

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ.
Р.Е. АЛЕКСЕЕВА»
АРЗАМАССКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ К ПРАКТИЧЕСКИМ ЗАНЯТИЯМ

по дисциплине

Базы данных и банки знаний

Направление подготовки

11.04.03 Конструирование и технология электронных средств

Направленность

Информационные технологии проектирования радиоэлектронных средств

ФГОС ВО 3++ по направлению подготовки утвержден приказом
Минобрнауки России от «22» сентября 2017 г. № 956

Квалификация (степень) «магистр»

Рассмотрено на заседании
кафедры «Конструирование и
технология радиоэлектронных
средств»

АПИ НГТУ
«25» мая 2021 г.
протокол №6

Арзамас 2021

Оглавление

Практическое занятие №1.....	4
<i>Построение ER-модели для заданной предметной области.</i>	4
Практическое занятие №2.....	12
Выборка данных из одной таблицы.....	12
Выборка данных из двух и более таблиц.	12
Практическое занятие №3.....	14
Выборка данных с использованием условий отбора групп.	14
Соединение таблиц.....	14
Практическое занятие №4.....	17
Выборка данных с использованием квантора общности.....	17
Выборка данных с использованием квантора существования.....	17
Практические занятия №5-7.	17
<i>Выборка данных с использованием вложенных запросов.</i>	17
Практическое занятие №8.....	19
<i>Создание таблиц.</i>	19
Практические занятия №9-10.	21
Изменение и удаление таблиц и индексов.	21
Изменение структуры таблиц.....	21
Практическое занятие №11.....	28
<i>Добавление, обновление и удаление записей.</i>	28
Практические занятия №12-13.	35
Создание транзакции.....	35
Управление транзакциями.....	35
Практическое занятие №14.....	41
Написание триггеров.....	41
Практические занятия №15-16.	45
<i>Написание хранимых процедур простой структуры.</i>	45
<i>Написание хранимых процедур циклической структуры.</i>	45

Требования к организации и проведению практических занятий

На практических занятиях студент выполняет конкретное персональное (индивидуальное) задание, что способствует более эффективному формированию практических умений, навыков и компетенций. Выполнение задания является необходимым условием допуска студента к зачету.

Выполнение студентами практических работ регистрируется преподавателем в журнале. Практические занятия проводятся согласно утвержденному расписанию учебных занятий. Отработка пропущенных студентами практических занятий осуществляется по графику, утвержденному соответствующей кафедрой, как правило, в конце семестра. Замена пропущенных студентами практических занятий другими видами учебных занятий не допускается.

Структура отчета практического занятия и правила его оформления. По результатам выполнения практического занятия студентами оформляется отчет, форма которого утверждается кафедрой.

Прием защиты отчетов практического занятия. Защита отчетов практического занятия является одной из форм текущего контроля успеваемости студентов. Прием защиты отчетов практического занятия осуществляется преподавателем, ведущим практическое занятие. Процедура приема отчетов практического занятия включает проверки:

- соответствия оформления предъявляемым требованиям;
- знаний студентом основных понятий, определений и теоретических положений, применяемых при выполнении заданий;
- знаний студентом методики выполнения заданий;
- умений студентом объяснить полученные результаты;
- степени самостоятельности выполнения практической работы.

Практическое занятие №1 Построение ER-модели для заданной предметной области.

Цель работы: проектирование информационно-логической модели базы данных при помощи case-средства **mySQL Workbench**

Теоретические сведения

1. Создание ER-диаграммы

Среда **mySQL Workbench** предназначена для визуального проектирования баз данных и управления сервером **mySQL**.

Для построения моделей предназначена секция **Data Modeling**:



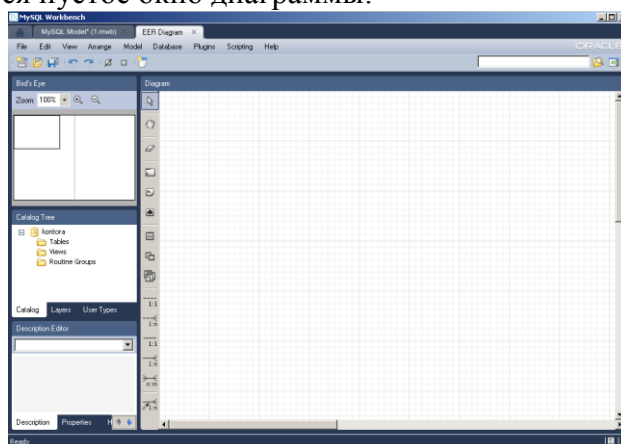
Выберем пункт **Create new EER Model**.


EERmodel расшифровывается как **Extended Entity-Relationship Model** и переводится как **Расширенная модель сущностей-связей**.

По умолчанию имя созданной модели **myDB**. Щелкните правой кнопкой мыши по имени модели и выберите в появившемся меню пункт **Edit schema**. В появившемся окне **можно** изменить имя модели. Назовем ее, например, **kontora**. В именах таблиц и столбцов нельзя использовать русские буквы.

В этом окне также **нужно** настроить так называемую «кодировку страницы» для корректного отображения русских букв **внутри** таблиц. Для этого выберите из списка пункт **«utf-8_general_ci»**. Окно свойств можно закрыть.

Диаграмму будем строить с помощью визуальных средств. Щелкнем по пункту **Add diagram**, загрузится пустое окно диаграммы:



Создать новую таблицу можно с помощью пиктограммы . Нужно щелкнуть по этой пиктограмме, а потом щелкнуть в рабочей области диаграммы. На этом месте появится таблица с названием по умолчанию **table1**. Двойной щелчок по этой таблице открывает окно редактирования, в котором можно изменить имя таблицы и настроить её структуру.

Будем создавать таблицу **Отделы** со следующими столбцами: номер_отдела, полное_название_отдела, короткое_название_отдела. Переименуем **table1** в **k_dept** и начнем создавать столбцы.

Каждый столбец имеет:

- имя (не используйте русские буквы в имени!),
- тип данных. Самые распространенные типы данных:
 - INT – целое число;
 - VARCHAR(размер) – символьные данные переменной длины, в скобках указывается максимальный размер;
 - DECIMAL(размер, десятичные_знаки) – десятичное число;
 - DATE – дата;
 - DATETIME – дата и время.

Далее располагаются столбцы, в которых можно настроить дополнительные свойства поля, включив соответствующий флажок:

- PK (primary key) – первичный ключ;
- NN (notnull) – ячейка не допускает пустые значения;
- UQ (unique) – значение должно быть уникальным в пределах столбца;
- AI (autoincremental) – это свойство полезно для простого первичного ключа, оно означает, что первичный ключ будет автоматически заполняться натуральными числами: 1, 2, 3, и т.п.;
- DEFAULT – значение по умолчанию, т.е., значение, которое при добавлении новой строки в таблицу автоматически вставляется в ячейку сервером, если пользователь оставил ячейку пустой.

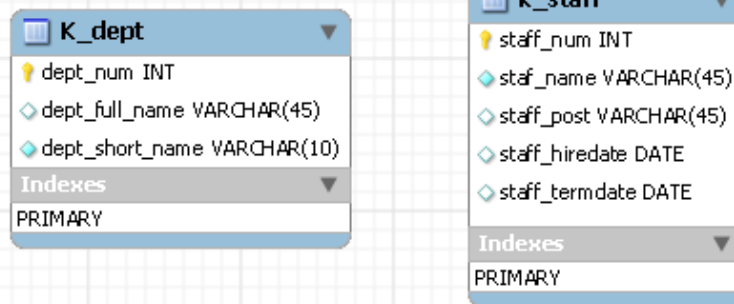
Таблица **Отделы** имеет следующий вид:

K_dept										
Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default	
dept_num	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
dept_full_name	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
dept_short_name	VARCHAR(10)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

Затем создадим таблицу **Сотрудники** со следующими столбцами: номер_сотрудника, имя_сотрудника, должность, дата_начала_контракта, дата_окончания_контракта


k_staff										
Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default	
staff_num	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
staf_name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
staff_post	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
staff_hiredate	DATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
staff_termdate	DATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

Созданные таблицы выглядят следующим образом:

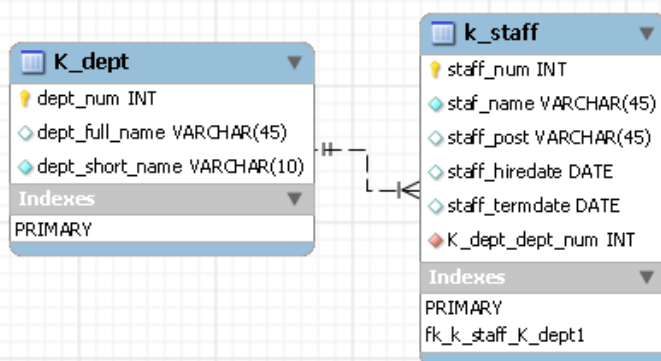


Обратите внимание, что при создании первичного ключа автоматически создается **индекс** по этому первичному ключу. **Индекс** представляет собой вспомогательную структуру, которая служит, прежде всего, для **ускорения поиска** и **быстрого доступа** к данным.

Теперь свяжем эти таблицы. Сначала создадим связь «Работает» между **Сотрудником** (дочерняя таблица) и **Отделом** (родительская таблица), степень связи M:1. Для создания


связей M:1 служит пиктограмма на панели инструментов  (с пунктирной линией). С ее помощью создается так называемая «неидентифицирующая связь», т.е. обыкновенный внешний ключ, при этом первичный ключ родительской таблицы добавляется в список столбцов дочерней таблицы.

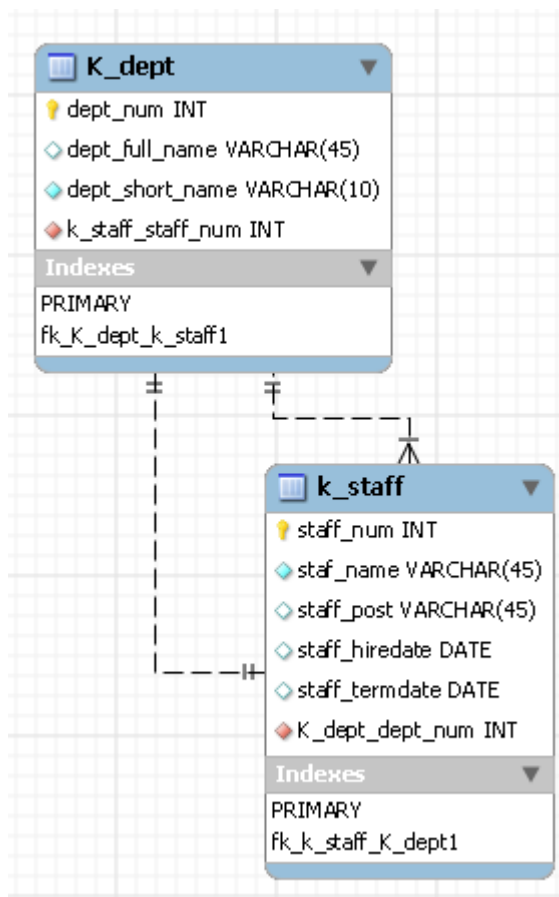
Итак, щелкнем на пиктограмме, затем щелкнем на дочерней таблице **Сотрудники**, затем на родительской таблице **Отделы**:



Обратите внимание, что при этом произошло. Между таблицами образовалась пунктирная линия; в сторону «к одному» она отмечена двумя черточками, в сторону «ко многим» - «куриной лапкой». Кроме того, в таблице **Сотрудники** образовался дополнительный столбец, которому автоматически присвоено имя **k_dept_dept_num** (т.е., имя родительской таблицы плюс имя первичного ключа родительской таблицы). А в группе **Индексы** создан индекс по внешнему ключу.

Теперь добавим связь между этими же таблицами «Руководит» 1:1. Выберем пиктограмму

, затем щелкнем по **Отделам**, затем по **Сотрудникам**.



Чтобы 2 связи на картинке не «завязывались узлом», мы их разместили друг под другом.

Обратите внимание, что в таблицу Отделы был автоматически добавлен столбец *k_staff_staff_num*, а также индекс по внешнему ключу.

