLE. Формат команды:

CREATE TABLE <имя таблицы> (поле 1 тип данных флаги, поле 2 тип данных флаги, …);

Команда для создания таблицы, с учётом выбранных типов данных и флагов, будет выглядеть следующим образом:

```
CREATE TABLE table_films (
id INTEGER NOT NULL PRIMARY KEY,
filmNameVARCHAR(256) NOT NULL,
episodeNemaVARCHAR(256) NOT NULL,
budget INTEGER NOT NULL,
fees INTEGER NOT NULL,
premiereDateDATENOTNULL
);
```

После предыдущей команды можно проверить создалась ли таблица. Для этого выполним мета команду showtable. Для просмотра структуры таблицы можно ввести мета команду showtable<name_table>;.

Теперь необходимо задать алгоритм инкрементирования поля ID. Как уже было сказано, FireBird не поддерживает флаг

автоинкремента поля. Для реализации данной функции чначала необходимо создать генератор:

```
CREATE GENERATOR generator_films;
```

Послегенераторсоздаетсятриггер:

```
SETTERM^ ;
CREATE TRIGGER trigger_films FOR table_films ACTIVE
BEFORE INSERT POSITION 1
AS
BEGIN
   if (new.id is null ) then
   new.id = gen_id (generator_films, 1);
END^
SET TERM ; ^
```

Теперь необходимо занести данные в таблицу. Для вставки данных в таблицу используется команда insert. Формат команды:

```
INSERT INTO <название таблицы> ([<Имя столбца>, ... ])
VALUES (<Значение>,...);
```

Заполним данными нашу таблицу, для этого выполним следующую последовательность команд:

```
INSERT INTO table_films
(filmName,episodeNema,budget,fees,premiereDate)
VALUES ('Звёздные войны','ЭпизодIV: Новая
надежда','11000000','775398007','1977-05-25');

INSERT INTO table_films
(filmName,episodeNema,budget,fees,premiereDate)
VALUES ('Звёздные войны','ЭпизодV: Империя наносит ответный
удар','16000000','538375067','1980-05-17');

INSERT INTO table_films
(filmName,episodeNema,budget,fees,premiereDate)
```

```
VALUES ('Звёздные войны','ЭпизодVI: Возвращение
джедая','32500000','475106177','1983-05-25');

INSERT INTO table_films
(filmName,episodeNema,budget,fees,premiereDate)
VALUES ('Звёздные войны','ЭпизодI: Скрытая
угроза','115000000','474544677','1999-05-16');

INSERT INTO table_films
(filmName,episodeNema,budget,fees,premiereDate)
VALUES ('Звёздные войны','ЭпизодII: Атака
клонов','115000000','649398328','2002-05-16');

INSERT INTO table_films
(filmName,episodeNema,budget,fees,premiereDate)
VALUES ('Звёздные войны','ЭпизодIII: Месть
ситхов','113000000','848754768','2005-05-17');

INSERT INTO table_films
(filmName,episodeNema,budget,fees,premiereDate)
VALUES ('Звёздные войны','ЭпизодVII: Пробуждение
Силы','245000000','2068223624','2015-12-14');

INSERT INTO table_films
(filmName,episodeNema,budget,fees,premiereDate)
VALUES ('Звёздные войны','Звёздные войны:
Истории','200000000','1035462946','2016-12-14');

INSERT INTO table_films
(filmName,episodeNema,budget,fees,premiereDate)
VALUES ('Пираты карибского моря','Проклятие Чёрной
жемчужины','140000000','654264015','2003-06-23');

INSERT INTO table_films
(filmName,episodeNema,budget,fees,premiereDate)
VALUES ('Пираты карибского моря','Сундук
мертвеца','225000000','1066179725','2006-06-24');
```

```
INSERT INTO table_films
(filmName,episodeNema,budget,fees,premiereDate)
VALUES ('Пираты карибского моря','На краю
света','300000000','963420425','2007-05-19');

INSERT INTO table_films
(filmName,episodeNema,budget,fees,premiereDate)
VALUES ('Пираты карибского моря','На странных
берегах','350000000','1045713802','2011-05-18');
```

Для просмотра содержимого таблицы можно вести команду выбора без условия:

```
SELECT * FROM product;
```

3.2. Задание для самостоятельного выполнения

Разработайте и создайте базу данных, содержащую список книг по предмету “Информационное обеспечение систем управлени» (должны содержаться поля: Название книги, авторы, ISBN, год выпуска, страна, издательство). Реализовать ключевое поле с автоматическим инкрементом значений.

4. Содержание отчета

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а также снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаний после выполнения работы.

5. Контрольные вопросы

1. Что такое FireBird?
2. Под какой лицензией распространяется?
3. Как создается база данных?
4. Какие целочисленные данные поддерживает?
5. Как реализуется функция автоинкремента поля?
6. Какие строковые типы данных поддерживаются?

Лабораторная работа № 3

СУБД MySQL.
Консольный тонкий клиент

1. Цель работы

Изучить принципы работы со встраиваемой СУБД MySQL, с применением консольного тонкого клиента.

2. Основные теоретические сведения

2.1. Что такое MySQL

MySQL – свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой SunMicrosystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU GeneralPublicLicense, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

MySQL имеет двойное лицензирование. MySQL может распространяться в соответствии с условиями лицензии GPL. Однако по условиям GPL, если какая-либо программа использует библиотеки (или включает в себя другой GPL-код) MySQL, то она тоже должна распространяться по лицензии GPL. Это может расходиться с планами разработчиков, не желающих открывать исходные тексты своих программ. Для таких случаев предусмотрена коммерческая лицензия, которая также обеспечивает качественную сервисную поддержку. Для свободного программного обеспечения Oracle предоставляет отдельное исключение из правил, явным образом

разрешающее использование и распространение MySQL вместе с ПО, распространяемым под лицензией из определённого Oracle списка.

2.2. Типы данных

Рассмотрим перечень основных типов, поддерживаемых MySQL (табл. 1).

Таблица 1

Типы данных MySQL

Тип данных	Использование	Диапазоны
TINYINT	Очень маленькое целое число	Диапазон числа со знаком от –128 до 127. Диапазон числа без знака (unsigned) от 0 до 255.
SMALLINT	Маленькое целое число	Диапазон числа со знаком от –32768 до 32767. Диапазон числа без знака (unsigned) от 0 до 65535.
MEDIUMINT	Среднее целое число	Диапазон числа со знаком от –8388608 до 8388607. Диапазон числа без знака (unsigned) от 0 до 16777215.
INT INTEGER	Целое число	Диапазон числа со знаком от –2147483648 до 2147483647. Диапазон числа без знака (unsigned) от 0 до 4294967295.
BIGINT	Большое целое число	Диапазон числа со знаком от –9223372036854775808 до 9223372036854775807. Диапазон числа без знака (unsigned) от 0 до 18446744073709551615.
FLOAT	Малое (одинарной точности) число с плавающей запятой. Не может быть числом без знака	Диапазоны от –3.402823466E+38 до –1.175494351E-38, 0 и 1.175494351E-38 до 3.402823466E+38. Если количество знаков после запятой не установлено или <= 24 это число с плавающей запятой одинарной точности.
DOUBLE DOUBLE PRECISION REAL	Нормальное (двойной точности) число с плавающей запятой. Не может быть числом без знака	Диапазоны от -1.7976931348623157E+308 до -2.2250738585072014E-308, 0 и 2.2250738585072014E-308 до 1.7976931348623157E+308. Если количество знаков после запятой не установлени или 25 <= количество знаков <= 53 означает числоа с плавающей запятой двойной точности.

Продолжение табл. 1

Тип данных	Использование	Диапазоны
DECIMAL NUMERIC	Распакованное число с плавающей запятой	Работает подобно типу данных CHAR: «распакованный» означает, что число хранится в виде строки, используя один символ для каждой цифры-значения. Символ десятичной запятой и символ отрицательного числа "-" не учитывается в длину. Если десятичное значение равно 0, значение не будет иметь десятичной запятой или дробной части. Максимальный размер для DECIMAL значение такое же, как и для DOUBLE, но фактический диапазон для данного столбца DECIMAL может быть ограничен в выборе длины и десятичные дроби.
DATE	Дата	Дата в диапазоне от «1000-01-01» до «9999-12-31». MySQL хранит поле типа DATE в виде «YYYY-MM-DD» (ГГГГ-ММ-ДД).
DATETIME	Дата и время	Допустимые диапазоны от «1000-01-01 00:00:00» до «9999-12-31 23:59:59». MySQL хранит поле типа DATETIME в виде «YYYY-MM-DD HH:MM:SS» (ГГГГ-ММ-ДД ЧЧ-ММ-СС).
TIMESTAMP	Дата и время	Диапазон от «1970-01-01 00:00:00» до, примерно, 2037 года. MySQL может хранить поле типа TIMESTAMP в видах «YYYYMMDDHHMMSS» (TIMESTAMP(14)), «YYMMDDHHMMSS» (TIMESTAMP(12)), «YYYYMMDD» (TIMESTAMP(8)) и др.
TIME	Время	Диапазон от «-838:59:59» до «838:59:59». MySQL хранит поле TIME в виде «HH:MM:SS», но позволяет присваивать значения столбцам TIME с использованием либо строки или числа.
YEAR	Год в 2- или 4-хцифровом виде (4 цифры по-умолчанию)	Если вы используете 4 цифра, то допустимые значения 1901-2155, и 0000. Если 2 цифры, то 1970-2069 (70-69). MySQL хранит значения поля YEAR в формате «YYYY».

Окончание табл. 1

Тип данных	Использование	Диапазоны
CHAR	Строка фиксированной длины, которая справа дополняются пробелами до указанной длины, при хранении	Диапазон длины от 1 до 255 символов. Завершающие пробелы удаляются, когда значение извлекается. Значения CHAR сортируются и сравниваются без учета регистра в зависимости от кодировки по умолчанию, если не установлен флаг BINARY.
VARCHAR	Строка переменной длины. Примечание: конечные пробелы удаляются при сохранении (в отличие от спецификации ANSI SQL).	Диапазон длины от 1 до 255 символов. Значения VARCHAR сортируются и сравниваются без учета регистра, если не установлен флаг BINARY.
TINYBLOB TINYTEXT		BLOB или TEXT с максимальной длиной 255 (2^8 - 1) символов.
BLOB TEXT		BLOB или TEXT с максимальной длиной 65535 (2^16 - 1) символов.
MEDIUMBLOB MEDIUMTEXT		BLOB или TEXT с максимальной длиной 16777215 (2^24 - 1) символов.
LONGBLOB LONGTEXT		BLOB или TEXT с максимальной длиной 4294967295 (2^32 - 1) символов.
ENUM	Перечисление	Строка-объект, который может принимать только одно значение, выбирается из списка значений «значение 1», «значение 2» или NULL. ENUM максимум может иметь 65535 различных значений.
SET	Набор	Строка-объект, который может принимать ноль и более значений, каждоу из которых должно быть выбрано из списка значений «значение 1», «значение 2», … Поле SET может иметь максимум 64 варианта значений.