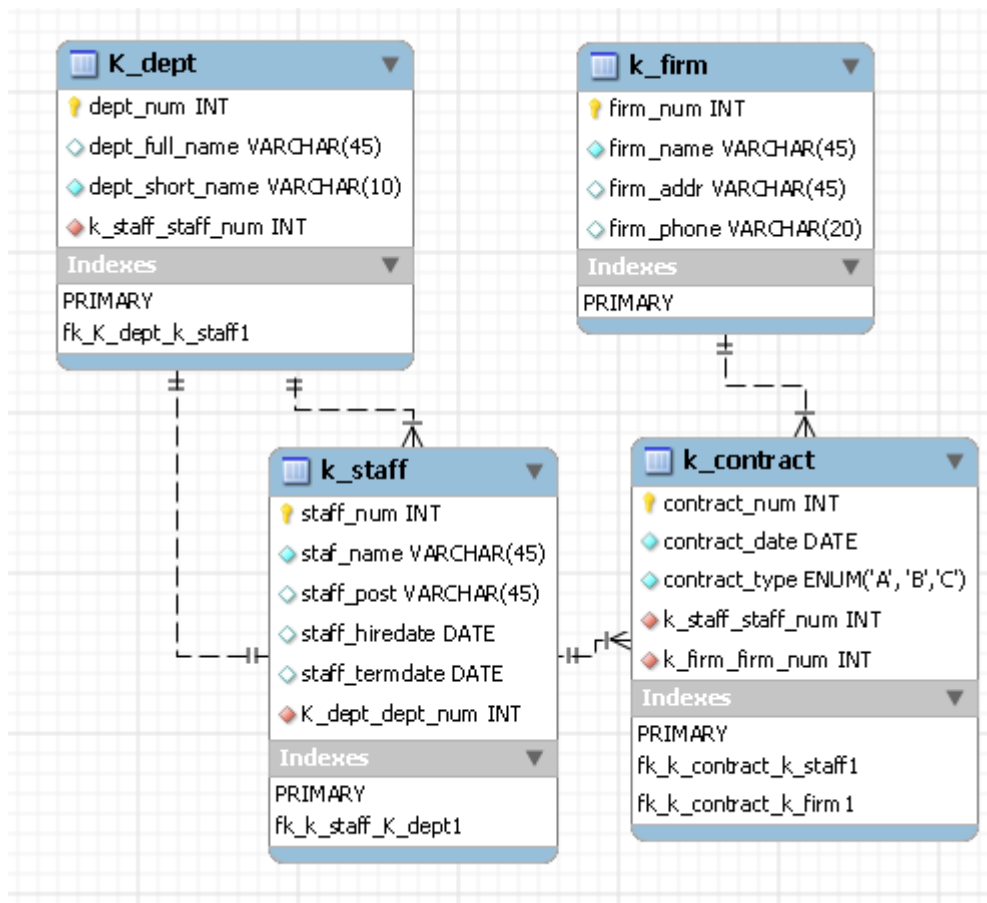
Создадим таблицу **Предприятия**:

k_firm									
Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
firm_num	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
firm_name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
firm_addr	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
firm_phone	VARCHAR(20)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

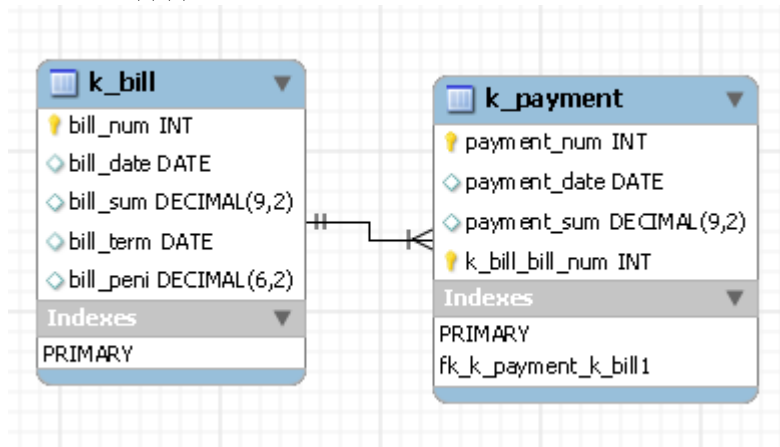
Создадим таблицу **Договоры**. У столбца Тип_договора зададим следующий формат: это буква из списка 'A', 'B', 'C'.


k_contract									
Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
contract_num	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
contract_date	DATE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
contract_type	ENUM('A', 'B', 'C')	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

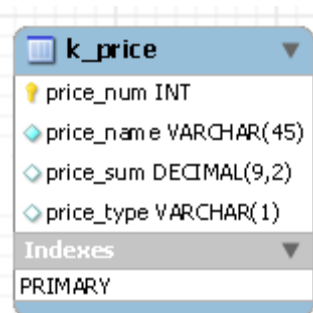
Свяжем Договоры с Сотрудниками и Предприятиями связями М:1.



Затем создадим **Счета** и **Платежи**:




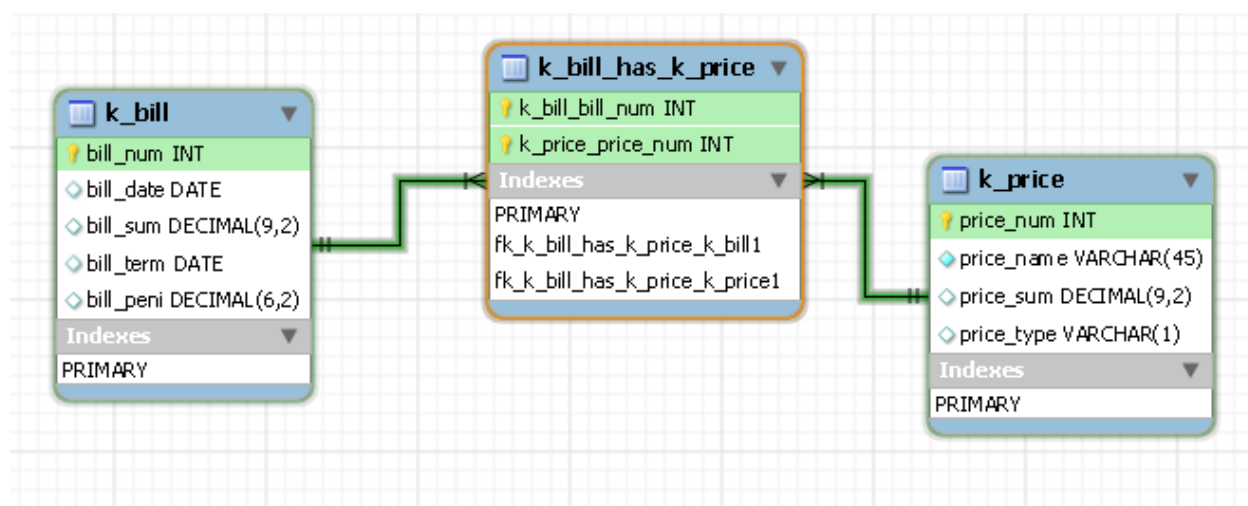
Поскольку сущность **Платеж** была «слабой», у нее нет полноценного первичного ключа, и каждый платеж однозначно идентифицируется группой атрибутов (номер_счета, номер платежа). Отметим в качестве ключевого поля *payment_num*, а затем создадим **идентифицирующую** связь между **Счетом** и **Платежом**. Идентифицирующая связь создается с помощью пиктограммы  (со сплошной линией). При этом новый столбец *k_bill_bill_num* становится не только внешним ключом в таблице **Платеж**, но и частью первичного ключа.



Далее создадим таблицу **Прайс-лист** со столбцами (номер_товара, название_товара, цена_товара и тип_товара).

Между объектами **Счет** и **Прайс-лист** имеется связь «многие- ко многим». Для создания

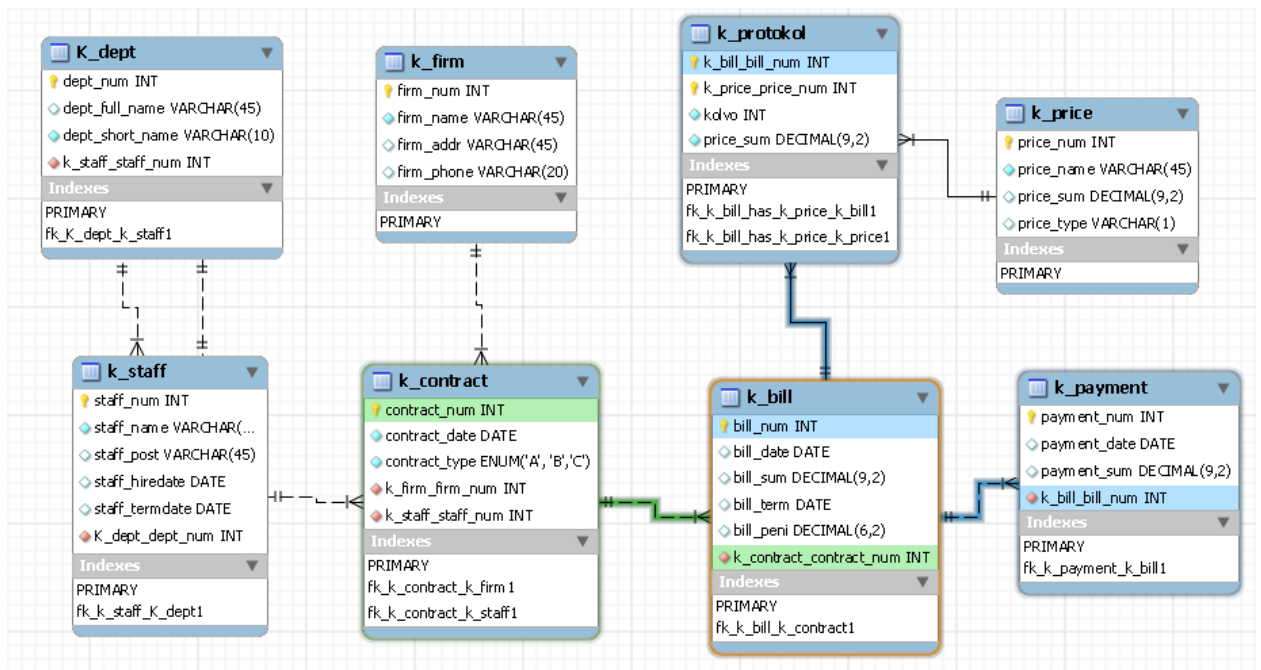
этой связи нужно использовать пиктограмму . Следует щелкнуть мышью по этой пиктограмме, а затем последовательно щелкнуть по связываемым таблицам. Между ними появится новая таблица, обратите внимание на ее столбцы, первичный ключ и внешние ключи:



Для удобства переименуем эту таблицу в *k_protokol* (**ПротоколСчета**), добавим столбцы *kolvo* и *price_sum*.

k_protokol									
Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
⚡ k_bill_bill_num	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
⚡ k_price_price_num	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
💠 kolvo	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
💠 price_sum	DECIMAL(9,2)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Теперь EER-диаграмма имеет такой вид:



Задание. Создайте в MySQLWORKBENCHER-диаграмму для своей задачи.

Номер варианта соответствует номеру задачи.

1. Институт. Таблицы: студенты, преподаватели, предметы, группы, кафедры, книги по предметам. Представление: по студенту найти книги, по предметам, которые он проходит. Роли: Студент — может просматривать, но не может вносить изменения; Преподаватель — может просматривать и вносить изменения в базу данных.
2. Магазин. Таблицы: покупатели, продавцы, товары, покупки (связывает покупателей с товарами и продавцами, продавшими товар), товарные группы, отделы. Представление: по отделам найти покупателей. Роли: Покупатель — видит товары и свои покупки; Продавец: обладает всеми правами.
3. Банк. Таблицы: клиенты, договора(между клиентом и операционистом, на конкретный тип вклада), типы вкладов, отделения банка, операционисты. Представление: найти клиентов, с которыми заключил договора выбранный операционист.. Роли: Клиент, Операционист.
4. Библиотека. Таблицы: читательские билеты, книги, заказы книг (сопоставление книг и читательских билетов), авторы, жанры. Представление: определить любимые жанры выбранного читателя. Роли: Библиотекарь, Читатель.
5. Сотовый оператор. Таблицы: клиенты, записи разговоров (записи о клиенте, времени разговора, тариф), счета клиентов, тарифы, области (где обслуживает оператор). Представление: вывести разговоры, сделанные в выбранной области. Роли: Клиент, Оператор.
6. Агентство недвижимости. Таблицы: недвижимость, клиент, агенты, договор аренды, отделы агентов, цены аренды. Представление: сколько аренды платит выбранный клиент. Роли: Клиент, Агент.
7. Школа. Таблицы: ученики, учителя, оценки, классы, предметы. Представление: по оценкам найти учеников. Роли: Ученики, Учителя.
8. Автосервис. Таблицы: клиенты, машины, мастера. Роли: Клиент, Мастер.
9. Железнодорожная касса. Таблицы: маршруты, поезда, билеты, клиенты, заказы. Представление: вывести заказы билетов по известному поезду. Роли: Кассир, Администратор — имеет полные права.

10. Служба поддержки. Таблицы: объекты, сотрудники, заявки на выполнение работ, неполадки, расходные материалы. Представление: вывести расходные материалы, использованные на выбранном объекте. Роли: Администратор, Техник.

Требования к оформлению отчета

1. Титульный лист
2. Название работы
3. Тему, цель и задание к работе
4. Снимки экрана (скриншоты) процесса разработки
5. Снимок экрана завершенной базы данных
6. Итоговый скрипт БД.
7. Выводы (что узнали, где можно применить полученные знания)

Лабораторная работа №2.Выборка данных из одной таблицы.

Выборка данных из двух и более таблиц.

Цель работы: Изучение структуры и механизма работы оператора SELECT на примере выборки данных из одной и нескольких таблиц без условия отбора и с условием отбора.

Ход работы

1. Создать визуальными средствами редактора phpMyAdmin базу данных в СУБД MySQL, соответствующую приведенной ниже схеме

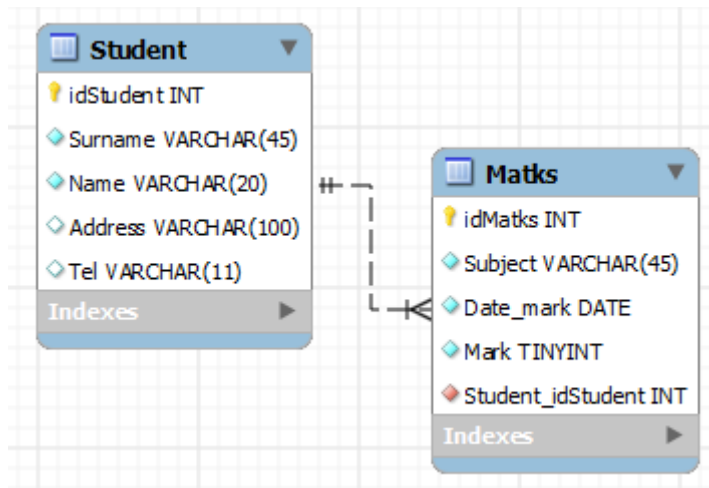


рис.1

2. Напишите следующие запросы
 1. Вывести всю информацию из таблицы Student.
 2. Вывести пары Surname – tel из таблицы Student.
 3. Выведите список студентов, у которых отсутствует в данных номер телефона.
 4. Вывеститройки Surname – Subject — Mark.
 5. Вывести список студентов, у которых по предмету “Математика” оценка 2
 6. Вывести фамилию и первую букву от имени через пробел, тех студентов, фамилии которых начинаются на «а» и содержат не менее одной буквы «в»
 7. Вывести фамилии студентов и номера телефонов, чьи телефоны содержат только цифры от 2-5 и 7
 8. Выведите список фамилий студентов, проживающих в д.78
 9. Выведите всю информацию о студентах, чьи фамилии Иванов, Петров, Сидоров
 10. Выведите фамилии студентов по алфавиту от «Иванова» до «Сидорова»
 11. Выведите фамилии студентов, у которых есть хотя бы одна оценка по Математике ≥ 3 (без повторений)
 12. Переведите каждую оценку в сто бальную систему*. Выведите два столбца: оценка по пяти бальной системе, оценка по сто бальной системе (без повторений).

Ответьте на контрольные вопросы

1. Определить в какой нормальной форме находится отношение, представленное на рис. 1. Обосновать.
2. Структура запроса на выборку.
3. Привести варианты состава оператора SELECT.
4. Описать процесс соединения таблиц.
5. Псевдонимы таблиц.
6. Описать возможные ошибки при выборке данных из нескольких таблиц при отсутствия или неправильного указания условия соединения.

Требования к оформлению отчета

1. Титульный лист.
2. Название работы.
3. Тему, цель и задание к работе.
4. Снимки экрана (скриншоты) процесса разработки.
5. Выводы (что узнали, где можно применить полученные знания).
6. Ответы на контрольные вопросы.

Лабораторная работа №3.Выборка данных с использованием условий отбора групп.

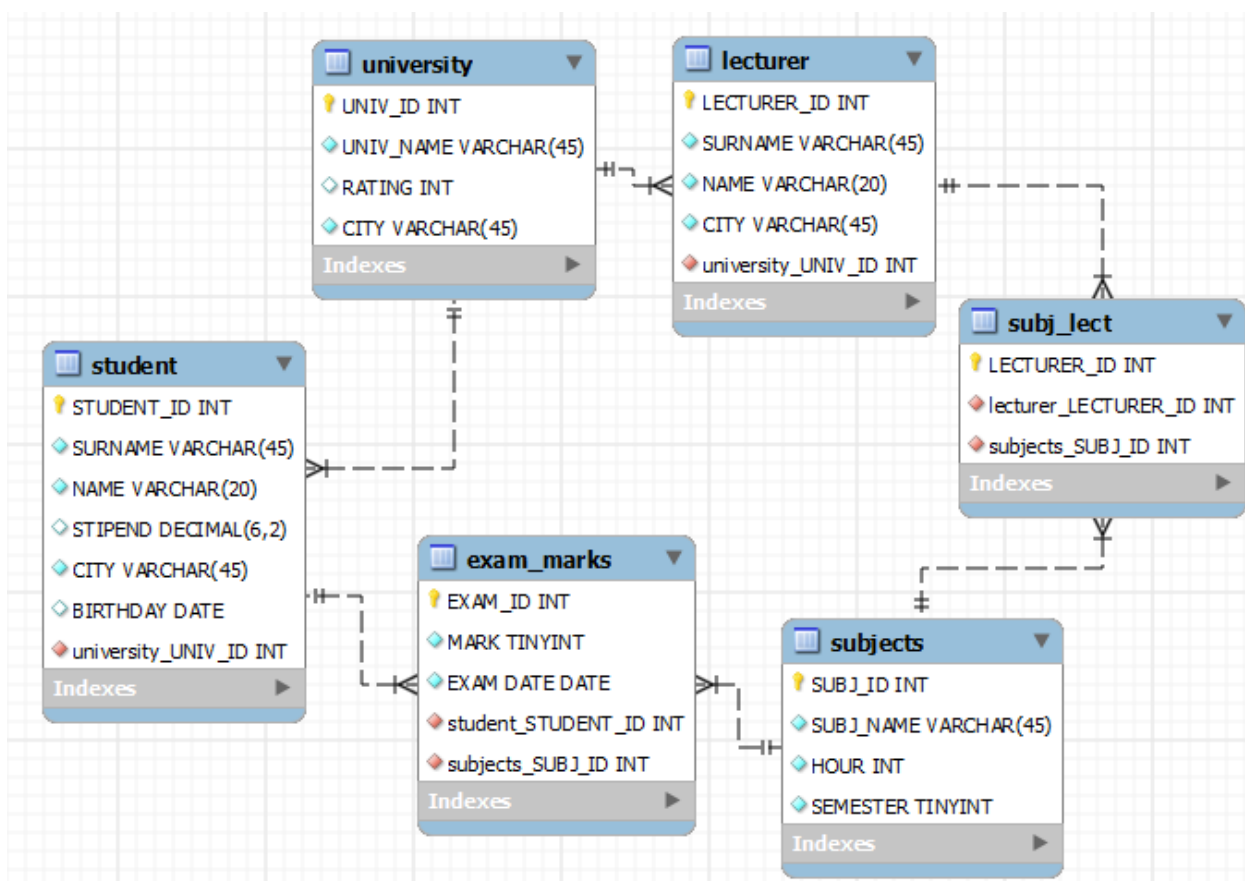
Соединение таблиц.

Цель работы: Изучение групповых функций в SQL и видов соединения таблиц.

Ход работы

1. Открыть созданную на предыдущей лабораторной работе базу данных в СУБД MySQL
2. Напишите следующие запросы
 1. Определить количество студентов
 2. Определить количество оценок по каждому предмету
 3. Определить средний бал по каждому предмету
 4. Определить максимальную оценку
 5. Подсчитать количество студентов, в фамилии которых встречается буква «а» более одного раза

Таблица student



STUDENT_ID — числовой код, идентифицирующий студента,
SURNAME — фамилия студента,
NAME — имя студента,
STIPEND — стипендия, которую получает студент,
KURS — курс, на котором учится студент,
CITY — город, в котором живет студент,

BIRTHDAY — дата рождения студента,

Таблица lecturer

LECTURER_ID — числовой код, идентифицирующий преподавателя;

SURNAME — фамилия преподавателя,

NAME — имя преподавателя,

CITY — город, в котором живет преподаватель,

UNIV_ID — идентификатор университета, в котором работает преподаватель.

Таблица subjects

SUBJ_ID — идентификатор предмета обучения,

SUBJ_NAME — наименование предмета обучения,

HOURL — количество часов, отводимых на изучение предмета,

SEMESTER — семестр, в котором изучается данный предмет.

Таблица university

UNIV_ID — идентификатор университета,

UNIV_NAME — название университета,

RATING — рейтинг университета,

CITY — город, в котором расположен университет.

Таблица exam_marks

EXAM_ID — идентификатор экзамена,

MARK — экзаменационная оценка,

EXAM DATE — дата экзамена.

Таблица subj_lect

LECTURER_ID — идентификатор преподавателя.

1. Напишите запрос, который выполняет выборку значений фамилии *всех* студентов с указанием для студентов, сдававших экзамены, идентификаторов сданных ими предметов обучения.
2. Напишите запрос на выдачу списка фамилий студентов (в алфавитном порядке) вместе со значением рейтинга университета, где каждый из них учится, включив в список и тех студентов, для которых в базе данных не указано место их учебы.
3. Напишите запрос на выдачу данных о названиях всех предметов, по которым студенты получили только хорошие (4 и 5) оценки. В выходных данных должны быть приведены фамилии студентов, названия предметов и оценка.
4. Написать запрос, выполняющий вывод списка всех пар фамилий студентов, проживающих в одном городе. При этом не включать в список комбинации фамилий студентов самих с собой (то есть комбинацию типа «Иванов-Иванов») и комбинации фамилий студентов, отличающиеся порядком следования (то есть включать одну из двух комбинаций типа «Иванов-Петров» и «Петров-Иванов»).

Контрольные вопросы

1. Перечислить и описать основные групповые функции в SQL
2. Привести примеры ошибочных запросов с групповыми функциями

3. Виды соединения.
4. Опишите каждый вид соединения таблиц и приведите примеры.

Требования к оформлению отчета

1. Титульный лист.
2. Название работы.
3. Тему, цель и задание к работе.
4. Снимки экрана (скриншоты) процесса разработки.
5. Выводы (что узнали, где можно применить полученные знания).
6. Ответы на контрольные вопросы.

Лабораторная работа №4.Выборка данных с использованием квантора общности.

Выборка данных с использованием квантора существования.

Цель работы: изучение механизма работы кванторов.

Ход работы

1. Открыть созданную на предыдущей лабораторной работе базу данных в СУБД MySQL
2. Напишите следующие запросы
 1. Напишите запрос, выбирающий из таблицы EXAM_MARKS данные о названиях предметов обучения, для которых значение полученных на экзамене оценок (поле MARK) превышает любое значение оценки для предмета, имеющего идентификатор, равный 105.
 2. Напишите запрос, по каким предметам меньше оценок, чем по другим
 3. Написать запрос, выполняющий вывод списка всех пар названий университетов, расположенных в одном городе, не включая в список комбинации названий университетов самих с собой и пары названий университетов, отличающиеся порядком следования.
 4. Напишите команду SELECT, использующую **связанные** подзапросы и выполняющую **вывод** имен **и** идентификаторов студентов, у которых стипендия совпадает с максимальным значением стипендии для города, в котором живет студент.
 5. Напишите запрос, который позволяет вывести имена и идентификаторы всех студентов, для которых точно известно, что они проживают в городе, где нет ни одного университета.

Контрольные вопросы

1. Назначение квантора общности в SQL-запросах.
2. Обоснование квантора общности на примере логики предикатов.
3. Опишите механизм работы оператора EXISTS.

Требования к оформлению отчета

1. Титульный лист.
2. Название работы.
3. Тему, цель и задание к работе.
4. Снимки экрана (скриншоты) процесса разработки.
5. Выводы (что узнали, где можно применить полученные знания).
6. Ответы на контрольные вопросы.

Лабораторная работа №5-7Выборка данных с использованием вложенных запросов.

Цель работы: Изучить виды вложенных подзапросов.

Ход работы

1. Открыть созданную на предыдущей лабораторной работе базу данных в СУБД MySQL.

2. Напишите следующие запросы

1. Написать запрос, который позволяет получить данные о названиях университетов и городов, в которых они расположены, с рейтингом, равным или превышающим рейтинг ВГУ.
2. Напишите запрос, выбирающий данные об именах всех студентов, имеющих по предмету с идентификатором 101 балл выше общего среднего балла.
3. Напишите запрос, который выполняет выборку имен всех студентов, имеющих по предмету с идентификатором 102 балл ниже общего среднего балла.
4. Напишите запрос, выполняющий вывод количества предметов, по которым экзаменовался каждый студент, сдававший более 20 предметов. Напишите два запроса, которые позволяют вывести имена и идентификаторы всех студентов, для которых точно известно, что они проживают не в том городе, где расположен их университет. Один запрос с использованием соединения, а другой — с использованием связанного подзапроса.
5. Извлечь из таблицы EXAM_MARK данные о студентах, получивших хотя бы одну неудовлетворительную оценку, среди тех студентов у которых средний балл по остальным предметам больше 4.
6. Напишите команду SELECT, использующую связанные подзапросы и выполняющую вывод имен и идентификаторов студентов у которых стипендия совпадает с максимальным значением стипендии для города, в котором живет студент.
7. Напишите запрос с EXISTS, выбирающий сведения обо всех студентах, для которых в том же городе, где живет студент, существуют университеты, в которых он не учится.
8. Напишите запрос, который выполняет вывод данных о фамилиях студентов, сдававших экзамены, вместе с наименованиями каждого сданного ими предмета обучения.
9. Напишите запрос на выдачу для каждого студента названий всех предметов обучения, по которым этот студент получил оценку 4 или 5.
10. Напишите запрос, который выполняет вывод списка университетов с рейтингом, превышающим 300, вместе со значением максимального размера стипендии, получаемой студентами в этих университетах.