2.3 Консольный клиент и его команды

Для управления СУБД MySQL существует строенный консольный клиент. Запустить его можно несколькими способами:

1. Выбрать в системном меню соответствующий ярлык (MySQL 5.7 CommandLineClient)
2. Ввести соответствующую команду в командной строке. Команда для запуска клиента выглядит следующим образом:

mysql -uroot –p

где опция –u задает имя пользователя, а опция –p задает пароль. Если пароль явным образом не указан, то после запуска клиента будет выведен запрос. Пароль для доступа к СУБД в нашем случае является user.

Таблица 2

Основные команды управления

Команда	**Описание**
showdatabases	Выводит список существующих баз данных
use<база данных>	Выбор базы данных
showtables from <базаданных>	Вывод список таблиц базы данных
desc<имя таблицы>	Вывод структуры таблицы

3. Порядок выполнения работы

3.1. Создание базы данных

В качестве примера создадим базу данных Purchases, состоящую из одной табл. 3.

В данной таблице содержаться как текстовые, так и цифровые данные. Определим типы данных для каждого из столбцов таблицы:

ID – Целочисленное значение, которое является ключевым полем таблицы. Данное поле так же должно быть уникально для каждой записи. Тип данных выбираем целочисленный SMALLINT. Так же необходимо указать флаг ключевого поля (PRIMARY KEY) и флаг указывающий на то что значение не может быть нулевым (NOT NULL). Так как значение данного поля должно быть уникальным для каждой записи, стоит указать флаг автоинкремента (AUTO_INCREMENT).

Наименование – текстовое поле, наименование товара. Т.к. для данного поля может быть различной, выберем тип данных VARCHAR. А также указываем флаг NOT NULL.

Поставщик – это так же текстовое значение и содержит информацию о поставщике товара. По своим параметрам данное поле аналогично предыдущему значении.

Таблица 3

Перечень оборудования

ID	Наимено-вание	Поставщик	Коли-чество	Сумма (тыс. руб)	Классифи-катор
1	Блок питания Mastech HY3005D-2	Фирма "Белконтмаш" 220092, РБ, г. Минск, ул. Одоевского 28	5	19550	31.10.50.400
2	Измеритель иммитанса Е7-20	ОАО «МНИПИ» 220113, РБ, г Минск, ул. Я. Коласа	3	18033,6	33.20.45
3	Вольтметр В7-77	ОАО «МНИПИ» 220113, РБ, г Минск, ул. Я. Коласа	7	44499	33.20.43.300
4	Генератор Г6-46	ОАО «МНИПИ» 220113, РБ, г Минск, ул. Я. Коласа	8	21168	31.62.13.500
5	Осциллограф с8-47/1	ОАО «МНИПИ» 220113, РБ, г Минск, ул. Я. Коласа	11	75071	29.43.20.230

Количество содержит целочисленное значение, отображающее количество единиц товара. Тип данных выберем целочисленный (TINYINT). А также указываем флаг NOT NULL.

Сумма – содержит вещественное значение, отображающие общую сумму позиции. Выбираем тип данных FLOAT. А так же устанавливаем фалг NOT NULL.

Классификатор – содержит код классификатора позиции. Тип данных выберем VARCHAR. А также указываем флаг NOT NULL.

После определения параметров каждого поля приступим к созданию базы данных. Запускаем консольный клиент MySQL.

После запуска на экране появится окно консольного клиента с приглашением. Рассмотрим на рис. 1.

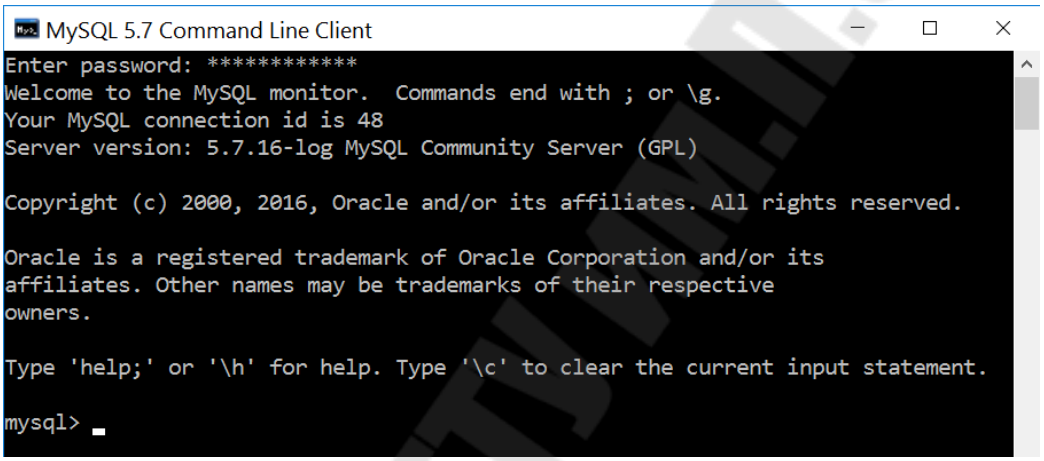


Рис. 1. Окно консольного клиента

Создание базы данных:

CREATEDATABASEdb_purchase;

После создания базы данных необходимо ее выбрать. Форматкоманды:

use db_purchase;

После проведения манипуляций с базой данных необходимо создать таблицу. Команда используется стандартная:

CREATE TABLE <имя таблицы> (поле 1 тип данных флаги, поле 2 тип данных флаги, …);

С учетом выбранных типов данных команда для создания нашей таблице выглядит следующим образом:

```
CREATE TABLE product (
id SMALLINT AUTO_INCREMENT NOT NULL PRIMARY KEY,
name VARCHAR(256) NOT NULL,
supplier VARCHAR(256) NOT NULL,
quantity TINYINT NOT NULL,
sum FLOAT NOT NULL,
classifierVARCHAR(15) NOTNULL
);
```

После предыдущей команды можно проверить создалась ли таблица. Для этого выполним мета команду showtables. Для просмотра структуры таблицы можно ввести мета команду desc<name_table>;.

Теперь необходимо занести данные в таблицу. Для вставки данных в таблицу используется команда insert. Формат команды:

```
INSERT INTO <название таблицы> ([<Имя столбца>, ... ])
VALUES (<Значение>,...);
```

Заполним данными нашу таблицу, для этого выполним следующую последовательность команд:

```
INSERT INTO product (name,supplier,quantity,sum,classifier)
VALUES ('БлокпитанияMastech HY3005D-2',
'Фирма "Белконтмаш" 220092, РБ, г. Минск, ул. Одоевского 28',
'5','19550','31.10.50.400');

INSERT INTO product (name,supplier,quantity,sum,classifier)
VALUES ('Измеритель имми-танса Е7-20',
'ОАО «МНИПИ» 220113, РБ, г Минск, ул. Я. Коласа',
'3','18033.6','33.20.45');

INSERT INTO product (name,supplier,quantity,sum,classifier)
VALUES ('Вольтметр В7-77',
'ОАО «МНИПИ» 220113, РБ, г Минск, ул. Я. Коласа',
'7','44499','33.20.43.300');
```

```
INSERT INTO product (name,supplier,quantity,sum,classifier)
VALUES ('Генератор Г6-46',
'ОАО «МНИПИ» 220113, РБ, г Минск, ул. Я. Коласа',
'8','21168','31.62.13.500');

INSERT INTO product (name,supplier,quantity,sum,classifier)
VALUES ('Осциллограф с8-47/1',
'ОАО «МНИПИ» 220113, РБ, г Минск, ул. Я. Коласа',
'11','75071','29.43.20.230');
```

Для просмотра содержимого таблицы можно вести команду выбора без условия:

```
SELECT * FROM product;
```

3.2. Задание для самостоятельного выполнения

Разработайте и создайте базу данных, содержащую список товаров с описанием и штрих-кодом (поля: id записи, наименование товара, описание товара, штрих-код).

4. Содержание отчета

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а также снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаний после выполнения работы.

5. Контрольные вопросы

1. Что такое MySQL?
2. Под какой лицензией распространяется?
3. Как создается база данных?
4. Какие целочисленные данные поддерживает?
5. Как реализуется функция автоинкремента поля?
6. Какие строковые типы данных поддерживаются

Лабораторная работа № 4

СУБД SQLite.
Связанные таблицы.

1 Цель работы

Изучить принципы работы со связанными таблицами в SQLite.

2 Основные теоретические сведения

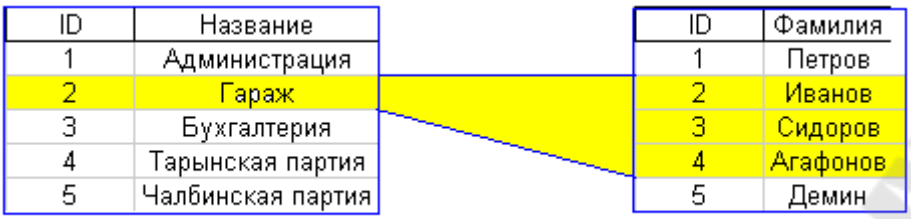
Связь работает путем сопоставления данных в ключевых столбцах; обычно это столбцы с одним и тем же именем в обеих таблицах. В большинстве случаев связь сопоставляет первичный ключ одной таблицы, являющийся уникальным идентификатором каждой строки этой таблицы, с записями внешнего ключа другой таблицы. Например продажи книг можно связать с названиями проданных книг и создать связь между столбцом title_id таблицы titles (первичный ключ) и столбцом title_id таблицы sales (внешний ключ).

Существует три типа связей между таблицами. Тип создаваемой связи зависит от того, как определены связанные столбцы.

1. Связи «один ко многим»
2. Связи «многие ко многим»
3. Связи «один к одному»

2.1. Связи «один ко многим»

Отношение «один ко многим» имеет место, когда одной записи родительской таблицы может соответствовать несколько записей дочерней. Связь «один ко многим» иногда называют связью «многие к одному». И в том, и в другом случае сущность связи между таблицами остается неизменной. Связь «один ко многим» является самой распространенной для реляционных баз данных. Она позволяет моделировать также иерархические структуры данных, приведена на рис. 1.



ID	Название
1	Администрация
2	Гараж
3	Бухгалтерия
4	Тарынская партия
5	Чалбинская партия

ID	Фамилия
1	Петров
2	Иванов
3	Сидоров
4	Агафонов
5	Демин

Рис. 1. Иерархические структуры данных

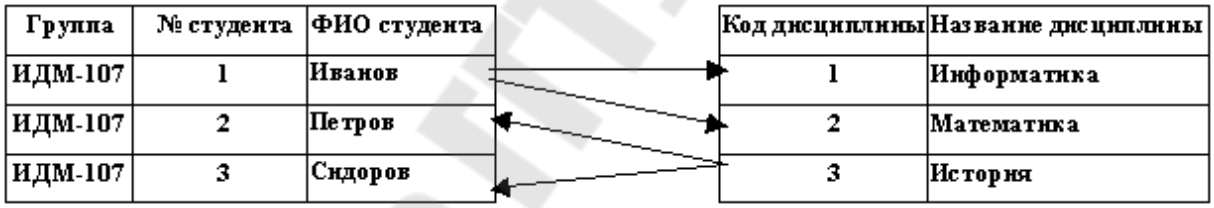
2.2. Связи «многие ко многим»

Отношение «многие–ко–многим» применяется в следующих случаях:

– одной записи в родительской таблице соответствует более одной записи в дочерней;

– одной записи в дочерней таблице соответствует более одной записи в родительской.

Всякую связь «многие–ко–многим» в реляционной базе данных необходимо заменить на связь «один–ко–многим» (одну или более) с помощью введения дополнительных таблиц. Рассмотрим на рис. 2.



Группа	№ студента	ФИО студента
ИДМ-107	1	Иванов
ИДМ-107	2	Петров
ИДМ-107	3	Сидоров

Код дисциплины	Название дисциплины
1	Информатика
2	Математика
3	История

Рис. 2. Связь «многие ко многим»

2.3. Связи «один к одному»

Отношение «один к одному» имеет место, когда одной записи в родительской таблице соответствует одна запись в дочерней. Это отношение встречается намного реже, чем отношение «один ко многим». Его используют, если не хотят, чтобы таблица БД «распухала» от второстепенной информации, однако для чтения связанной информации в нескольких таблицах приходится производить ряд операций чтения вместо одной, когда данные хранятся в одной таблице. Пример рассмотрим на рис. 3.

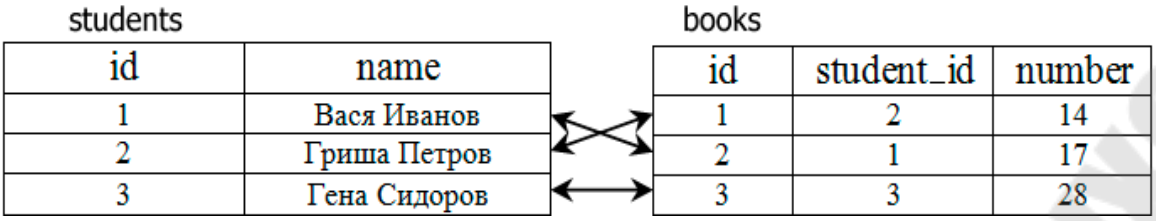


Рис. 3. Связь «один к одному»

2.4. Реализация связи таблиц в CEMLSQLite

Внешний ключ или FOREIGN KEY – это ограничение уровня таблицы в реляционных базах данных, в том числе и в базе данных SQLite3. Внешние ключи определяют правила, по которым будут связаны таблицы в базах данных SQLite. Но, кроме того, что внешние ключи определяют то, как будут связаны таблицы в базах данных SQLite3, они еще нужны для обеспечения целостности данных в базах данных.

В SQLite внешний ключ объявляется при помощи конструкции FOREIGN KEY, а таблица, на которую ссылается внешний ключ указывается после ключевого слово REFERENCE. Обратите внимание: указывается не только таблица, но и столбец, на который идет ссылка.

Ниже рассмотрим рис. 4, на котором показан синтаксис использования внешнего ключа в базах данных SQLite, вернее синтаксис конструкции REFERENCE.

Правила использования внешнего ключа не очень сложные, но давайте разберемся с тем, как реализован внешний ключ в SQLite3 и его конструкции: FOREIGEN KEY и REFEERENCE. Обратите внимание: когда вы связываете таблицы при помощи внешнего ключа одна таблица является родительской, а вторая таблица является дочерней. Внешний ключ всегда ссылается на родительскую таблиц, другими словами конструкция FOREIGN KEY и REFERENCE указывается в дочерней таблице.

Внешний ключ в базах данных SQLite необходим для реализации связей между таблицами. FOREIGEN KEY позволяет реализовывать связи между таблицами в базах данных. Конструкция REFERENCE используется для указания ссылки на родительскую таблицу. Внешний ключ обеспечивает целостность данных между двумя таблицами и необходим для нормализации базы данных.

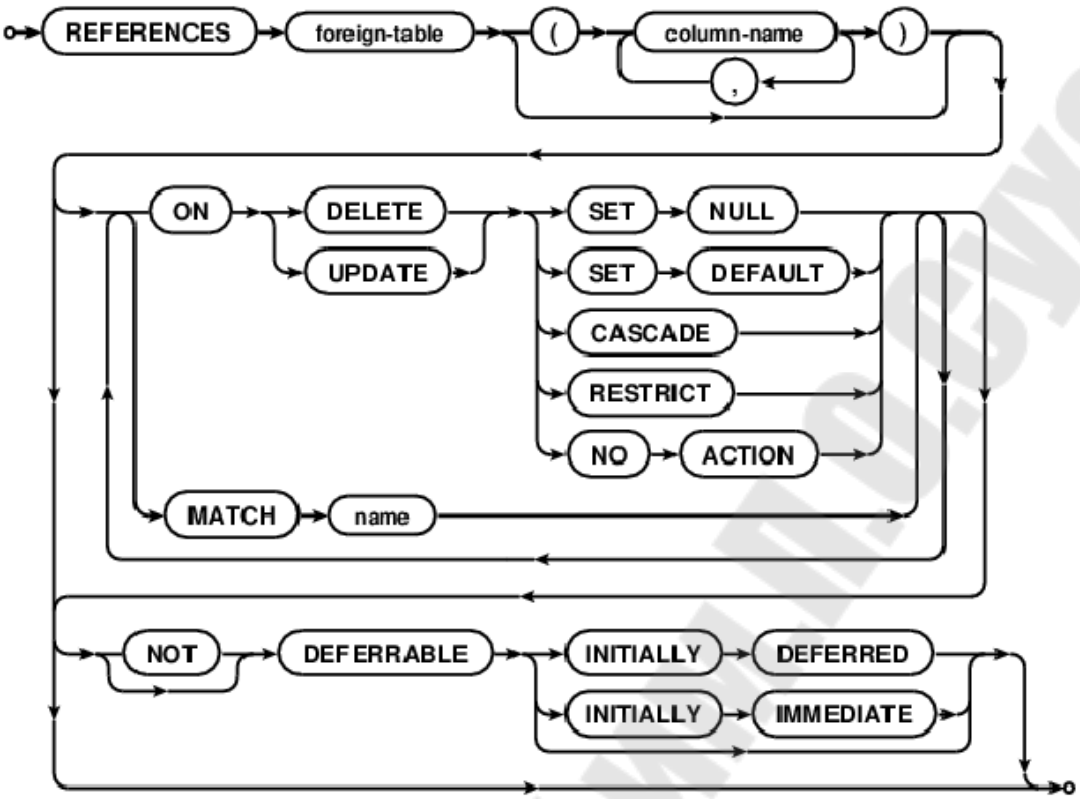


Рис. 4. Синтаксис использования внешнего ключа
в базах данных SQLite

Вторая нормальная форма и третья нормальная форма не могут быть реализованы без внешнего ключа. Вернее будет сказать: мы можем организовать связи между таблицами без внешнего ключа, но проверка правил при этом выполняться не будет.

3. Порядок выполнения работы

3.1. Реализация связи один ко многим в базах данных SQLite

Реализуем связь один ко многим при помощи внешнего ключа, для этого воспользуемся конструкциями FORIGEN KEY и REFERENCE. Будем связывать при помощи внешнего ключа две таблицы: таблицу авторов и таблицу книг. Один автор может написать много книг, но у книги может быть только один автор.

Чтобы реализовать связь один ко многим, нам нужно воспользоваться конструкцией FOREIGEN KEY и REFERENCE при создании таблицы при помощи команды CREATE.

```
PRAGMA foreign_keys=on;
CREATE TABLE books(
        Id INTEGER PRIMARY KEY,
        title TEXT NOT NULL,
        count_page INTEGER NOT NULL CHECK (count_page>0),
        price REAL CHECK (price >0),
        auth_id INTEGER NOT NULL,
        FOREIGN KEY (auth_id) REFERENCES auth(id)
);
CREATE TABLE auth(
        id INTEGER PRIMARY KEY,
        name TEXT NOT NULL,
        age INTEGER  CHECK (age >16)
);
```

Здесь нужно дать пояснение к том, как создан внешний ключ для базы данных. Во-первых, в SQLite3 по умолчанию отключена поддержка внешних ключей, команда PRAGMA позволяет включить внешние ключи в базах данных SQLite. Во-вторых, помимо внешнего ключа наши таблицы имеют ограничения уровня столбца. В-третих, столбец который ссылается и столбец, на который ссылается, должны иметь одинаковый тип данных, это закон реляционных баз данных.

Если столбцы будут иметь разные типы данных внешний ключ не будет работать. Скорее всего, другие СУБД вам даже не дадут возможность создать такой внешний ключ, но SQLite имеет динамическую типизацию данных, поэтому внешний ключ будет создан.

Конструкция FOREIGEN KEY объявляет о том, что столбец auth_id является ссылкой, а конструкция REFERENCE указывает, что столбец auth_id является ссылкой на столбец id таблицы auth. Таким нехитрым образом мы реализовали связь один ко многим в базе данных SQLite при помощи внешнего ключа.

Необходимо добавить данные в таблицу auth базы данных.

```
INSERT INTO auth (id, name, age)
VALUES (1, ‘ДжекЛондон’, 40);
INSERT INTO auth (id, name, age)
VALUES (2, ‘Лев Толстой’, 82);
```

Добавлять данные при создании внешнего ключа и использовании FOREIGN KEY удобнее сперва в ту таблицу, на которую идет ссылка.

Теперь необходимо добавить строки в таблицу books и указать значения для внешнего ключа.

INSERT INTO books (id, title, count_page, price, auth_id)
VALUES (1, ‘Белыйклык’, 287, 300.00, 1);

INSERT INTO books (id, title, count_page, price, auth_id)
VALUES (2, ‘Войнаимир’, 806, 780.00, 2);

INSERT INTO books (id, title, count_page, price, auth_id)
VALUES (3, ‘Дочьснегов’, 350, 370.00, 1);

INSERT INTO books (id, title, count_page, price, auth_id)
VALUES (4, ‘Игра’, 438, 210.00, 1);

INSERT INTO books (id, title, count_page, price, auth_id)
VALUES (5, ‘Сердцатрёх’, 505, 543.00, 1);

INSERT INTO books (id, title, count_page, price, auth_id)
VALUES (6, ‘АннаКаренина’, 900, 1100.00, 2);

INSERT INTO books (id, title, count_page, price, auth_id)
VALUES (7, ‘Крейцерова соната’, 206, 120.00, 2);

Теперь можно провести несколько опытов, которые покажут суть связи между таблицами.