Требования к оформлению отчета

1. Титульный лист.
2. Название работы.
3. Тему, цель и задание к работе.
4. Снимки экрана (скриншоты) процесса разработки.
5. Выводы (что узнали, где можно применить полученные знания).

Лабораторная работа №.8Создание таблиц.

Цель работы: Изучение структуры запроса на создание таблиц.

Ход работы

1. Создать в базе данных следующие таблицы

Таблица Абитуриенты

Поле	Номер
Тип	Длинное целое (4)
Примечание	Поле имеет специальный тип «Счетчик»
Новые значения	Последовательные
Примечание	Данное поле является ключом таблицы
Индексированное поле	Да (совпадения не допускаются)
Поле	Фамилия
Тип	Текстовый
Размер	20
Обязательное поле	Да
Пустые строки	Нет
Индексированное поле	Нет
Поле	Имя
Тип	Текстовый
Размер	15
Обязательное поле	Да
Пустые строки	Нет
Индексированное поле	Нет
Поле	Отчество
Тип	Текстовый
Размер	15
Обязательное поле	Нет
Индексированное поле	Нет
Поле	Номер_аттестата
Тип	Текстовый
Размер	10
Поле	Дата_аттестата

Тип	Дата/время
Размер	8
Формат поля	Краткий формат даты
Обязательное поле	Нет

Таблица Экзамены

Поле	Код_экзамена
Тип	Длинное целое (4)
Примечание	Поле имеет специальный тип «Счетчик»
Новые значения	Последовательные
Примечание	Данное поле является ключом таблицы
Индексированное поле	Да (совпадения не допускаются)
Поле	Предмет
Тип	Текстовый
Размер	30
Поле	Дата_экзамена
Тип	Дата/время
Размер	8
Формат поля	Краткий формат даты
Обязательное поле	Да
Поле	Тип_экзамена
Тип	Текстовый
Размер	30
Поле	Факультет
Тип	Текстовый
Размер	5

Таблица Оценки

Поле	Код_экзамена
Тип	Длинное целое (4)
Обязательно поле	Да
Индексированное поле	Да (совпадения допускаются)
Поле	Номер_абитуриента
Тип	Длинное целое (4)
Индексированное поле	Да (совпадения допускаются)

Поле	Оценка
Тип	Числовой (одинарное с плавающей точкой)
Формат поля	Фиксированный
Число десятичных знаков	1
Условие на значение	≥ 2 And ≤ 5
Обязательное поле	Нет

2. **Изобразите схему полученной базы данных**
3. **Ответьте на контрольные вопросы**
 1. Структура запроса CREATE TABLE.
 2. Скрипты на создание приведенных выше таблиц (с комментариями).
 3. Объясните различие между типами char и varchar.

Требования к оформлению отчета

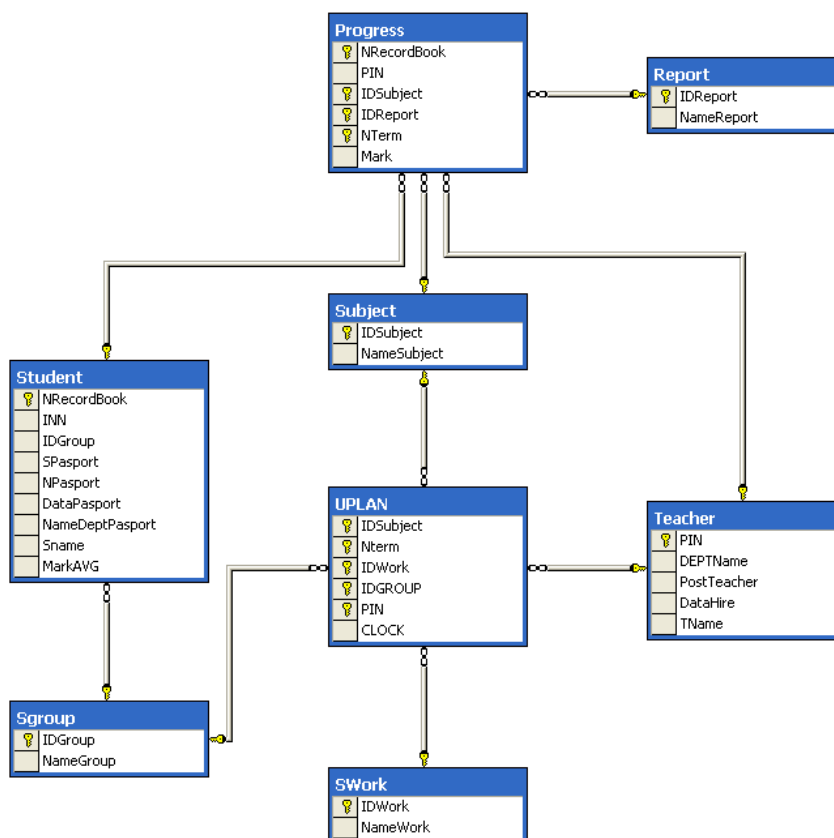
1. Титульный лист.
2. Название работы.
3. Тему, цель и задание к работе.
4. Снимки экрана (скриншоты) процесса разработки.
5. Выводы (что узнали, где можно применить полученные знания).
6. Ответы на контрольные вопросы.

Лабораторная работа №9-10.Изменение и удаление таблиц и индексов.

Изменение структуры таблиц.

Цель работы: Изучить команды DDL: ALTER TABLE, DROP TABLE.

Ход работы



Команда ALTER TABLE

Если при создании таблицы были допущены ошибки в ее описании, исправить их можно несколькими способами. Во-первых, если таблица еще не содержит информации, ее можно просто удалить и создать снова. В противном случае целесообразно использовать команду ALTER TABLE.

Команда ALTER TABLE позволяет

Добавлять и удалять столбцы.

Добавлять и изменять описания столбцов.

Добавлять, удалять, отключать ограничения.

При успешном завершении команды выдается сообщение Thecommand(s) completedsuccessfully. (Команда выполнена успешно).

Модификация ограничений

Синтаксис команды: ALTER TABLE <имя таблицы> ALTER| ADD| DROP {CONSTRAINT <имя ограничения>} FOREIGN KEY [REFERENCES<имя таблицы> (<имя столбца> [,...n])] | PRIMARY KEY | UNIQUE | CHECK (<имя столбца> [,...n])} [ON DELETE CASCADE]

Если необходимо наложить дополнительное ограничение на значение атрибута, то следует использовать команду ALTER совместно с опцией ADD, которая позволяет добавлять столбец или новое ограничение в таблицу, в свою очередь опция DROP даст возможность удалить то или иное ограничение, наложенное на данные в таблице. Если ограничение необходимо исправить (Foreignkey, Primarykey, Unique, Check), то его сначала удаляют, а затем создают снова. Для того чтобы удалить ограничение, необходимо указать его имя. Просмотреть созданные ограничения можно выполнив системную хранимую процедуру:

EXEC SP_HELP <имя таблицы>

Пример 1

Задача.

Ввести ограничение на столбец DataPasport, полагая, что значение этого атрибута должно превышать значение 01.01.2000.

Решение.

```
ALTER TABLE Student ADD CONSTRAINT  
StudentYearBeginCheckCheck(DataPasport>'01.01.2000');
```

Примечание.

При попытке вставить следующую строку

```
INSERT INTO Student (NRecordBook,SName,IDGroup,SPasport,NPasport,DataPasport,  
NameDeptPasport,INN) VALUES('050004','Митькин М.М.', 2,'8701','192417','11.26.1999',  
'УВД г.Ухты','111111114') система генерирует следующее сообщение об ошибке: Server:  
Msg 547
```

```
INSERT statement conflicted with COLUMN CHECK constraint 'StudentYearBeginCheck'.  
The conflict occurred in database 'Student', table 'Student', column 'DataPasport'.  
The statement has been terminated.
```

При запуске же команды:

```
INSERT INTO Student (NRecordBook,SName,IDGroup,SPasport,NPasport,DataPasport,  
NameDeptPasport,INN)VALUES('050004','Митькин М.М.',  
2,'8701','192417','11.26.2000','УВД г.Ухты','111111114')
```

система генерирует сообщение:

(1 row(s) affected)

Пример 2

Задача.

Ввести ограничения FOREIGN KEY для таблицы Uplan. Таблица Uplan ссылается на таблицу Subject по атрибуту IDSubject.

Решение.

```
ALTER TABLE UPlan ADD CONSTRAINT PlanSubjectForeign FOREIGN KEY (IDSubject)  
REFERENCES Subject(IDSubject);
```

! Обратите внимание, что когда ограничение FOREIGN KEY задается таким образом, что ни одна строка в отношении Subject не может быть удалена, до тех пор пока в отношении UPlan есть строки, ссылающиеся на удаляемый предмет. Как избежать этой коллизии смотрите в следующем примере

Задача.

Ввести ограничения FOREIGN KEY для таблицы UPlan таким образом, чтобы при удалении из таблицы Subject записей по тому или иному предмету, были бы удалены и соответствующие этим предметам записи из таблицы UPlan.

Решение. ALTER TABLE UPlan ADD CONSTRAINT PlanSubjectForeign FOREIGN KEY (IDSubject) REFERENCES Subject(IDSubject) ON DELETE CASCADE;

Проблему каскадного удаления соответствующих строк в ссылающейся таблице (Progress) при удалении строк в ссылаемой таблице (Subject) решает опция ON DELETE CASCADE. Для того чтобы удалить ограничение, необходимо указать его имя.

Пример 3

Задача.

Удалить ограничение FOREIGN KEY PlanSubjectForeign на атрибуте IDSubject в таблице Uplan.

Решение.

Прежде чем удалить ограничение, выясним его имя. В случае, если вы руководствовались своим правилом именования объектов, эта задача значительно упрощается. В противном случае можно использовать уточнить имя ограничения, выполнив команду EXEC sp_help UPLAN.

```
ALTER TABLE UPlan DROP CONSTRAINT PlanSubjectForeign ;
```

Добавление ограничений с ограниченной областью проверки

Синтаксискоманды: ALTER TABLE <имя таблицы> [WITH CHECK | WITH NOCHECK] ADD {CONSTRAINT <имя ограничения>} FOREIGN KEY [REFERENCES <имя таблицы> (<имя столбца> [...n])] | CHECK (<имя столбца> [...n])} [ON DELETE CASCADE]

В некоторых ситуациях может возникнуть необходимость в отмене проверки ограничений на какой-то период времени. Или возникнет ситуация, когда необходимо ввести новое ограничение, но вывести из-под его действия уже существующие данные. Сразу следует отметить, что ни одна из вышеперечисленных операций не может быть произведена для ограничений PRIMARY KEY и UNIQUE.

Таким образом, если мы вводим новое ограничение и хотим, чтобы SQL Server 2000 проверил все существующие данные на соответствие этому ограничению, следует добавить это ограничение с опцией WITH CHECK, в противном случае – с опцией WITH NOCHECK

Временное отключение ограничений и использование опции WITH NOCHECK требует большой осторожности. В противном случае вы можете получить некорректный результат. Например, в таблице Progress могут оказаться оценки несуществующего студента, учащегося в несуществующей группе, изучающего несуществующие дисциплины и т. д.

Пример 4

Задача.

Добавить ограничение FOREIGN KEY для столбца IDGroup в таблице Student5. Все данные в таблице Student проверить на соответствие новому ограничению.

Решение.

```
ALTER TABLE Student WITH CHECK ADD CONSTRAINT StudentIDGroupForeign FOREIGN KEY (IDGroup) REFERENCES SGroup(IDGroup)
```

Результат выполнения команды будет зависеть от состояния вашей базы данных. Вполне возможно, что команда не будет выполнена, тогда вы получите следующее сообщение, говорящее о том, что нарушена целостность базы данных.

Server: Msg 547 ALTER TABLE statement conflicted with COLUMN FOREIGN KEY constraint 'StudentIDGroupForeign'. The conflict occurred in database 'Student', table 'SGroup', column 'IDGroup'.

Пример 5

Задача.

Добавить ограничение FOREIGN KEY для столбца IDGroup в таблице Student. Все данные в таблице Student, введенные на момент создания ограничения, проверке не подлежат.

Решение.

```
ALTER TABLE Student WITH NOCHECK ADD CONSTRAINT StudentIDGroupForeign FOREIGN KEY (IDGroup) REFERENCES SGroup(IDGroup)
```

В этом случае независимо от содержания таблиц результат будет следующим.