В качестве первого опыта попробуем выполнить запрос:

INSERT INTO books (id, title, count_page, price, auth_id)
VALUES (8, ‘Луннаядолина’, 121, 160.00, 3);

Этот SQL запрос INSERT небудетвыполненв SQLite3, посколькудействуетограничениевнешнегоключа, ошибка: Error: FOREIGN KEY constraint failed. В таблице справочнике с авторами нет значения id = 3.

В качестве второго опыта попробуем изменить id с помощью команды UPDATE.

UPDATE books SET auth_id = 4 WHERE title = 'Белыйклык';

Выполнения данного запроса так же закончится ошибкой. Т.к. в таблице авторов нет записи с id = 4.

И в качестве третьего опыта попробуем произвести удаление записи автора. Для этого попробуем выполнить следующий запрос:

DELETE FROM auth WHERE name = 'Лев Толстой';

Однако при его выполнении так же будет ошибка. Т.к. нельзя удалить запись, на которую есть ссылки в других связанных таблица.

3.2. Реализация связи многие ко многим в базах данных SQLite3

Усложним предыдущий пример. Добавим условие, что одна книга может иметь несколько авторов.

Для реализации связи многие ко многим нам необходимо создавать третью таблицу, ее можно назвать результирующей, а можно назвать и промежуточной. В данном случае она будет называться auth_book.

```
PRAGMAforeign_keys=on;

CREATE TABLE books(
Id INTEGER PRIMARY KEY,
title TEXT NOT NULL,
count_page INTEGER NOT NULL CHECK (count_page>0),
price REAL CHECK (price >0)
);

CREATE TABLE auth(
id INTEGER PRIMARY KEY,
name TEXT NOT NULL,
age INTEGER  CHECK (age >16)
);

CREATE TABLE auth_book (
auth_id INTEGER NOT NULL,
books_id INTEGER NOT NULL,
FOREIGN KEY (auth_id) REFERENCES auth(id)
FOREIGN KEY (books_id) REFERENCES books(id)
```

);

Теперь заполним таблицы данными.
INSERT INTO books (id, title, count_page, price)
VALUES (1, ‘Белыйклык’, 287, 300.00);

INSERT INTO books (id, title, count_page, price)
VALUES (2, ‘Войнаимир’, 806, 780.00);

INSERT INTO auth (id, name, age)
VALUES (1, ‘ДжекЛондон’, 40);

INSERT INTO auth (id, name, age)
VALUES (2, ‘ЛевТолстой’, 82);

INSERT INTO books (id, title, count_page, price)
VALUES (3, ’12 стульев’, 516, 480.00);

INSERT INTO auth (id, name, age)
VALUES (3, ‘ИльяИльф’, 39);

INSERT INTO auth (id, name, age)
VALUES (4, ‘Евгений Петров’, 38);

Пока данные не связаны друг с другом, хоть и находятся в своих таблицах. Для того чтобы их связать необходимо заполнить 3-ю таблицу.

INSERTINTOauth_book (auth_id, books_id)
VALUES (1, 1);

INSERT INTO auth_book (auth_id, books_id)
VALUES (2, 2);

INSERT INTO auth_book (auth_id, books_id)
VALUES (3, 3);

INSERT INTO auth_book (auth_id, books_id)
VALUES (4, 3);

3.3. Задание для самостоятельного выполнения

Разработайте и создайте базу данных, содержащую список товаров с описанием и штрих-кодом, а так же категорией товара и поставщиком. Списки категорий товаров и поставщиков должны содержаться в отдельных таблицах и быть связаны с таблицей товаров.

4. Содержание отчета

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а также снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаний после выполнения работы.

5. Контрольные вопросы

1. Что такое связи таблиц и для чего они нужны?
2. Как осуществляется связь таблиц?
3. В чем особенность связи один к одному?
4. В чем особенность связи один ко многим?
5. В чем особенность связи многие ко многим?
6. С помощью какой команды осуществляется связь таблиц?

Лабораторная работа № 5

СУБД MySQL.
Связанные таблицы

1. Цель работы

Изучить принципы работы со связанными таблицами в MySQL.

2. Основные теоретические сведения

Реляционная база данных существует в виде таблиц, имеющих свои имена. На пересечении каждого столбца и каждой строки располагается одно значение. Рассмотрим таблицу 5.1, содержащую сведения о клиентах компании.

Таблица 1

Customers (Клиенты)

Id (идентификатор)	**name (имя)**	**phone (телефон)**	**address (адрес)**	**rationg (рэйтинг)**
533	ООО «Кускус»	313-48-48	пр. Независимости	1000
534	Петров	516-11-00	ул. Советская	1500
536	Крылов	41-13-48	ул. Сверидова	1000

Строки табл. 1 могут храниться в произвольной последовательности и не должны повторяться. Каждый столбец таблицы имеет имя и тип данных, которому соответствуют все значения в столбце. Так, в нашем примере столбцы с именами id и rating – числовые, а с именами name, phone и address – символьные. По существу, таблица реляционной базы данных представляет собой набор информации об однотипных объектах. При этом каждая строка содержит сведения об одном объекте, а каждый столбец – значения некоторого атрибута этих объектов. Например, строка с идентификационным номером 533 содержит информацию об объекте, у которого атрибут name (имя) имеет значение ООО «Кускус», атрибут phone (телефон) – значение 313-48-48 и т. д.

2.1. Первичный ключ

Строки таблицы неупорядочены и не имеют номеров, поэтому различить их можно только по содержащимся значениям. В связи с этим возникает необходимость рассмотреть понятие первичного ключа (primarykey). Первичный ключ – это минимальный набор столбцов, совокупность значений которых однозначно определяет строку. Это означает, что в таблице не должно быть строк, у которых значения во всех столбцах первичного ключа совпадают, при этом ни один столбец нельзя исключить из первичного ключа, иначе это условие нарушится. На практике первичным ключом служит специальный столбец, значения которого автоматически задает СУБД. Например, в таблице Customers (Клиенты) (см. табл. 1) это столбец id (идентификатор). Использовать такой искусственный первичный ключ значительно проще, чем естественный (основанный на атрибутах объекта). Например, в таблице Customers столбец name (имя) не может быть первичным ключом, так как имена клиентов могут совпадать; а первичный ключ из столбцов name (имя) и phone (телефон) был бы слишком громоздким. Дополнительными преимуществами искусственного ключа являются гарантированная уникальность значений (ее обеспечивает СУБД), постоянство значений (может меняться значение атрибута, но не значение искусственного ключа), а также числовой тип данных (поиск по числовым значениям выполняется намного быстрее, чем по символьным). Еще одна функция первичного ключа – организация связей между таблицами.

Существует три типа связей, устанавливаемых между таблицами в базе данных. • Связь «один ко многим». Этот тип связи используется чаще всего. В этом случае одна или несколько строк таблицы A ссылаются на одну из строк таблицы B. Для установки связи между таблицами в дочернюю таблицу добавляется внешний ключ (foreignkey) – один или несколько столбцов, содержащих значения первичного ключа родительской таблицы (иными словами, во внешнем ключе хранятся ссылки на строки родительской таблицы). Рассмотрим табл. 2, которая содержит сведения о заказах, сделанных клиентами, и является дочерней по отношению к таблицеCustomers (Клиенты).

Таблица 2

Orders (Заказы)

Id (идентифи-катор)	date (дата)	product_id (товар)	qty (количество)	amount (сумма)	customer_id (кдиент)
1012	12.12.2019	5	8	4500	533
1013	12.12.2019	2	14	22000	536
1014	21.01.2020	5	12	5750	533

В табл. 2 Orders внешним ключом является столбец customer_id (клиент), в котором содержатся номера клиентов из таблицы Customers (Клиенты). Таким образом, каждая строка таблицы Orders ссылается на одну из строк табл. 1 Customers. Например, строка с идентификационным номером 1012 содержит в столбце customer_id (клиент) значение 533: это означает, что заказ № 1012 сделан клиентом ООО «Кускус».

Столбец product_id таблицы Orders также является внешним ключом – он содержит номера товаров из столбца id (идентификатор) таблицы Products (Товары). Таким образом, таблица Orders является дочерней по отношению к таблицам Customers и Products.

Связь «один к одному». Такая связь между таблицами означает, что каждой строке одной таблицы соответствует одна строка другой таблицы, и наоборот. Например, если требуется хранить паспортные данные клиентов, можно создать таблицу Passports (Паспорта), связанную отношением «один к одному» с таблицей Customers (Клиенты). Таблицы, соединенные связью «один к одному», можно объединить в одну. Две таблицы вместо одной используют по соображениям конфиденциальности (например, можно ограничить доступ пользователей к таблице Passports), для удобства (если в единой таблице слишком много столбцов), для экономии дискового пространства (в дополнительную таблицу выносят те столбцы, которые часто бывают пустыми, тогда дополнительная таблица содержит значительно меньше строк, чем основная, и обе они занимают меньше места, чем единая таблица). Связь «один к одному» может быть организована так же, как связь «один ко многим», – с помощью первичного ключа родительской таблицы и внешнего ключа дочерней. Другой вариант – связь посредством первичных ключей обеих таблиц, при этом связанные строки имеют одинаковое значение первичного ключа.

Связь «многие ко многим». Этот тип связи в реляционной базе данных реализуется только с помощью вспомогательной таблицы. Например, если потребуется включить в заказ несколько наименований товаров, связь «многие ко многим» между таблицами Orders (Заказы) и Products (Товары) можно организовать с помощью вспомогательной таблицы Items (Позиции заказа), содержащей столбцы product_ id (номер товара из таблицы Products), qty (количество товаров данного наименования в заказе) и order_id (номер заказа из таблицы Orders). При этом столбцы product_id и qty из таблицы Orders исключаются. Таким образом, таблица Items будет дочерней по отношению к таблицам Orders и Products и каждая строка таблицы Items будет соответствовать одному наименованию товара в заказе. Как видим, реляционная база данных представляет собой весьма запутанную структуру, в которой все части (то есть записи) ссылаются на другие самым произвольным образом. А раз структура сложная, то неизбежны ее нарушения, происходящие по различным причинам, включая сбои программы, ошибки оператора и др. Последствия такого нарушения могут быть просто катастрофическими: скажем, что будет, если таблица заказов будет неверно ссылаться на таблицу товаров? Вся деятельность фирмы будет дезорганизована – вместо заказанного товара, допустим лопат, заказчику доставят топоры, а то и вовсе ничего, если ссылка на заказанный товар указывает на несуществующую строку таблицы товаров. Итак, важнейшим понятием теории реляционных баз данных является целостность данных.

2.2. Связи между таблицами. Внешний ключ

Реляционная база данных – это не просто набор таблиц. Объединить разрозненные фрагменты информации в единую структуру данных позволяют связи между таблицами, посредством которых строка одной таблицы сопоставляется строке (строкам) другой таблицы. Благодаря связям можно извлекать информацию одновременно из нескольких таблиц (например, выводить с помощью одного запроса и сведения о клиенте, и сведения о его заказах), избегать дублирования информации (не требуется в каждом заказе хранить адрес клиента), поддерживать полноту информации (не хранить сведения о заказанном товаре, если в базе данных отсутствует его описание) и многое другое.

Рассмотрим на примере, что такое связь между таблицами. Допустим, у нас есть таблицы A и B, и мы хотим их связать. Для этого в каждую строку таблицы A мы должны поместить некую информацию, позволяющую идентифицировать связанную с ней строку таблицы B. Эта информация называется ссылкой, а поля таблицы A, содержащие эту ссылку, – внешними ключами. Наверное, вы уже сами догадались, что в качестве ссылки используется первичный ключ таблицы B, поскольку именно его значения позволят однозначно идентифицировать нужную строку таблицы B. После того как мы во все строки таблицы A поместим ссылки на строки таблицы B, эти таблицы будут связаны. При этом таблица A будет называться дочерней, а таблица B – родительской.

2.3. Целостность данных

Целостностью данных, хранимых в СУБД, называется их корректность и непротиворечивость. Базовыми требованиями целостности, которые должны выполняться в любой реляционной базе данных, являются целостность сущностей и целостность связей (ссылочная целостность). Целостность сущностей означает, что в каждой таблице есть первичный ключ – уникальный идентификатор строки. Первичный ключ не должен содержать повторяющихся и неопределенных значений. Например, если в таблицу Customers (Клиенты) добавить еще одну строку с идентификатором 533 (притом что одна строка с таким идентификатором уже существует в таблице), то целостность сущностей будет нарушена и невозможно будет определить, кому из этих двух клиентов с одинаковыми идентификаторами принадлежат заказы №№ 1012 и 1014. Целостность связей означает, что внешний ключ в дочерней таблице не содержит значения, отсутствующие в первичном ключе родительской таблицы. Иными словами, строка дочерней таблицы не должна ссылаться на несуществующую строку родительской таблицы. В отличие от первичного, внешний ключ может содержать неопределенные значения (NULL), и в этом случае целостность не нарушится. Например, в таблицу Orders (Заказы) добавлена строка, содержащая в столбце customer_id значение 999. Здесь нарушится целостность связи между таблицами Customers и Orders: с одной стороны, заказ не является «ничьим», так как в этом случае в столбце customer_id было бы установлено значение NULL, с другой стороны,

невозможно выяснить имя и адрес клиента, сделавшего этот заказ. Как видно из приведенных примеров, если целостность данных нарушена, то с ними невозможно нормально работать. Поэтому поддержание целостности данных является одной из основных функций любой СУБД. Для поддержания целостности сущностей СУБД проверяет корректность значения первичного ключа при добавлении и изменении строк. Механизм поддержания ссылочной целостности более сложный. Помимо проверки корректности значения внешнего ключа при добавлении и изменении строк дочерней таблицы, необходимо также предотвратить нарушение ссылочной целостности при удалении и изменении строк родительской таблицы. Для этого существует несколько способов.

1. Запрет (RESTRICT): если на строку родительской таблицы ссылается хотя бы одна строка дочерней таблицы, то удаление родительской строки и изменение значения первичного ключа в такой строке запрещаются. Например, не допускается удаление информации о клиенте из таблицы Customers (Клиенты), если у этого клиента есть зарегистрированные заказы, то есть строки в таблице Orders (Заказы), которые ссылаются на строку со сведениями об этом клиенте.

2. Каскадное удаление/обновление (CASCADE): при удалении строки из родительской таблицы автоматически удаляются все ссылающиеся на нее строки дочерней таблицы; при изменении значения первичного ключа в строке родительской таблицы автоматически обновляется значение внешнего ключа в ссылающихся на нее строках дочерней таблицы. Например, при удалении записи о клиенте из таблицы Customers (Клиенты) автоматически удаляются сведения о заказах этого клиента, то есть соответствующие строки в таблице Orders (Заказы).

3. Обнуление (SET NULL): при удалении строки и при изменении значения первичного ключа в строке значение внешнего ключа во всех строках, ссылающихся на данную, автоматически становится неопределенным (NULL). Например, при удалении записи о клиенте из таблицы Customers (Клиенты) заказы этого клиента автоматически становятся «ничьими», то есть в соответствующих строках таблицы Orders (Заказы) в столбце customer_id (клиент) устанавливается значение NULL.

В СУБД MySQL способ поддержания целостности связи указывается при создании или изменении структуры дочерней

таблицы. С понятием целостности данных тесно связано понятие транзакции. Транзакцией называется группа связанных операций, которые должны быть либо все выполнены, либо все отменены. Если при выполнении одной из операций происходит ошибка или сбой, то транзакция отменяется. При этом все уже внесенные другими операциями изменения автоматически аннулируются и восстанавливается исходное состояние базы данных. Важнейшее применение транзакций – это объединение тех операций, которые, будучи выполнены по отдельности, могут нарушить целостность данных. Например, рассмотренная выше операция каскадного удаления выполняется как единая транзакция: строка родительской таблицы должна быть удалена вместе со всеми ссылающимися на нее строками дочерней таблицы, а если по каким-либо причинам одну из этих строк удалить невозможно, то не будет удалена ни одна из строк.

3. Порядок выполнения работы

3.1. Реализация связи многие ко многим в СУБД MySQL

Реализуем связь многие ко многим при помощи внешнего ключа и дополнительной таблицы. Суть задачи: Участники подают заявки на участие в номинациях в каком-то онлайн конкурсе -- один участник может участвовать в любом числе номинаций (их много) и самих участников тоже может быть сколько угодно (тоже много).

Все манипуляции будут производиться в тонком клиенте mysql. Для начала необходимо создать базу данных db_competition:

```
CREATE DATABASE IF NOT EXISTS db_competition;
```

Для удобства дальнейшей работы стоит установить данную базу данных по умолчанию:

```
USE db_competition;
```

После этого можно переходить к разработке структуры самой базы данных. В данном случае база данных будет содержать три таблицы:

1. Таблицу "Заявка" (tbl_tickets)
2. Таблицу "Номинация" (tbl_nominations)
3. и так называемую. "таблицусвязи" (tbl_tickets_nominations)

Для создания необходимых таблиц нужно создать соотвествующий запрос на языке SQL:

```
CREATE TABLE `tbl_tickets` (
  `ticketID` INT(11) NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(255) NOT NULL
    DEFAULT '' COMMENT
'Имяучастника/названиеорганизации',
  `info` VARCHAR(255) NULL
DEFAULT '' COMMENT 'Информацияономинанте',
PRIMARY KEY (`ticketID`)
)
COMMENT='Заявкиучастниковконкурса'
ENGINE=InnoDB
;

CREATE TABLE `tbl_nominations` (
  `nominationID` INT(11) NOT NULL AUTO_INCREMENT,
  `title` VARCHAR(255) NULL
    DEFAULT NULL COMMENT 'Названиеноминации',
  PRIMARY KEY (`nominationID`)
)
COMMENT='Номинацииконкурса'
ENGINE=InnoDB
;

CREATE TABLE `tbl_tickets_nominations` (
  `ticket_id` INT(11) NOT NULL,
  `nomination_id` INT(11) NOT NULL,
  PRIMARY KEY (`ticket_id`, `nomination_id`),
  INDEX `ticket_id` (`ticket_id`),
  INDEX `nomination_id` (`nomination_id`),
  CONSTRAINT `FK_Nominations` FOREIGN KEY
(`nomination_id`)
    REFERENCES `tbl_nominations` (`nominationID`) ON
DELETE CASCADE,
  CONSTRAINT `FK_Ticket` FOREIGN KEY (`ticket_id`)
    REFERENCES `tbl_tickets` (`ticketID`) ON DELETE
CASCADE
)
COMMENT='Таблица связи заявок участников и номинаций
конкурса'
```

```
ENGINE=InnoDB
;
```

После создания таблиц их необходимо заполнить данными, для этого необходимо выполнить следующие три запроса на языке SQL:

```
INSERT INTO `tbl_tickets` VALUES
(1, 'Василий Иванов', 'Преподаватель математики и информатики в средней школе №123'),
(2, 'Анна Сергеева', 'Инженер-программист в очень известной IT-фирме'),
(3, 'Программирование для всех', 'Некоммерческая образовательная организация'),
(4, 'Юный программист', 'Кружок для детей в д. Простоквашино'),
(5, 'IT FOR FREE', 'Русскоязычное IT-сообщество с уклоном в web'),
(6, 'Саша Петров', 'Студент 2 курса, автор пособия по SQL')
;

INSERT INTO `tbl_nominations` VALUES
(1, 'Дополнительноеобразование'),
(2, 'Открытый исходный код (лучший проект open-source)'),
(3, 'Детское образование'),
(4, 'Лучшее пособие'),
(5, 'Новый алгоритм (научно-инженерный вклад)')
;

INSERT INTO `tbl_tickets_nominations` VALUES
(1, 3),
(2, 5),
(3, 1),
(3, 2),
(3, 3),
(3, 4),
(3, 5),
(4, 3),
(4, 4),
(5, 1),
(5, 2),
```

```
(5, 4),
(6, 4)
;
```

3.2. Извлечение данных для связи "многие ко многим" (SELECT)

Рассмотрим задачу извлечения участников, связанных с данной номинацией -- или короче "номинации, и всех, кто подал в неё заявки" (алгоритм извлечения данных в обратную сторону -- т.е. "участик и все его номинации" абсолютно аналогичен).

На практике приходится сталкиваться с двумя базовыми ситуациями:

1. Извлечение одной сущности номинации и связанных с ней участников
2. Извлечение списка сущностей номинаций и связанных с каждой из номинаций участников (т.е. фактически список участников для каждого элемента из списка номинаций).

Пусть у нас известен id () номинации и мы хотим получить сведения об этой номинации и всех участниках в ней.