The command(s) completed successfully.

Задание 1 Ввести ограничение на оценку в отношении Успеваемость. Оценка не должна превышать 5 баллов. Номер семестра не должен превышать 10.

Задание 2 Создать внешние ключи во всех таблицах, используя опцию ForeignKey, при этом установить опцию каскадного удаления там, где это необходимо.

Задание 3 Удалить первичный ключ в отношении Student.

Задание 4 Проследить за изменением ограничения ForeignKey в отношениях, связанных с отношением Student. Еще раз восстановите все удаленные ограничения.

Отключение и подключение ограничений

Отключить можно как отдельное ограничение, указав его имя, так и все, используя опцию ALL

Синтаксис команды:

ALTER TABLE <имя таблицы>[CHECK| NOCHECK]{CONSTRAINT <имя ограничения>| ALL}FOREIGN KEY [REFERENCES<имя таблицы> (<имя столбца> [...n])] | CHECK (<имя столбца> [...n])}

Задание 5 Отключите ограничения внешнего ключа в таблице Student. Введите в таблицу Student студента Васькина В.В. из несуществующей группы. Попробуйте подключить ранее отключенное ограничение.

Выполните все необходимые действия для того, чтобы вновь подключить ограничение, а все данные в отношении Student соответствовали условиям целостности базы данных.

Задание 6 Моделируйте ситуацию, когда необходимо отключить ограничения и разработайте мероприятия, которые позволят вам в дальнейшем привести базу данных в согласованное состояние, отвечающее всем условиям целостности.

Правила для изменения и модификации описания столбцов

При корректировке таблиц нельзя:

добавлять новый столбец с опцией NOT NULL.

добавлять к столбцу опцию NOT NULL, если в нем есть пустые значения

уменьшить размер поля или изменить его тип, если в нем содержатся какие-либо данные.

Удалить столбец из таблицы, если на этот столбец были установлены какие-либо ограничения кроме NOT NULL| NULL^

Добавление столбца

Синтаксис команды: ALTER TABLE <имя таблицы>ADD <имя столбца><тип данных><ширина столбца>[DEFAULT <значение>] [...n]; где DEFAULT – определяет значение столбца по умолчанию.

При добавлении столбца он автоматически становится последним в таблице. Изменить положение столбца в таблице не представляется возможным.

Пример

Задача.

Добавить столбец YearBegin (год начала учебы в институте) в таблицу Student, задав тип данных Datetime.

Решение.

ALTER TABLE StudentADDYearBeginDatetime;

Модификация столбца

Синтаксис команды: ALTER TABLE <имя таблицы>

ALTER COLUMN <имя столбца><новый тип данных><длина>[DEFAULT <значение>][NULL|NOT NULL] [...n];

SQL Server не разрешает изменять столбцы типа text, ntext, image, rowversion, вычисляемые столбцы, столбцы, используемые в репликации, и столбцы, на которые имеются ссылки в выражениях вычисляемых столбцов или ограничений, а также столбцы с установленным свойством ROWGUIDCOL6. Нельзя удалить или изменить столбец, имеющий значение по умолчанию (ограничение DEFAULT). Однако можно увеличить размер столбцов переменной длины, которые используются в индексах, в ограничениях CHECK или UNIQUE.

Задание 7 Добавить в таблицу Student столбец Single, тип данных VARCHAR(3), назначив значение по умолчанию “Да”. Удалить столбец.

Задание 8 Добавить в таблицу Student столбец AVGMARK, тип Numeric (5,2). В столбце будет храниться средняя оценка студента. Мы оставим этот столбец в базе данных лишь для того, чтобы в дальнейшем продемонстрировать с помощью него работу некоторых команд и процедур, написание которых как раз и будет обусловлено наличием этого избыточного столбца. Отсюда вывод – такие столбцы, содержащие расчетные данные, полученные на основании уже хранящихся в таблице данных, не следует включать в таблицы.

Задание 9 Изменить длины полей в соответствии с таблицей. Выполнить анализ – почему не удалось выполнить заданные операции с некоторыми столбцами? Что необходимо предпринять, чтобы эти изменения всё же произвести?

Имя поля	Тип поля	Размер	Ограничения
IDReport	Varchar	4	
NameWork	Varchar	4	
NameSubject	Varchar	4	
DateHire	Smalldatetime		
Mark	Numeric	2	NULL
DeptName	Varchar	4	NULL
NRecordBook	Varchar	6	
NTerm	Numeric	2	
NameReport	Varchar	35	
NameSubject	Varchar	35	
PIN	Varchar	4	
TeachPost	Varchar	25	NULL
Clock	Numeric	5.2	
SName	Varchar	35	
TeachName	Varchar	35	

Удаление столбца

Синтаксис команды: ALTER TABLE <имя таблицы> DROP COLUMN <имя столбца>

Ранее уже отмечалось, что нельзя удалить столбец, если на него наложено хотя бы одно ограничение за исключением NULL. Кроме этого следует отметить, что нельзя удалить Реплицированные столбцы.

Индексированные столбцы.

Столбцы, для которых определено правило (rule).

Это связано с тем, что все ограничения целостности, значения по умолчанию, индексы и правила хранятся в виде отдельных объектов базы данных и связываются со столбцом таблицы. Удаление столбцов без предварительного удаления объектов привело бы к появлению в базе данных несвязанных объектов, что способствовало бы ее засорению.

Удаление таблицы при наличии на нее ссылок

Таблица удаляется с помощью команды DROP. Одной командой можно удалить сразу несколько таблиц. Однако вы не сможете удалить таблицу, если на нее есть ссылки из других таблиц. То есть вам сначала придется удалить соответствующий внешний ключ в ссылочной таблице и только после этого выполнить команду DROP

Синтаксис команды: DROP TABLE <имя таблицы>[, <имя таблицы>]

Переименование таблицы

Иногда приходится корректировать не только ограничения и структуру таблицы, но и ее имя. Для переименования таблиц используется системная хранимая процедура SP_RENAME. Однако следует быть предельно осторожным, так как в этом случае необходимо корректировать и все ограничения, ссылающиеся на эту таблицу.

Синтаксис команды: EXEC SP_RENAME 'старое имя', 'новое имя' ;

Задание 10 Переименовать таблицу Progress в таблицу Progress1.

Контрольные вопросы

Каково основное назначение команды ALTER?

Какие операции над ограничениями можно выполнить с помощью команды ALTER?

Какие ограничения подлежат корректировке?

Каковы правила назначения ограничения NULL/NOT NULL?

Когда нельзя изменить ширину столбца в таблице?

Какие существуют ограничения на изменения параметров столбца?

Как можно отменить действие ограничения?

Для столбцов с каким типом данных разрешены изменения их размера?

Как можно удалить столбец, если на него наложено одно из ограничений?

Как переименовать столбец в таблице? В каких случаях это возможно?

В каких случаях используют отключение ограничений?

Допустим, мы вводим новое ограничение и хотим, чтобы SQL проверил все существующие данные на соответствие этому ограничению, что следует для этого сделать?

Каковы могут быть последствия временного отключения ограничений?

Какова последовательность действий при модификации столбца, имеющего ограничение DEFAULT?

Какое место в таблице занимает вновь создаваемый столбец?

Возможно ли изменить порядок следования столбцов в таблице, не используя операцию удаления?

Какие из ограничений не могут быть временно отключены?

Требования к оформлению отчета

1. Титульный лист.
2. Название работы.
3. Тему, цель и задание к работе.
4. Снимки экрана (скриншоты) процесса разработки.
5. Выводы (что узнали, где можно применить полученные знания).
6. Ответы на контрольные вопросы.

Лабораторная работа №11. Добавление, обновление и удаление записей.

Цель работы: знакомство с операторами DML.

Теоретические сведения

Рассмотрим следующие вопросы:

- вставка данных с помощью оператора *INSERT*;
- удаление данных операторами *DELETE* и *TRUNCATE*;
- обновление данных с помощью оператора *UPDATE*.

После создания БД и таблиц перед разработчиком встает задача заполнения таблиц данными. В реляционных БД традиционно применяют три подхода:

- однострочный оператор *INSERT* – добавляет в таблицу новую запись;
- многострочный оператор *INSERT* – добавляет в таблицу несколько записей;
- пакетная загрузка *LOADDATA INFILE* – добавление данных из файла.

Вставка данных с помощью оператора *INSERT*. Однострочный оператор *INSERT* может использоваться в нескольких формах. Упрощенный синтаксис первой формы:

*INSERT [IGNORE] [INTO] имя_таблицы [(имя_столбца, ...)]
VALUES (выражение, ...);*

Оператор вставляет новую запись в таблицу *имя_таблицы*. Значения полей записи перечисляются в списке (*выражение, ...*). Порядок следования столбцов задается списком (*имя_столбца, ...*). Список столбцов (*имя_столбца, ...*) позволяет менять порядок следования столбцов при добавлении.

Первичный ключ таблицы является уникальным, и попытка добавить уже существующее значение приведет к ошибке. Чтобы новые записи с дублирующим ключом отбрасывались без генерации ошибки, следует добавить после оператора *INSERT* ключевое слово *IGNORE*.

Другая форма оператора *INSERT* предполагает использование слова *SET*:

*INSERT [IGNORE] [INTO] имя_таблицы
SET имя_столбца1 = выражение1, имя_столбца2 = выражение2, ... ;*

Оператор заносит в таблицу *имя_таблицы* новую запись, столбец *имя_столбца* которой получает значение *выражение*.

Многострочный оператор *INSERT* совпадает по форме с однострочным оператором, но после ключевого слова *VALUES* добавляется через запятую несколько списков (*выражение, ...*).

Практические примеры использования оператора *INSERT* для заполнения учебной БД *book* см. ниже, в пункте «Пример выполнения работы».

Удаление данных. Для удаления записей из таблиц предусмотрены:

- оператор *DELETE*;
- оператор *TRUNCATE TABLE*.

Оператор *DELETE* имеет следующий синтаксис:

*DELETE FROM имя_таблицы
[WHERE условие]
[ORDER BY имя_поля]
[LIMIT число_строк];*

Оператор удаляет из таблицы *имя_таблицы* записи, удовлетворяющие условию. В следующем примере из таблицы *catalogs* удаляются записи, имеющие значение первичного ключа *catalog_id* больше двух.

```
mysql> DELETE FROM catalogs WHERE cat_ID>2;
Query OK, 3 rows affected (0.05 sec)
```

```
mysql> SELECT * FROM catalogs;
+-----+-----+
| cat_ID | cat_name |
+-----+-----+
|      1 | Программирование |
|      2 | Интернет |
+-----+-----+
2 rows in set (0.00 sec)
```

Если в операторе отсутствует условие *WHERE*, удаляются все записи таблицы.

```
mysql> DELETE FROM catalogs;
Query OK, 2 rows affected (0.03 sec)
```

```
mysql> SELECT * FROM catalogs;
Empty set (0.00 sec)
```

Ограничение *LIMIT* позволяет задать максимальное число записей, которые могут быть удалены. Следующий запрос удаляет все записи таблицы *orders*, но не более 3 записей.

```
mysql> DELETE FROM orders LIMIT 3;
Query OK, 3 rows affected (0.01 sec)
```

```
mysql> SELECT * FROM orders;
+-----+-----+-----+-----+-----+
| order_ID | o_user_ID | o_book_ID | o_time | o_number |
+-----+-----+-----+-----+-----+
|      4 |      4 |      20 | 2009-03-10 18:20:00 |      1 |
|      5 |      3 |      20 | 2009-03-17 19:15:36 |      1 |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Конструкция *ORDER BY* обычно применяется вместе с ключевым словом *LIMIT*. Например, если необходимо удалить 20 первых записей таблицы, то производится сортировка по полю типа *DATETIME* – тогда в первую очередь будут удалены самые старые записи.

Оператор *TRUNCATE TABLE* полностью очищает таблицу и не допускает условного удаления. Он аналогичен оператору *DELETE* без условия *WHERE* и ограничения *LIMIT*. Удаление происходит гораздо быстрее, т. к. осуществляется не перебор записей, а полное очищение таблицы.

```
mysql> TRUNCATE TABLE orders;
Query OK, 5 rows affected (0.03 sec)
```

```
mysql> SELECT * FROM orders;
Empty set (0.00 sec)
```

Обновление данных. Обновление данных (изменение значений полей в существующих записях) обеспечивают:

- оператор *UPDATE*;
- оператор *REPLACE*.

Оператор *UPDATE* позволяет обновлять отдельные поля в существующих записях. Имеет следующий синтаксис

UPDATE [IGNORE] имя_таблицы

```

SET имя_столбца1 = выражение1 [, имя_столбца2 = выражение2 ... ]
[WHERE условие]
[ORDER BY имя_поля]
[LIMIT число_строк] ;

```

После ключевого слова *UPDATE* указывается таблица, которая изменяется. В предложении *SET* указывается, какие столбцы обновляются и устанавливаются их новые значения. Необязательное условие *WHERE* позволяет задать критерий отбора строк (обновляться будут только строки, удовлетворяющие условию).

Если указывается необязательное ключевое слово *IGNORE*, то команда обновления не будет прервана, даже если при обновлении возникнет ошибка дублирования ключей. Строки, породившие конфликтные ситуации, обновлены не будут.

Запрос, изменяющий в таблице *catalogs* «Сети» на «Компьютерные сети».