Во-первых, сделать это можно двумя sql запросами:

Сначала просто получим кортеж этой номинации:

```
SELECT * FROM tbl_nominations WHERE nominationID=4;
```

В табл. 1 рассмотрим результат выполнения запроса

Таблица 3

Результат выполнения запроса

nominationID	**title**
4	Лучшее пособие

После, опять же зная id номинации (используем в WHERE), достаточно просто сделать LEFT JOIN между таблицей связи и таблицей участников:

```
SELECT * FROM  tbl_tickets_nominations LEFT JOIN tbl_tickets
   ON ticket_id = ticketID
   WHERE tbl_tickets_nominations.nomination_id = 4;
```

В табл. 4 рассмотрим результат выполнения запросам

Таблица 4

Результат выполнения запроса

ticket_id	nomination_id	ticketID	name	info
3	4	3	Программирование для всех	Некоммерческая образовательная организаци
4	4	4	Юный программист	Кружок для детей в д. Простоквашино
5	4	5	IT FOR FREE	Русскоязычное IT-сообщество с уклоном в web
6	4	6	Саша Петров	Студент 2 курса, автор пособия по SQL

Как видим, тут мы получили вообще все колонки (т.к. в запросе указали звездочку *) двух соединённых таблиц (связи и заявок). Также видим что на номинации с id=4 номинировалось 4-ре участника, кроме их имен видны также и описания. Все эти данные можно использовать в приложении, после выполнения запроса к БД – например, записать то, что нужно в поле, хранящее массив объекта конкретной номинации.

Если требуется от массива связанных сущностей только одно поле (напр. имена участников), то решить задачу можно вообще одним sql запросом, используя группировку (GROUP BY) и применимую к группируемым значения колонки функцию конкатенации GROUP_CONCAT():

```
SELECT
tbl_nominations.*,
   GROUP_CONCAT(tbl_tickets.name SEPARATOR ', ') as participants_names
 FROM
tbl_nominations LEFT JOIN tbl_tickets_nominations
     ON tbl_nominations.nominationID = tbl_tickets_nominations.nomination_id
   LEFT JOIN tbl_tickets
     ON tbl_tickets.ticketID = tbl_tickets_nominations.ticket_id

WHERE tbl_tickets_nominations.nomination_id = 4
```

```
GROUP BY tbl_nominations.nominationID;
```

Получим единственный кортеж:
4 | Лучшее пособие| Программирование для всех, Юный программист, IT FOR FREE, Саша Петров

3.3. Задание для самостоятельного выполнения

Разработать базу данных по учету поставок и продаж компьютерной техники в СУБД MySQL. В базе данных должен присутствовать справочники категорий товара, производителей, поставщиков. В Базе данных должны применятся связи один ко многим и многие ко многим.

4. Содержание отчета

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а также снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаний после выполнения работы.

5. Контрольные вопросы

1. Что такое связи таблиц и для чего они нужны?
2. Как осуществляется связь таблиц?
3. В чем особенность связи один к одному?
4. В чем особенность связи один ко многим?
5. В чем особенность связи многие ко многим?
6. С помощью какой команды осуществляется связь таблиц?

Лабораторная работа № 6

Разработка пояснительной записки к проекту информационной системы

Цель: Научиться разрабатывать пояснительную записку к проекту ИС.

Задание: Разработать пояснительную записку к проекту ИС по индивидуальному варианту.

Оформление и содержание пояснительной записки должно соответствовать требованиям стандарта «ГОСТ 19.404-79. ЕСПД. Пояснительная записка. Требования к содержанию и оформлению» и приложенного к заданию примера.

Порядок сдачи лабораторной работы: Предоставить отчёт, содержащий пояснительную записку к проекту ИС фирмы / организации (по индивидуальному варианту) для внедрения в фирме / организации информационной системы (см. пример на программу очистки оперативной памяти).

- По каждой лабораторной работе оформляется и сдается отчет преподавателю в электронном и бумажном виде.
- Работа выполняется самостоятельно в произвольное время и сдается в строго оговоренные сроки только в лаборатории в часы занятий.
- Выполнение лабораторной работы предполагает достаточно подробное изучение и правдоподобное отражение предметной области.

ПРИМЕР ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

Программа очистки оперативной памяти

Пояснительная записка

АННОТАЦИЯ

В данном программном документе приведена пояснительная записка к программе «Mem.exe», предназначенной для очистки и дефрагментации оперативной памяти ПК через заданные интервалы времени.

В разделе «Введение» указано наименование программы и условное обозначение темы разработки.

В разделе «Назначение и область применения» указано назначение программы и краткая характеристика области применения программы.

В разделе «Технические характеристики» содержатся следующие подразделы:

- постановка задачи на разработку программы с описанием применяемых математических методов, допущений и ограничений, связанных с выбранным математическим материалом;
- описание алгоритма и функционирования программы с обоснованием выбора схемы алгоритма решения задачи и возможное взаимодействие программы с другими программами;
- описание и обоснование выбора метода организации входных и выходных данных;
- описание и обоснование выбора состава технических и программных средств на основании проведенных расчетов и анализов.

В разделе «Ожидаемые технико-экономические показатели» указаны технико-экономические показатели, обосновывающие выбранный вариант технического решения, а также ожидаемые оперативные показатели.

В разделе «Источники, использованные при разработке» указан перечень научно-технических публикаций, нормативно-технических документов и других научно-технических материалов, на которые есть ссылки в основном тексте.

Оформление программного документа «Руководство оператора» произведено по требованиям ЕСПД.

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Наименование программы

Наименование – «Программа очистки оперативной памяти».

1.2. Условное обозначение темы разработки

Наименование темы разработки – «Разработка программы очистки оперативной памяти».

Условное обозначение темы разработки (шифр темы) – «А.В.00001»

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

2.1. Назначение программы

Основное назначение программы очистки оперативной памяти «Mem.exe» –повысить производительность системы.

Данная программа позволяет поддерживать бесперебойную работу ПК длительное время, предотвращать утечки памяти, засорение оперативной памяти неиспользуемыми DLL и программами, а также в итоге предотвращать зависание ПК.

2.2. Область применения программы

Программа предназначена к применению в профильных подразделениях АСУ ТП, на объектах, для автоматизации которых используется SCADa система Genesis32

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи на разработку программы

После многократного открытия и закрытия программ, записи в базы данных, считываний из них, а также в процессе работы SCADa-системы и ОС оперативная память значительно фрагментируется. В итоге команды будут обрабатываться на порядок медленнее. У процессора в кэш (L1 и L2) записывается определенная часть оперативной памяти еще до того, как будет произведено обращение к этой области памяти. Когда же настает черед «прокэшированных» команд или данных, то они читаются процессором из КЭШа, что во много раз быстрее, чем обращение к оперативной памяти. Если запущенная программа и ее данные оказались фрагментированными в памяти, происходит ситуация, когда в КЭШ попадают совсем не те данные, которые нужны, и процессору ничего не остается, как отыскивать нужное в оперативной памяти.

Подобное снижение hit rate КЭШа (число попаданий) способно снизить производительность независимо от объема оперативной памяти.

Многие программы, в том числе и сама ОС, загружают в память множество библиотек (DLL), и не факт, что эти библиотеки будут обязательно использоваться. Поэтому необходимо подобные DLL выгружать в своп-файл. Если они вдруг понадобятся, то незамедлительно будут загружены ОС назад.

Действия по оптимизации и дефрагментации оперативной памяти производят программы сторонних разработчиков, такие как «FreeMemory» и «CoolMEM». В ходе их тестирования было установлено, что данные программы (в режиме постоянного мониторинга ОЗУ и очистки памяти при превышении заданных пределов) значительно загружают операционную систему и в некоторых случаях приводят к зависанию ПК. В ходе изысканий было установлено, что при использовании программы «FreeMemory» (версии 1.7), при запуске её командной строкой с параметрами «C A» (C – очистка памяти, A – очистить всю память), после выполнения всех процедур по очистке и дефрагментации ОЗУ данная программа полностью выгружается из памяти компьютера. При этом зависаний ПК не происходит и не используются лишние ресурсы компьютера. Использование этих данных позволило разработать программу «Mem.exe», которая каждый час командной строкой с параметрами «C A» загружает программу «FreeMemory» для очистки ОЗУ.

Программа «Mem.exe» работает под управлением ОС Windows 98 или Windows Me.

3.2. Описание алгоритма и функционирования программы

3.2.1. Описание алгоритма программы

Описание алгоритма программы приведено в п. «Алгоритм программы» программного документа «Описание применения».

3.2.2.Описание функционирования программы

Основной функцией программы «Mem.exe» является вызов из каталога C:\Program Files\FreeMemory\ программы стороннего разработчика FreeMemory.exe с параметрами командной строки «C A» (C – очистка памяти,A – очистить всю память). Вызов программы

производится по таймеру, каждый час, в XX.15.00 (в 15 минут каждого часа). Дополнительно программа «Mem.exe» проверяет наличие по указанному пути (C:\Program Files\FreeMemory\) исполняемого модуля программы «FreeMemory.exe». В случае его отсутствия выдается сообщение «Файл FreeMemory.exe не найден, переустановите программу».

Основная задача вызываемой программы «FreeMemory» – повысить про- изводительность системы.

Программа «FreeMemory» реализует следующие функции:

- Очистка и дефрагментация оперативной памяти;
- Выгрузка ненужных DLL;
- Очистка КЭШа.

Данные функции программы «FreeMemory» позволяют поддерживать бесперебойную работу ПК длительное время, предотвращать утечку памяти, засорение оперативной памяти неиспользуемыми DLL и программами, а в итоге – предотвращать зависание ПК.

3.3.3. Возможные взаимодействия программы с другими программами

Программа «Mem.exe» не предназначена для самостоятельной очистки и дефрагментации оперативной памяти, она только вызывает программу

«FreeMemory», поэтому для ее функционирования необходима предустановленная в каталог «C:\Program Files\FreeMemory\» программа «FreeMemory.exe» (версии 1.7).

Программа «FreeMemory» должна быть установлена в каталог C:\Program Files\FreeMemory\.

Для установки данной программы достаточно скопировать перечисленные ниже файлы в указанную папку на компьютере пользователя. Каких-либо настроек после копирования программы «FreeMemory» не требуется.

Список необходимых файлов программы «FreeMemory»:

- FreeMemory.exe 36 352 байт
- FreeMemory.hlp 46 965 байт
- FreeMemory.cnt 813 байт
- File_id.diz 629 байт
- Каталог \Plugin\ Setup\ Setup.dll 12 800 байт

3.3. Описание и обоснование выбора метода организации входных ивыходных данных

3.3.1. Описание и обоснование выбора метода организации входных данных

Программа «Mem.exe» в ходе своей работы не использует никаких входных данных

3.3.2. Описание и обоснование выбора метода организации выходных данных

Программа «Mem.exe» в ходе своей работы не создает никаких выходных данных.

3.4. Описание и обоснование выбора состава технических средств

Программа «Mem.exe» обладает низкими требованиями к аппаратной части ПК. Для непосредственной работы данной программы достаточно 1 Мб ОЗУ и дискового пространства. Но выполняемые программой «Mem.exe» функции требуют более 100 Мб свободного дискового пространства для сброса содержимого ОЗУ в файл подкачки. Исходя из низких требований программы к аппаратной части ПК, можно заключить, что данная программа будет работать на любом современном компьютере.

3.5. Описание и обоснование выбора состава программных средств

Программа очистки оперативной памяти «Mem.exe» предназначена для работы под управлением операционной системы Windows 98 или Windows Me, так как данные ОС в ходе работы значительно «замусоривают» ОЗУ неисполь- зуемыми данными и компонентами.