```

mysql> UPDATE catalogs SET cat_name='Компьютерные сети'
-> WHERE cat_name='Сети';
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM catalogs;
+-----+-----+
| cat_ID | cat_name          |
+-----+-----+
| 1      | Программирование |
| 2      | Интернет         |
| 3      | Базы данных      |
| 4      | Компьютерные сети |
| 5      | Мультимедиа      |
+-----+-----+
5 rows in set (0.00 sec)

```

Обновлять можно всю таблицу. Пусть требуется уменьшить на 5 % цену на все книги. Для этого следует старую цену в рублях умножить на 0,95.

```
mysql> UPDATE books SET b_price=b_price*0.95;
Query OK, 30 rows affected (0.03 sec)
Rows matched: 30  Changed: 30  Warnings: 0
```

```
mysql> SELECT book_ID, b_name, b_price FROM books;
```

book_ID	b_name	b_price
1	JavaScript в кармане	39.90
2	Visual FoxPro 9.0	627.00
3	C++ Как он есть	207.10
4	Создание приложений с помощью C#	160.55
5	Delphi. Народные советы	230.85
6	Delphi. Полное руководство	475.00
7	Профессиональное программирование на PHP	293.55
8	Совершенный код	732.45
9	Практика программирования	203.30
10	Принципы маршрутизации в Internet	406.60
11	Поиск в Internet	101.65
12	Web-конструирование	168.15
13	Самоучитель Интернет	114.95
14	Популярные интернет-браузеры	77.90
15	Общение в Интернете	80.75
16	Базы данных	309.70
17	Базы данных. Разработка приложений	179.55
18	Раскрытие тайн SQL	190.00
19	Практикум по Access	82.65
20	Компьютерные сети	598.50
21	Сети. Поиск неисправностей	412.30
22	Безопасность сетей	438.90
23	Анализ и диагностика компьютерных сетей	326.80
24	Локальные вычислительные сети	77.90
25	Цифровая фотография	141.55
26	Музыкальный компьютер для гитариста	206.15
27	Видео на ПК	219.45
28	Мультипликация во Flash	200.45
29	Запись CD и DVD	158.65
30	Запись и обработка звука на компьютере	48.45

```
30 rows in set (0.00 sec)
```

Инструкции *LIMIT* и *ORDER BY* позволяют ограничить число изменяемых записей. При этом за один запрос можно обновить несколько столбцов таблицы. Например, необходимо в таблице *books* для десяти самых дешевых товарных позиций уменьшить количество книг на складе на единицу, а цену – на 5 %.

```
mysql> UPDATE books SET b_price=b_price*0.95,b_count=b_count-1
-> ORDER BY b_price LIMIT 10;
Query OK, 10 rows affected (0.05 sec)
Rows matched: 10  Changed: 10  Warnings: 0
```

```
mysql> SELECT book_ID, b_name, b_price, b_count FROM books;
```

book_ID	b_name	b_price	b_count
1	JavaScript в кармане	39.90	9
2	Visual FoxPro 9.0	660.00	2
3	C++ Как он есть	218.00	4
4	Создание приложений с помощью C#	169.00	1
5	Delphi. Народные советы	243.00	6
6	Delphi. Полное руководство	500.00	6
7	Профессиональное программирование на PHP	309.00	5
8	Совершенный код	771.00	1
9	Практика программирования	214.00	12
10	Принципы маршрутизации в Internet	428.00	4
11	Поиск в Internet	101.65	1
12	Web-конструирование	177.00	6
13	Самоучитель Интернет	114.95	3
14	Популярные интернет-браузеры	77.90	5
15	Общение в Интернете	80.75	4
16	Базы данных	326.00	2
17	Базы данных. Разработка приложений	189.00	6
18	Раскрытие тайн SQL	200.00	3
19	Практикум по Access	82.65	5
20	Компьютерные сети	630.00	6
21	Сети. Поиск неисправностей	434.00	4
22	Безопасность сетей	462.00	5
23	Анализ и диагностика компьютерных сетей	344.00	3
24	Локальные вычислительные сети	77.90	7
25	Цифровая фотография	141.55	19
26	Музыкальный компьютер для гитариста	217.00	15
27	Видео на ПК	231.00	10
28	Мультипликация во Flash	211.00	20
29	Запись CD и DVD	158.65	11
30	Запись и обработка звука на компьютере	48.45	7

```
30 rows in set (0.00 sec)
```

Оператор *REPLACE* работает как оператор *INSERT*, за исключением того, что старая запись с тем же значением индекса *UNIQUE* или *PRIMARY KEY* перед внесением новой будет удалена. Если не используются индексы *UNIQUE* или *PRIMARY KEY*, то применение оператора *REPLACE* не имеет смысла.

Синтаксис оператора *REPLACE* аналогичен синтаксису оператора *INSERT*:

```
REPLACE[INTO] имя_таблицы [(имя_столбца, ...)]
VALUES (выражение, ...)
```

В таблицу вставляются значения, определяемые в списке после ключевого слова *VALUES*. Задать порядок столбцов можно при помощи необязательного списка, следующего за именем таблицы. Как и оператор *INSERT*, оператор *REPLACE* допускает многострочный формат.

Практическая работа

При выполнении лабораторной работы необходимо для заданной предметной области средствами MySQL:

- заполнить согласованными данными таблицы БД;
- при необходимости исправить введенную информацию;
- составить отчет по лабораторной работе.

Пример выполнения работы

Операторы заполнения БД *book* имеют следующий вид.


```

USE book;
SET CHARACTER SET cp1251;

DELETE FROM catalogs;
INSERT INTO catalogs VALUES (1,'Программирование');
INSERT INTO catalogs VALUES (2,'Интернет');
INSERT INTO catalogs VALUES (3,'Базыданных');
INSERT INTO catalogs VALUES (4,'Сети');
INSERT INTO catalogs VALUES (5,'Мультимедиа');

DELETE FROM books;
INSERT INTO books VALUES (1,'JavaScript вкармане','РеваО.Н.', 2008, 42.00, 10, 1);
INSERT INTO books VALUES (2,'Visual FoxPro 9.0','КлепининВ.Б.', 2007, 660.00, 2, 1);
INSERT INTO books VALUES (3,'C++ Каконесть','ТимофеевВ.В.',2009, 218.00, 4, 1);
INSERT INTO books VALUES (4,'Создание приложений с помощью C#','Фаронов В.В.',
2008, 169.00, 1, 1);
INSERT INTO books VALUES (5,'Delphi. Народные советы','Шкрыль
А.А.',2007,243.00,6,1);
INSERT INTO books VALUES (6,'Delphi. Полное руководство','Сухарев
М.',2008,500.00,6,1);
INSERT INTO books VALUES (7,'Профессиональное программирование на PHP',
'Шлоснейгл Дж.', 2006, 309.00, 5, 1);
INSERT INTO books VALUES (8,'Совершенныйкод','МакконнеллС.', 2007, 771.00, 1, 1);
INSERT INTO books VALUES (9,'Практика программирования','Керниган Б.', 2004,
214.00, 12, 1);
INSERT INTO books VALUES (10,'Принципымаршрутизациив Internet','ХелебиС.', 2001,
428.00, 4, 2);
INSERT INTO books VALUES (11,'Поискв Internet','ГусевВ.С.',2004,107.00,2,2);
INSERT INTO books VALUES (12,'Web-конструирование','ДувановА.А.', 2003, 177.00, 6,
2);
INSERT INTO books VALUES (13,'СамоучительИнтернет','КонстантиновЮ.П.', 2009,
121.00, 4, 2);
INSERT INTO books VALUES (14,'Популярныеинтернет-браузеры','МарининС.А.',
2007, 82.00, 6, 2);
INSERT INTO books VALUES (15,'ОбщениевИнтернете','ЭклераА.', 2006, 85.00, 5, 2);
INSERT INTO books VALUES (16,'Базыданных','МалыхинаМ.П.', 2006, 326.00, 2, 3);
INSERT INTO books VALUES (17,'Базыданных. Разработка приложений','Рудикова
Л.В.', 2006, 189.00, 6, 3);
INSERT INTO books VALUES (18,'Раскрытиетайн SQL','ОнпельЭ.', 2007, 200.00, 3, 3);
INSERT INTO books VALUES (19,'Практикумпн Access','ЗолотоваС.И.', 2007, 87.00, 6,
3);
INSERT INTO books VALUES (20,'Компьютерныесети','ТанненбаумЭ.', 2007, 630.00, 6,
4);
INSERT INTO books VALUES (21,'Сети. Поиск неисправностей','Бигелоу С.', 2005,
434.00, 4, 4);
INSERT INTO books VALUES (22,'Безопасностьсетей','БреггР.', 2006, 462.00, 5, 4);
INSERT INTO books VALUES (23,'Анализ и диагностика компьютерных сетей','Хогдал
Дж.', 2001, 344.00, 3, 4);
INSERT INTO books VALUES (24,'Локальные вычислительные сети','Епанешников А.',
2005, 82.00, 8, 4);
INSERT INTO books VALUES (25,'Цифроваяфотография','НадеждинН.', 2004, 149.00,
20,5);

```

```

INSERT INTO books VALUES (26,'Музыкальный компьютер для гитариста', 'Петелин
Р.Ю.', 2004, 217.00, 15, 5);
INSERT INTO books VALUES (27,'Видео на ПК', 'Федорова А.', 2003, 231.00, 10, 5);
INSERT INTO books VALUES (28,'Мультипликация во Flash', 'Киркпатрик Г.', 2006,
211.00, 20, 5);
INSERT INTO books VALUES (29,'Запись CD и DVD', 'Гульятеев А.К.', 2003, 167.00, 12,
5);
INSERT INTO books VALUES (30,'Запись и обработка звука на компьютере', 'Лоянич
А.А.', 2008, 51.00, 8, 5);

DELETE FROM users;
INSERT INTO users VALUES (1,'Александр','Валерьевич','Иванов','58-98-78',
'ivanov@email.ru', 'active');
INSERT INTO users VALUES (2,'Сергей','Иванович','Лосев','90-57-77', 'losev@email.ru',
'passive');
INSERT INTO users VALUES (3,'Игорь','Николаевич','Симонов','95-66-61',
'simonov@email.ru', 'active');
INSERT INTO users VALUES (4,'Максим','Петрович','Кузнецов',NULL,
'kuznetsov@email.ru', 'active');
INSERT INTO users VALUES (5,'Анатолий','Юрьевич','Петров', NULL, NULL, 'lock');
INSERT INTO users VALUES (6,'Александр','Александрович','Корнеев','89-78-36',
'korneev@email.ru', 'gold');

DELETE FROM orders;
INSERT INTO orders VALUES (1,3,8,'2009-01-04 10:39:38',1);
INSERT INTO orders VALUES (2,6,10,'2009-02-10 09:40:29',2);
INSERT INTO orders VALUES (3,1,20,'2009-02-18 13:41:05',4);
INSERT INTO orders VALUES (4,4,20,'2009-03-10 18:20:00',1);
INSERT INTO orders VALUES (5,3,20,'2009-03-17 19:15:36',1);

```

Требования к оформлению отчета

1. Титульный лист.
2. Название работы.
3. Тему, цель и задание к работе.
4. Снимки экрана (скриншоты) процесса разработки.
5. Выводы (что узнали, где можно применить полученные знания).

Лабораторная работа №12-13.Создание транзакции.

Управление транзакциями.

Цель работы: Изучение механизма работы транзакций в MySQL.

Теоретические сведения

Изменения БД часто требуют выполнения нескольких запросов, например при покупке в электронном магазине требуется добавить запись в таблицу заказов и уменьшить число товарных позиций на складе. В промышленных БД одно событие может затрагивать большее число таблиц и требовать многочисленных запросов.

Если на этапе выполнения одного из запросов происходит сбой, это может нарушить целостность БД (товар может быть продан, а число товарных позиций на складе не обновлено). Чтобы сохранить целостность БД, все изменения должны выполняться как единое целое. Либо все изменения успешно выполняются, либо, в случае сбоя, БД принимает состояние, которое было до начала изменений. Это обеспечивается средствами обработки транзакций.

Транзакция – последовательность операторов SQL, выполняющихся как единая операция, которая не прерывается другими клиентами. Пока происходит работа с записями таблицы (обновление или удаление), никто другой не может получить доступ к этим записям, т. к. MySQL автоматически блокирует доступ к ним.

Таблицы *ISAM*, *MyISAM* и *HEAP* не поддерживают транзакции. В настоящий момент их поддержка осуществляется только в таблицах *BDB* и *InnoDB*.

Транзакции позволяют объединять операторы в группу и гарантировать, что все операторы группы будут выполнены успешно. Если часть транзакции выполняется со сбоем, результаты выполнения всех операторов транзакции до места сбоя отменяются, приводя БД к виду, в котором она была до выполнения транзакции.

По умолчанию MySQL работает в режиме автоматического завершения транзакций, т. е. как только выполняется оператор обновления данных, который модифицирует таблицу, изменения тут же сохраняются на диске. Чтобы объединить операторы в транзакцию, следует отключить этот режим: *SETAUTOCOMMIT=0*;

После отключения режима для завершения транзакции необходимо ввести оператор *COMMIT*, для отката – *ROLLBACK*.

Включить режим автоматического завершения транзакций для отдельной последовательности операторов можно оператором *START TRANSACTION*.

Для таблиц *InnoDB* есть операторы *SAVEPOINT* и *ROLLBACK TO SAVEPOINT*, которые позволяют работать с именованными точками начала транзакции.

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO catalogs VALUES(NULL,'Периферия');
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT point1;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO catalogs VALUES(NULL,'Разное');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM catalogs;
+-----+-----+
| cat_ID | cat_name |
+-----+-----+
|      1 | Программирование |
|      2 | Интернет |
|      3 | Базы данных |
|      4 | Сети |
|      5 | Мультимедиа |
|     12 | Периферия |
|     13 | Разное |
+-----+-----+
7 rows in set (0.00 sec)
```

```
mysql> ROLLBACK TO SAVEPOINT point1;
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> SELECT * FROM catalogs;
+-----+-----+
| cat_ID | cat_name |
+-----+-----+
|      1 | Программирование |
|      2 | Интернет |
|      3 | Базы данных |
|      4 | Сети |
|      5 | Мультимедиа |
|     12 | Периферия |
+-----+-----+
6 rows in set (0.00 sec)
```

Оператор *SAVEPOINT* устанавливает именованную точку начала транзакции с именем *point1*. Оператор *ROLLBACK TO SAVEPOINT point1* откатывает транзакцию к состоянию, в котором находилась БД на момент установки именованной точки. Все точки сохранения транзакций удаляются, если выполняются операторы *COMMIT* или *ROLLBACK* без указания имени точки сохранения.

Практическая работа

При выполнении лабораторной работы необходимо:

- создать транзакцию, произвести ее откат и фиксацию;
- составить отчет по лабораторной работе.

Пример выполнения работы

Для выполнения задания объединим несколько операций по добавлению в таблицу *catalogs* новых каталогов, а затем произведем откат транзакции, т. е. отмену произведенных действий. Отключаем режим автоматического завершения, добавляем новые записи и проверяем, добавились записи или нет.

```
mysql> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO catalogs VALUES(NULL,'Аппаратура');
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO catalogs VALUES(NULL,'Безопасность');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM catalogs;
+-----+-----+
| cat_ID | cat_name |
+-----+-----+
| 1      | Программирование |
| 2      | Интернет |
| 3      | Базы данных |
| 4      | Сети |
| 5      | Мультимедиа |
| 6      | Аппаратура |
| 7      | Безопасность |
+-----+-----+
7 rows in set (0.02 sec)
```

Откатываем транзакцию оператором ROLLBACK (изменения не сохранились).

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT * FROM catalogs;
+-----+-----+
| cat_ID | cat_name |
+-----+-----+
| 1      | Программирование |
| 2      | Интернет |
| 3      | Базы данных |
| 4      | Сети |
| 5      | Мультимедиа |
+-----+-----+
5 rows in set (0.00 sec)
```

Воспроизведем транзакцию и сохраним действия оператором COMMIT.

```
mysql> INSERT INTO catalogs VALUES(NULL,'Аппаратура');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO catalogs VALUES(NULL,'Безопасность');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM catalogs;
+-----+-----+
| cat_ID | cat_name |
+-----+-----+
| 1      | Программирование |
| 2      | Интернет |
| 3      | Базы данных |
| 4      | Сети |
| 5      | Мультимедиа |
| 8      | Аппаратура |
| 9      | Безопасность |
+-----+-----+
7 rows in set (0.00 sec)
```

```
mysql> COMMIT;  
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SELECT * FROM catalogs;
```

cat_ID	cat_name
1	Программирование
2	Интернет
3	Базы данных
4	Сети
5	Мультимедиа
8	Аппаратура
9	Безопасность

Изолированность транзакций

Таблицы InnoDB предлагают четыре различных уровня изолированности транзакций. В порядке от самого сильного к слабому, уровни изолированности могут быть следующими:

- упорядочение (serializable);
- повторяемое чтение (repeatable read);
- чтение подтвержденного (read committed);
- чтение неподтвержденного (read uncommitted).

Как и для многих других опций, здесь тоже приходится выбирать между устойчивостью и производительностью.

Упорядочение (serializable) — это идеал с точки зрения чистоты и устойчивости. С упорядочением чтение и запись в базе данных должны казаться выполняющимися по очереди, когда записываемые изменения вносятся полностью до начала последующего чтения. При этом транзакции не всегда будут выполняться в непрерывающейся последовательности — многие транзакции не мешают одна другой, но в случае коллизии будут мешать. Блокировка и ожидание вместе с непроизводительными усилиями на предсказание возможных коллизий превращают упорядочение в самый медленный режим изолированности. Если вы захотите использовать этот режим, воспользуйтесь следующей командой:

```
set transaction isolation level serializable;
```

Для InnoDB уровнем изолированности по умолчанию является повторяемое чтение (repeatable read). В этом режиме изолированности каждая транзакция работает в изолированной версии таблицы, где каждая строка остается в том виде, в котором она находилась перед началом транзакции. При чтении любой строки гарантируется повторяемость результата.

Если вызвать

```
select * from account where number=1;
```

в начале транзакции и выполнять тот же запрос в рамках транзакции позже, вы получите один и тот же результат. Тем не менее существует опасность так называемого фантомного чтения. Возможно, что другая транзакция, которая была только что зафиксирована, добавила в таблицу новые строки. Если выполнить один и тот же запрос с некоторым условием дважды, например

```
select * from account where balance>1000;
```

то вполне возможно, что во второй раз вы получите новые строки — фантомные.

На практике фантомное чтение в MySQL должно наблюдаться исключительно редко. Для решения этой проблемы механизм InnoDB использует алгоритм, называемый

блокировкой следующего ключа, но столбец, который используется в соответствующем условии, должен быть индексирован. Вы, вероятно, уже знаете, что InnoDB предлагает блокировку на уровне строк. Когда транзакция использует строку, она блокирует ее, чтобы транзакция могла изолироваться от других. Вместе с такой блокировкой строк блокировка следующего ключа заблокирует и пустоты между строками, найденные в индексе. В результате такого подхода к разрешению проблемы фантомного чтения лишь небольшому числу систем действительно может понадобиться режим упорядочения для изолированности.

```
set transaction isolation level repeatable read;
```

Если установить для сервера режим чтение подтвержденного ([readcommitted](#)), транзакции уже не будут слишком изолированными. Если выполнить запрос и повторить его позже в рамках той же транзакции, во второй раз вы получите другой результат, если за это время другая транзакция изменит данные и будет зафиксирована. Если вам потребуется установить этот режим, соответствующая команда должна выглядеть так:

```
set transaction isolation level read committed;
```

На самом слабом уровне изолированности, в режиме чтения неподтвержденного ([readuncommitted](#)), очевидно уже не только то, что транзакции больше не изолированы, не обеспечивают целостность и, таким образом, соответствие ACID, но и то, что транзакций по сути вообще иметь невозможно. В этом режиме транзакции могут читать изменения, которые вносят другие транзакции, до того, как эти изменения будут подтверждены (т.е. зафиксированы). Это называют "грязным" чтением. Вы можете допустить это только при исключительно необычных условиях, например, когда вы знаете, что все активные потоки будут либо только читать, либо только записывать данные, но не то и другое одновременно. Чтобы установить режим чтения неподтвержденного, используйте следующую команду:

```
settransactionisolationlevelreaduncommitted;
```

Характеристика уровней изолированности транзакций

	“Грязное” чтение	Неповторяемое чтение	Фантомное чтение
<i>Чтение неподтвержденного (read uncommitted)</i>	Возможно	Возможно	Возможно
<i>Чтение подтвержденного (read committed)</i>	Невозможно	Возможно	Возможно
<i>Повторяемое чтение (repeatable read)</i>	Невозможно	Невозможно	Возможно (но маловероятно)
<i>Упорядочение (serializable)</i>	Невозможно	Невозможно	Невозможно

Задание для самостоятельного выполнения. Исследовать по приведенному алгоритму механизм транзакций в СУБД MySQL.

Требования к оформлению отчета

1. Титульный лист.
2. Название работы.
3. Тему, цель и задание к работе.
4. Снимки экрана (скриншоты) процесса разработки.

5. Выводы (что узнали, где можно применить полученные знания).

Лабораторная работа №14. Написание триггеров.

Цель работы: Изучение механизма работы триггеров в MySQL.

Теоретические сведения

Триггер – это та же хранимая процедура, но привязанная к событию изменения содержимого конкретной таблицы.

Возможны три события, связанных с изменением содержимого таблицы, к которым можно привязать триггер:

- *INSERT* – вставка новых данных в таблицу;
- *DELETE* – удаление данных из таблицы;
- *UPDATE* – обновление данных в таблице.

Например, при оформлении нового заказа, т. е. при добавлении новой записи в таблицу *orders*, можно создать триггер, автоматически вычитающий число заказанных товарных позиций в таблице *books*.

Создание триггеров

Создать новый триггер позволяет оператор:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
```

```
ON tbl_name FOR EACH ROW trigger_stmt ;
```

Оператор создает триггер с именем *trigger_name*, привязанный к таблице *tbl_name*. Не допускается привязка триггера к временной таблице или представлению. Конструкция *trigger_time* указывает момент выполнения триггера и может принимать два значения:

- *BEFORE* – действия триггера производятся до выполнения операции изменения таблицы;
- *AFTER* – действия триггера производятся после выполнения операции изменения таблицы.

Конструкция *trigger_event* показывает, на какое событие должен реагировать триггер, и может принимать три значения:

- *INSERT* – триггер привязан к событию вставки новой записи в таблицу;
- *UPDATE* – триггер привязан к событию обновления записи таблицы;
- *DELETE* – триггер привязан к событию удаления записей таблицы.

Для таблицы *tbl_name* может быть создан только один триггер для каждого из событий *trigger_event* и момента *trigger_time*. Таким образом, для каждой из таблиц может быть создано всего шесть триггеров.

Конструкция *trigger_stmt* представляет тело триггера – оператор, который необходимо выполнить при возникновении события *trigger_event* в таблице *tbl_name*.

Если требуется выполнить несколько операторов, то необходимо использовать составной оператор *BEGIN ... END*. Синтаксис и допустимые операторы такие же, как и у хранимых процедур. Внутри составного оператора *BEGIN ... END* допускаются все специфичные для хранимых процедур операторы и конструкции:

- другие составные операторы *BEGIN ... END*;
- операторы управления потоком (*IF*, *CASE*, *WHILE*, *LOOP*, *REPEAT*, *LEAVE*, *ITERATE*);
- объявления локальных переменных при помощи оператора *DECLARE* и назначение им значений при помощи оператора *SET*;
- именованные условия и обработчики ошибок.

В MySQL триггеры нельзя привязать к каскадному обновлению или удалению записей из таблицы типа *InnoDB* по связи первичный ключ/внешний ключ.

Триггеры сложно использовать, не имея доступа к новым записям, которые вставляются в таблицу, или старым записям, которые обновляются или удаляются. Для доступа к новым и старым записям используются префиксы *NEW* и *OLD* соответственно. Если в таблице обновляется поле *total*, то получить доступ к старому значению можно по имени *OLD.total*, а к новому – *NEW.total*.

Пример простейшего триггера для учебной БД *book* см. в пункте «Пример выполнения работы» (пример 1). Он демонстрирует работу триггеров после добавления записи в таблицу без вмешательства в запрос. Рассмотрим триггер, который будет включаться до вставки новых записей в таблицу *orders* и ограничивает число заказываемых товаров до 1:

```
mysql> CREATE TRIGGER restrict_count BEFORE INSERT ON orders
-> FOR EACH ROW
-> BEGIN
-> SET NEW.o_number=1;
-> END//
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> INSERT INTO orders VALUES (NULL,1,2,NOW(),10)//
Query OK, 1 row affected (0.01 sec)
```

```
mysql> SELECT * FROM orders WHERE orderID = LAST_INSERT_ID()//
```

orderID	o_userID	o_bookID	o_time	o_number
16	1	2	2009-10-23 20:26:19	1

```
1 row in set (0.00 sec)
```

Часто при обновлении полей таблицы производится попытка добавления некорректных значений. Пример триггера, который при добавлении нового покупателя преобразует полные имена и отчества в инициалы, см. в пункте «Пример выполнения работы» (пример 2). Он привязан к событию *INSERT*. Чтобы имя и отчество не могло быть отредактировано при помощи оператора *UPDATE*, можно создать триггер, привязанный к событию *UPDATE*.