Программа «Mem.exe» не предназначена для работы под управлением ОС Windows 2000 и Windows XP, так как эти операционные системы имеют свои, встроенные, менеджеры памяти и программа FreeMemory неэффективна в данных операционных системах.

Программа очистки оперативной памяти «Mem.exe» не предназначена для самостоятельной очистки и дефрагментации оперативной памяти, она только вызывает программу «FreeMemory», поэтому для ее функционирования необходима предустановленная в каталог «C:\Program Files\FreeMemory\» программа «FreeMemory.exe» (версии 1.7).

4. Ожидаемые технико-экономические показатели

Программа очистки и дефрагментации оперативной памяти «Mem.exe» позволяет поддерживать бесперебойную работу ПК длительное время, предотвращать утечки памяти, засорение оперативной памяти неиспользуемыми DLL и программами, а также в итоге предотвращать зависание ПК. Данные функции программы позволяют снизить затраты времени на техническое обслуживание и обеспечить стабильную работу ПК, что в конечном итоге позволяет повысить производительность труда и наиболее полно использовать ресурсы компьютера.

Лабораторная работа № 7

РАЗРАБОТКА ТЕХНИЧЕСКОГО ЗАДАНИЯ НА ИНФОРМАЦИОННУЮ СИСТЕМУ

Цель: Научиться разрабатывать техническое задание на ИС.

Задание: Разработать техническое задание на ИС по индивидуальному варианту.

Оформление и содержание технического задания должно соответствовать требованиям стандарта «ГОСТ 19.201-78. ЕСПД. Техническое задание.

Порядок сдачи лабораторной работы: Представить отчёт, содержащий техническое задание на ИС фирмы/организации (по индивидуальному варианту) для внедрения в фирме/организации информационной системы (см. пример технического задания на «**Программу очистки оперативной памяти**»).

Пример технического задания.

Программа очистки оперативной памяти

Техническое задание

АННОТАЦИЯ

В данном программном документе приведено техническое задание на разработку программы очистки и дефрагментации оперативной памяти ПК через заданные интервалы времени.

В разделе «Введение» указано наименование, краткая характеристика области применения программы (программного изделия).

В разделе «Основания для разработки» указаны документы, на основании которых ведется разработка, наименование и условное обозначение темы разработки.

В разделе «Назначение разработки» указано функциональное и эксплуатационное назначение программы (программного изделия).

Раздел «Требования к программе» содержит следующие подразделы:

- требования к функциональным характеристикам;
- требования к надежности;
- условия эксплуатации;
- требования к составу и параметрам технических средств;
- требования к информационной и программной совместимости;
- специальные требования.

В разделе «Требования к программной документации» указаны предварительный состав программной документации и специальные требования к ней.

В разделе «Технико-экономические показатели» указаны: ориентировочная экономическая эффективность, предполагаемая годовая потребность, экономические преимущества разработки.

В разделе «Стадии и этапы разработки» установлены необходимые стадиираз работки, этапы и содержание работ.

В разделе «Порядок контроля и приемки» должны быть указаны виды испытаний и общие требования к приемке работы.

Оформление программного документа «Техническое задание» произведено по требованиям ЕСПД .

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Наименование программы

Наименование – «Программа очистки оперативной памяти».

1.2. Краткая характеристика области применения программы

Программа предназначена к применению в профильных подразделениях АСУ ТП, на автоматизируемых объектах ОАО «ХХХХ».

2. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

2.1. Основание для проведения разработки

Основанием для проведения разработки является необходимость ежечасно производить очистку и дефрагментацию оперативной памяти ПК для предотвращения замедления работы и повышения производительности системы.

2.2. Наименование и условное обозначение темы разработки

Наименование темы разработки – «Разработка программы очистки оперативной памяти».

Условное обозначение темы разработки (шифр темы) - «Х.Х.00001».

3. НАЗНАЧЕНИЕ РАЗРАБОТКИ

3.1. Функциональное назначение программы

3.2. Эксплуатационное назначение программы

4. ТРЕБОВАНИЯ К ПРОГРАММЕ

4.1. Требования к функциональным характеристикам

4.1.1. Требования к составу выполняемых функций

4.1.2. Требования к организации входных данных

Требования к организации входных данных не предъявляются.

4.1.3. Требования к организации выходных данных

Требования к организации выходных данных не предъявляются.

4.1.4. Требования к временным характеристикам

Требования к временным характеристикам программы не предъявляются.

4.2. Требования к надежности

4.2.1. ***Требования к обеспечению надежного (устойчивого) функционирования программы.***

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением совокупности организационно-технических мероприятий, перечень которых приведен ниже:

а) организацией бесперебойного питания технических средств;

б) регулярным выполнением рекомендаций Министерства труда и социального развития РБ,

в) регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие компьютерных вирусов;

г) необходимым уровнем квалификации сотрудников профильных подразделений.

4.2.2. ***Время восстановления после отказа***

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать времени, необходимого на перезагрузку операционной системы и запуск программы, при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого для устранения неисправностей технических средств и переустановки программных средств.

4.2.3. ***Отказы из-за некорректных действий оператора***

Отказы программы возможны вследствие некорректных действий оператора (пользователя) при взаимодействии с

операционной системой. Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу конечного пользователя без предоставления ему административных привилегий.

4.3. Условия эксплуатации

4.3.1. *Климатические условия эксплуатации*

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

4.3.2. *Требования к видам обслуживания*

См. Требования к обеспечению надежного (устойчивого) функционирования программы.

4.3.3. *Требования к численности и квалификации персонала*

Минимальное количество персонала, требуемого для работы программы, должно составлять не менее 2 штатных единиц – системный программист и конечный пользователь программы – оператор.

Системный программист должен иметь, как минимум, среднее техническое образование.

В перечень задач, выполняемых системным программистом, должны входить:

а) задача поддержания работоспособности технических средств;

б) задачи установки (инсталляции) и поддержания работоспособности системных программных средств – операционной системы;

в) задача установки (инсталляции) программы.

Конечный пользователь программы (оператор) должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

Персонал должен быть аттестован не ниже чем на II квалификационную группу по электробезопасности (для работы с конторским оборудованием).

4.4. Требования к составу и параметрам технических средств

В состав технических средств должен входить IBM-совместимый персональный компьютер (ПЭВМ), включающий в себя:

а) процессор Pentium – 4 с тактовой частотой не менее 1.2 ГГц;

б) оперативную память объемом не менее 128 Мб;

в) жесткий диск объемом 40 Гб и выше;

г) оптический манипулятор типа «мышь»;д) 2 COM-порта.

4.5. Требования к информационной и программной совместимости

4.5.1. *Требования к информационным структурам и методам решения*

Требования к информационным структурам на входе и выходе, а также кметодам решения не предъявляются.

4.5.2. *Требования к исходным кодам и языкам программирования*

Исходные коды программы должны быть реализованы на языке Visual Basic 6. В качестве интегрированной среды разработки программы должна быть использована среда Microsoft Visual Basic 6.0 (локализованная, русская версия).

4.5.3. *Требования к программным средствам, используемым программой*

Системные программные средства, используемые программой, должны быть представлены локализованной версией операционной системы Windows 98 или Windows Me.

4.5.4. *Требования к защите информации и программ*

Требования к защите информации и программ не предъявляются.

4.6. Специальные требования

Специальные требования к программе не предъявляются.

5. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

5.1. Предварительный состав программной документации

Состав программной документации:

1) техническое задание;

2) спецификация;
3) текст программы;
4) описание программы;
5) программа и методики испытаний;
6) пояснительная записка;
7) ведомость эксплуатационных документов;
8) формуляр;
9) описание применения;
10) руководство системного программиста;
11) руководство программиста;
12) руководство оператора.

5.2. Специальные требования к программной документации

Специальные требования к программной документации не предъявляются.

6. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

6.1. Ориентировочная экономическая эффективность

Ориентировочная экономическая эффективность не рассчитывается.

6.2. Предполагаемая годовая потребность

Предполагаемое число использования программы в год – круглосуточнаяработа программы на одном рабочем месте.

6.3. Экономические преимущества разработки

Экономические преимущества разработки не рассчитываются.

7. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

7.1. Стадии разработки

Разработка должна быть проведена в три стадии:
1) разработка технического задания;
2) рабочее проектирование;
3) внедрение.

7.2. Этапы разработки

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.

На стадии рабочего проектирования должны быть выполнены перечисленные ниже этапы работ:

1) разработка программы;

2) разработка программной документации;

3) испытания программы.

На стадии внедрения должен быть выполнен этап разработки - подготовкаи передача программы.

7.3. Содержание работ по этапам

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

1) постановка задачи;

2) определение и уточнение требований к техническим средствам;

3) определение требований к программе;

4) определение стадий, этапов и сроков разработки программы и документации на неё;

5) выбор языков программирования;

6) согласование и утверждение технического задания.

На этапе разработки программы должна быть выполнена работа по программированию и отладке программы.

На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями ГОСТ 19.101–77 и требованием пункта «Предварительный состав программной документации» настоящего технического задания.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

1) разработка, согласование и утверждение программы и методики испытаний;

2) проведение приемо-сдаточных испытаний;

3) корректировка программы и программной документации по результатамиспытаний.

На этапе подготовки и передачи программы должна быть выполнена работа по подготовке и передаче программы и программной документации в эксплуатацию.

7.4. Исполнители

Руководитель разработки
Начальник XXXX
Xxxxx
xxx X.X.Ответственный исполнитель
Начальник гр. РиВ АСУТП
Xxxxx
xxx X.X.Исполнитель
Вед. инженер XXXX Xxxxxxxx X.X.

8. ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

8.1. Виды испытаний

Приемо-сдаточные испытания программы должны проводиться согласно разработанной и согласованной «Программы и методики испытаний».

Ход проведения приемо-сдаточных испытаний документируется в Протоколе проведения испытаний.

8.2. Общие требования к приемке работы

После проведения испытаний в полном объеме, на основании «Протокола испытаний» утверждают «Свидетельство о приемке» и производят запись в программном документе «Формуляр».

Лабораторная работа № 8
ПОСТРОЕНИЕ МОДЕЛИ БИЗНЕС-ПРОЦЕССОВ ПРЕДПРИЯТИЯ

1. Цель работы

Научиться строить модель бизнес-процессов предприятия.

Задание:

1. Разработать модель бизнес-процессов обследуемого предприятия / организации / фирмы (заказчика), для которой разрабатывается вариант информационной системы. Определить основные, дополнительные, вспомогательные бизнес-процессы, а также бизнес-процесс управления.

2. Определить состав бизнес-функций по каждому бизнес-процессу. Описать работы, выполняемые в рамках каждой бизнес-функции.

3. Определить штат сотрудников для выполнения описанного в пункте 2 состава бизнес-функций. Описать: кто, на каком рабочем месте выполняет перечисленные в пункте 2 работы. Построить матрицу ответственности. По матрице ответственности составить штатное расписание.

4. Построить структуру программного обеспечения проектируемой информационной системы. Уровень детализации: одно рабочее место – один функциональный программный модуль информационной системы.

2. Порядок сдачи лабораторной работы

Представить отчёт, содержащий модель бизнес-процессов предприятия / организации / фирмы (по индивидуальному варианту) для разработки Информационной системы.

Дополнительные требования к отчёту:

Отчет должен содержать следующую информацию:

- описание процесса построения бизнес-модели и представление модели бизнес-процессов на рисунке;
- состав бизнес-функций (и выполняемых работ по ней) по каждомубизнес-процессу (в виде таблицы);
- матрица ответственности:

– сверху – бизнес-функции / работы;

– слева – подразделения и сотрудники;

– на пересечении (в клеточках матрицы) – рабочие места, на которых выполняются соответствующие функции / работы;

• штатное расписание в форме таблицы:

– подразделение,

– по каждому подразделению – должности,

– по каждой должности – количество сотрудников данной должности;

• структура программного обеспечения проектируемой информационной системы: модули рабочих мест и их взаимосвязи (рисунок);

Указания к выполнению работы

Общие замечания

Здесь используется классическая технология проектирования информационных систем, позволяющая интуитивно ясно и последовательно перейти от миссии фирмы и её целей существования к функциональной структуре фирмы и, соответственно, к структуре программного обеспечения информационной системы.

Построение бизнес-модели

Создаётся описание бизнес-процессов фирмы / организации. При этом, как правило, предполагается, что фирма / организация является узко- профильной, то есть производит только какой-то один товар, или предоставляет только какую-то одну услугу. В реальной жизни это, как правило, не так: большинство фирм / организаций являются многопрофильными. Но при выполнении лабораторной, для упрощения ситуации, предполагается, что фирма / организация – узкопрофильная. При таком предположении задача описания бизнес-процессов существенно упрощается и становится типовой:

• выделяется основной бизнес-процесс – это тот, который приносит деньги. Он декомпозируется на составляющие бизнес-функции, то есть, выделяются действия, выполнение которых обеспечивает выполнение этого основного бизнес-процесса (см. примеры ниже, в частности рисунок 1);

• помимо основного бизнес-процесса возможно выделение поддерживающих бизнес-процессов (дополнительных к основному,

обеспечивающих его выполнение). Например, для библиотеки основным бизнес- процессом будет обслуживание читателей, а поддерживающими будут бизнес-процессы «книгохранилище» и «комплектация книжного фонда». Эти поддерживающие бизнес-процессы являются затратными, но они непосредственно связаны с основным и поддерживают его выполнение;

• поддерживающие бизнес-процессы также декомпозируются на составляющие бизнес-функции. Например, бизнес-процесс «книгохранилище» может декомпозироваться на бизнес-функции:

– поиск и выдача единиц хранения;

– приём и раскладка единиц хранения;

– отслеживание состояния единиц хранения;

– ремонт единиц хранения и др.

• почти во всех самостоятельных фирмах / организациях существуют бизнес-процессы «управление», «учёт» и «вспомогательные».

Учёт – это, обычно, бухгалтерия + формирование различного вида отчётности, выдаваемой вовне по запросам государственных или местных органов власти. Сюда же может входить функция создания рекламы.

«Вспомогательный» бизнес-процесс – это бизнес-функции «бухгалтерия», «отдел кадров», «охрана», «уборка» и другие вспомогательные операции, не имеющие прямого отношения к основному бизнес процессу.

Пример 1. ЧАСТНАЯ ТИПОГРАФИЯ

Бизнес-процессы:

1. Обслуживание клиентов – основной:
   - приём заказа,
   - подготовка макета,
   - печать образца и согласование,
   - изготовление заказа,
   - выдача заказа.
2. Ремонт оборудования – первый поддерживающий:
   - регламентные работы,
   - ремонт оборудования.
3. Снабжение – второй поддерживающий:

- определение потребностей (сбор заявок) по позициям номенклатуры,
- заказ/закупка бумаги, картриджей, типографской краски и др.,
- заказ/закупка запасных частей и материалов,
- заказ/закупка оборудования и инструментов,
- спецодежда, инвентарь, оргтехника, канцпринадлежности.

4. Учёт и бухгалтерия:
   - учёт заказов, калькуляция,
   - учёт материалов,
   - бухгалтерская отчётность,
   - другая отчётность (налог., пенс., соцстрах, госстат., местная и др.).
   - расчёт зарплаты.
5. Вспомогательные бизнес-функции:
   - отдел кадров,
   - охрана,
   - уборка.
6. Управление.

Модель бизнес-процессов приведена на рис. 1.

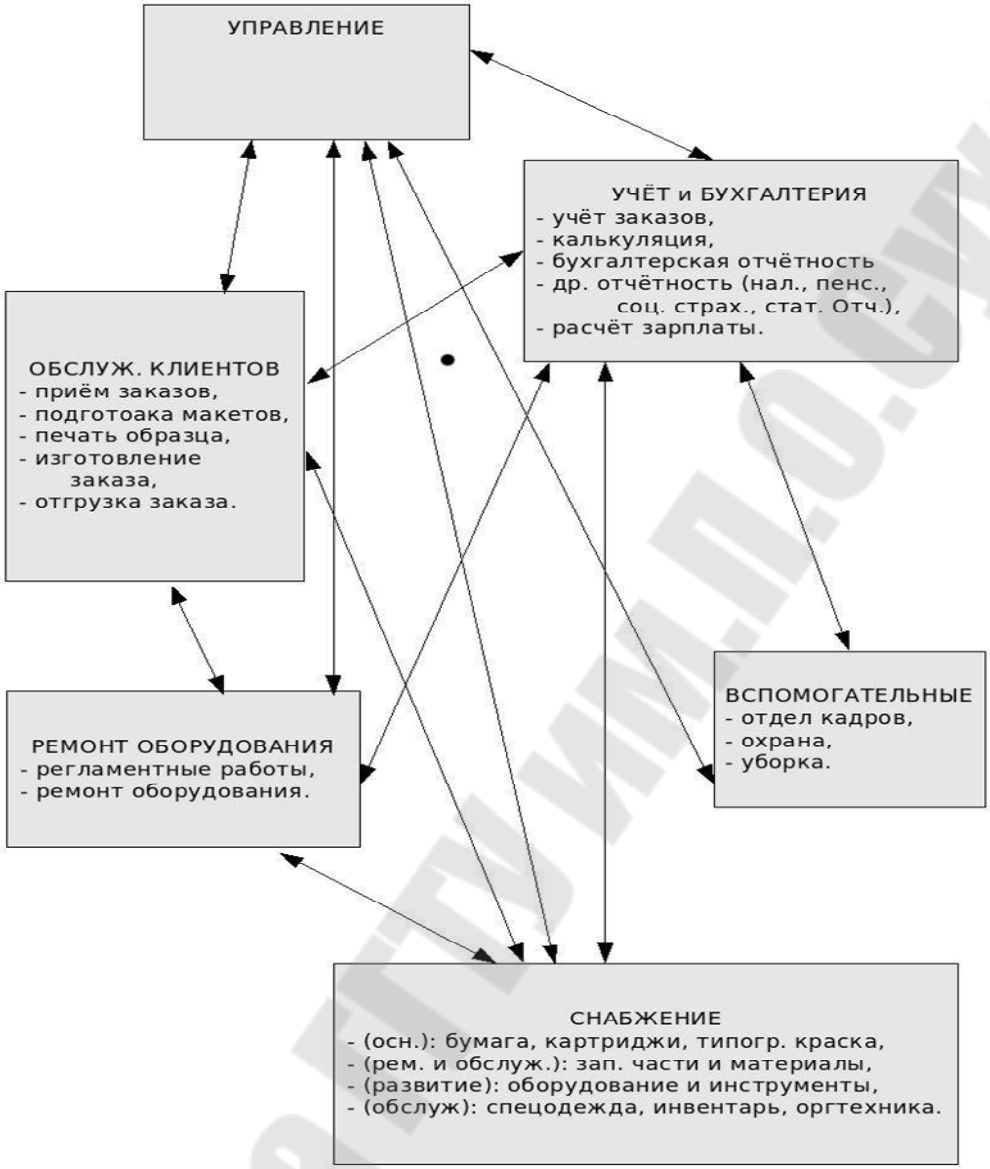


Рис. 1. Модель бизнес-процессов

Основной бизнес-процесс в данной фирме – «обслуживание клиентов»: печать по заказам рекламных материалов, календарей, визиток, брошюр и другой графической печатной продукции, не требующей переплёта.

Поддерживающие бизнес-процессы: «снабжение» и «ремонт и обслуживание техники». Они являются затратными и непосредственно обеспечивают выполнение основного бизнес-процесса.

Примечание: обратите внимание – бухгалтерия может находиться в бизнес-процессах «учёт» или «вспомогательные». Где конкретно – определяется спецификой конкретного предприятия/организации, то есть, степенью привязанности

бухгалтерии к основному бизнес-процессу. Например, в приведённом выше примере, бухгалтерия включена в бизнес-процесс «учёт и бухгалтерия», поскольку она явно обслуживает основной бизнес-процесс: помимо чисто бухгалтерских функций она обеспечивает также калькуляцию.

ЛИТЕРАТУРА

1. Левчук, Е. А. Технологии организации, хранения и обработки данных : учеб. пособие для вузов / Е. А. Левчук. – 2-е изд. – Минск : Выш. шк., 2005. – 239с.

2. Хомоненко, А. Д. Базы данных : учебник для вузов / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев ; под ред. А. Д. Хомоненко. – 5-е изд., доп. – М. : Бином-Пресс : СПб : КОРОНА-Век, 2006. – 736 с.

3. Голенищев, Э. П. Информационное обеспечение систем управления : учеб. пособие для вузов / Э. Л. Голенищев, И. В. Клименко. – Ростов-н/Д : Феникс, 2010. – 315 с.

4. Мельников, В. П. Информационное обеспечение систем управления : учебник / В. П. Мельников. – М. : Академия, 2010. – 335 с.

5. Агальцов, В. П. Базы данных : учебник для вузов / В. П. Агальцов. – М. : ФОРУМ : ИНФРА-М, 2009. – 270 с.

СОДЕРЖАНИЕ

ПРОЕКТИРОВАНИЕ ИНТЕГРИРОВАННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Практикум
по выполнению лабораторных работ
для студентов специальности 1-39 80 03
«Электронные системы и технологии»
дневной и заочной форм обучения

Составители: **Красовская** Наталья Александровна
Щуплов Вячеслав Валентинович

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 03.04.23.

Рег. № 8Е.
http://www.gstu.by