Удаление триггеров. Удалить существующий триггер позволяет оператор *DROP TRIGGER trigger_name*;

Практическая работа

При выполнении лабораторной работы необходимо:

- для заданной предметной области написать два триггера для разных таблиц базы данных;

- составить отчет по лабораторной работе.

Пример выполнения работы

1. Создадим триггер, который при оформлении нового заказа (при добавлении новой записи в таблицу *orders*) будет увеличивать на 1 значение пользовательской переменной *@tot*.

```
mysql> delimiter //
mysql> CREATE TRIGGER sub_count AFTER INSERT ON orders
-> FOR EACH ROW
-> BEGIN
-> SET @tot =@tot+1;
-> END//
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT @tot //
+-----+
| @tot |
+-----+
| NULL |
+-----+
1 row in set (0.00 sec)
```

Для корректной работы триггера необходимо, чтобы пользовательская переменная *@tot* имела значение, отличное от *NULL*, т. к. операция сложения с *NULL* также приводит к *NULL*.

```
mysql> SET @tot=5//
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO orders VALUES (NULL,1,5,NOW(),10)//
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @tot//
+-----+
| @tot |
+-----+
| 6     |
+-----+
1 row in set (0.00 sec)
```

2. Создадим триггер, который при добавлении новых покупателей преобразует имена и отчества покупателей в инициалы.

```
mysql> CREATE TRIGGER restrict_user BEFORE INSERT ON users
-> FOR EACH ROW
-> BEGIN
-> SET NEW.u_name = LEFT(NEW.u_name,1);
-> SET NEW.u_patronymic = LEFT(NEW.u_patronymic,1);
-> END//
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> SELECT u_surname, u_name, u_patronymic FROM users
-> WHERE userID = LAST_INSERT_ID();//
```

u_surname	u_name	u_patronymic
Титова	С	П

```
1 row in set (0.00 sec)
```

Требования к оформлению отчета

1. Титульный лист.
2. Название работы.
3. Тему, цель и задание к работе.
4. Снимки экрана (скриншоты) процесса разработки.
5. Выводы (что узнали, где можно применить полученные знания).

Лабораторная работа №15-16. *Написание хранимых процедур простой структуры.*

Написание хранимых процедур циклической структуры.

Цель работы: Изучение механизма работы хранимых процедур в MySQL.

Теоретические сведения

На практике часто требуется повторять последовательность одинаковых запросов. Хранимые процедуры позволяют объединить последовательность таких запросов и сохранить их на сервере. После этого клиентам достаточно послать один запрос на выполнение хранимой процедуры.

Хранимые процедуры обладают следующими преимуществами.

- *Повторное использование кода* – после создания хранимой процедуры ее можно вызывать из любых приложений и SQL-запросов.
- *Сокращение сетевого трафика* – вместо нескольких запросов экономнее послать серверу запрос на выполнение хранимой процедуры и сразу получить ответ.
- *Безопасность* – действия не приведут к нарушению целостности данных, т.к. для выполнения хранимой процедуры пользователь должен иметь привилегию.
- *Простота доступа* – хранимые процедуры позволяют инкапсулировать сложный код и оформить его в виде простого вызова.
- *Выполнение бизнес-логики* – хранимые процедуры позволяют перенести код сохранения целостности БД из прикладной программы на сервер БД. Бизнес-логика в виде хранимых процедур не зависит от языка разработки приложения.

Создание хранимых процедур. Реализуется оператором

CREATEPROCEDURE *имя_процедуры* ([*параметр* [, ...]])

[*характеристика ...*] *тело_процедуры*

В скобках передается необязательный список параметров, перечисленных через запятую. Каждый параметр позволяет передать в процедуру (из процедуры) входные данные (результат работы) и имеет следующий синтаксис:

[*IN* / *OUT* / *INOUT*] *имя_параметра тип*

Ключевые слова *IN*, *OUT*, *INOUT* задают направление передачи данных:

- *IN* – данные передаются строго внутрь хранимой процедуры; если параметру с данным модификатором присваивается новое значение, при выходе из процедуры оно не сохраняется и параметр принимает значение, которое он имел до вызова;
- *OUT* – данные передаются строго из хранимой процедуры, если параметр имеет какое-то начальное значение, то внутри хранимой процедуры это значение во внимание не принимается;
- *INOUT* – значение этого параметра как принимается во внимание внутри процедуры, так и сохраняет свое значение при выходе из нее.

Список аргументов, заключенных в круглые скобки, присутствует всегда. Если аргументы отсутствуют, следует использовать пустой список. Если ни один из модификаторов не указан, считается, что параметр объявлен с ключевым словом *IN*.

Телом процедуры является составной оператор *BEGIN ... END*, внутри которого могут располагаться другие операторы:

[*label*:] *BEGIN*

statements

END [*label*]

Оператор, начинающийся с необязательной метки *label* (любое уникальное имя), может заканчиваться выражением *ENDlabel*. Внутри составного оператора *BEGIN ... END* может находиться другой составной оператор. Если хранимая процедура содержит один запрос, то составной оператор можно не использовать.

При работе с хранимыми процедурами символ точки с запятой в конце запроса воспринимается консольным клиентом как сигнал к отправке запроса на сервер. Поэтому следует переопределить разделитель запросов – например, вместо точки с запятой использовать последовательность `//` :

```
mysql> DELIMITER //
mysql> SELECT VERSION< >//
+-----+
| VERSION< > |
+-----+
| 5.0.51b-community-nt |
+-----+
1 row in set (0.00 sec)
```

Пример создания простейшей хранимой процедуры:

```
mysql> CREATE PROCEDURE my_version<
-> BEGIN
-> SELECT VERSION<;
-> END //
Query OK, 0 rows affected (0.00 sec)
```

Чтобы вызвать хранимую процедуру, необходимо применить оператор *CALL*, после которого помещается имя процедуры и ее параметры в круглых скобках:

```
mysql> CALL my_version<>//
+-----+
| VERSION< > |
+-----+
| 5.0.51b-community-nt |
+-----+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
```

Рекомендуется избегать использования названий хранимых процедур, совпадающих с именами встроенных функций MySQL. В теле хранимой процедуры можно использовать многострочный комментарий, который начинается с последовательности `/*` и заканчивается последовательностью `*/`. Рассмотрим хранимые процедуры с параметрами. Создадим и вызовем процедуру, которая присваивает пользовательской переменной `@x`

```
mysql> CREATE PROCEDURE set_x<IN value INT>
-> BEGIN
-> SET @x = value;
-> END//
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> CALL set_x<123456>//
Query OK, 0 rows affected (0.00 sec)
```

новое значение:

Через параметр *value* процедуре передается числовое значение 123456, которое она присваивает пользовательской переменной @x. Модификатор *IN* сообщает, что данные передаются внутрь функции. Проверим корректность работы процедуры:

```
mysql> SELECT @x//
+-----+
| @x    |
+-----+
| 123456 |
+-----+
1 row in set (0.00 sec)
```

В отличие от пользовательской переменной @x, которая является глобальной и доступна как внутри хранимой процедуры *set_x()*, так и вне ее, параметры процедуры являются локальными и доступны для использования только внутри нее.

Создадим процедуру *numcatalogs()*, которая подсчитывает число записей в таблице *catalogs* базы данных *book*:

```
mysql> CREATE PROCEDURE numcatalogs(OUT total INT)
-> BEGIN
-> SELECT COUNT(*) INTO total FROM catalogs;
-> END //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL numcatalogs(@a)//
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT @a//
+-----+
| @a    |
+-----+
| 5     |
+-----+
1 row in set (0.00 sec)
```

Хранимая процедура *numcatalogs()* имеет один целочисленный параметр *total*, в котором сохраняется число записей в таблице *catalogs*. Осуществляется это при помощи оператора *SELECT ... INTO ... FROM*. В качестве параметра функции *numcatalogs()* передается пользовательская переменная @a.

Создадим хранимую процедуру *catalogname()*, которая будет возвращать по первичному ключу *catID* название каталога *cat_name*. Для этого потребуется определить параметр *id* с атрибутом *IN*, и *catalog* с атрибутом *OUT*.

```
mysql> CREATE PROCEDURE catalogname(IN id INT, OUT catalog TINYTEXT)
-> BEGIN
-> SELECT cat_name INTO catalog FROM catalogs
-> WHERE catID = id;
-> END //
Query OK, 0 rows affected (0.01 sec)

mysql> SET @id = 5//
Query OK, 0 rows affected (0.00 sec)

mysql> CALL catalogname(@id, @name)//
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT @id, @name//
+-----+-----+
| @id | @name      |
+-----+-----+
| 5   | Мультимедиа |
+-----+-----+
1 row in set (0.00 sec)
```

Операторы управления потоком данных. Хранимые процедуры позволяют реализовать сложную логику с помощью операторов ветвления и циклов. Вне хранимых

процедур эти операторы применять нельзя. Ветвление программы по условию позволяет реализовать оператор:

```
IFлог_выражениеTHEN оператор  
  
[ELSEIFлог_выражениеTHEN оператор] ...  
  
[ELSE оператор]  
  
ENDIF ;
```

Логическое выражение может принимать два значения:

- 0 (ложь);
- значение, отличное от нуля (истина).

Если логическое выражение истинно, то выполняется оператор в блоке *THEN*, иначе выполняется список операторов в блоке *ELSE* (если блок *ELSE* имеется). В логических выражениях можно использовать операторы сравнения (= , > , >= , < , <= , <=). Логические выражения можно комбинировать с помощью операторов && (И), а также || (ИЛИ). Если в блоках *IF*, *ELSEIF* и *ELSE* – два или более операторов, необходимо использовать составной оператор *BEGIN ... END*.

Множественный выбор позволяет осуществить оператор:

```
CASE выражение  
  
WHEN значение THEN оператор  
  
[WHEN значение THEN оператор] ...  
  
[ELSE оператор]  
  
ENDCASE ;
```

Выражение сравнивается со значениями. Как только найдено соответствие, выполняется соответствующий оператор. Если соответствия не найдены, выполняется оператор, размещенный после ключевого слова *ELSE* (если оно присутствует).

В MySQL имеется несколько операторов, позволяющих реализовать циклы. Первый оператор цикла имеет следующий синтаксис:

```
[ label: ] WHILE условие DO  
  
операторы  
  
ENDWHILE [ label ] ;
```

Операторы выполняются в цикле, пока истинно условие. При каждой итерации условие проверяется, и если при очередной проверке оно будет ложным (0), цикл завершится. Если условие ложно с самого начала, то цикл не выполнится ни разу. Если в цикле выполняется более одного оператора, не обязательно заключать их в блок *BEGIN ... END*, т. к. эту функцию выполняет сам оператор *WHILE*.

Досрочный выход из цикла обеспечивает оператор:

```
LEAVE label ;
```


Оператор прекращает выполнение блока, помеченного меткой *label* (например, прекращает выполнение цикла по достижении критического числа итераций).

Досрочное прекращение цикла также обеспечивает оператор

ITERATE label ;

В отличие от оператора *LEAVE* оператор *ITERATE* не прекращает выполнение цикла, он лишь выполняет досрочное прекращение текущей итерации. Оператор *LEAVE* эквивалентен оператору *BREAK*, а оператор *ITERATE* эквивалентен оператору *CONTINUE* в С-подобных языках программирования.

Второй оператор цикла имеет следующий вид:

[label:] REPEAT

операторы UNTIL условие END REPEAT [label] ;

Условие проверяется не в начале, а в конце оператора цикла. Таким образом, цикл выполняет по крайней мере одну итерацию независимо от условия. Следует отметить, что цикл выполняется, пока условие ложно. Оператор цикла может быть снабжен необязательной меткой *label*, по которой можно осуществлять досрочный выход из цикла при помощи операторов *LEAVE* и *ITERATE*.

Реализовать бесконечный цикл позволяет оператор

[label:] LOOP

операторы ENDLOOP[label] ;

Цикл *LOOP* (в отличие от операторов *WHILE* и *REPEAT*) не имеет условий выхода. Поэтому данный цикл должен обязательно иметь в составе оператор *LEAVE*.

Осуществлять безусловный переход позволяет оператор

GOTO label ;

Оператор осуществляет переход к оператору, помеченному меткой *label*. Это может быть как оператор *BEGIN*, так и любой из циклов: *WHILE*, *REPEAT* и *LOOP*. Кроме того, метка может быть не привязана ни к одному из операторов, а объявлена при помощи оператора

LABEL label ;

Использовать оператор *GOTO* для реализации циклов не рекомендуется, т. к. обычные циклы гораздо нагляднее и проще поддаются модификации, в них сложнее допустить логическую ошибку.

Удаление хранимых процедур. Для удаления процедур используется оператор

DROP PROCEDURE [IF EXISTS] имя_процедуры ;

Если удаляемой процедуры с таким именем не существует, оператор возвращает ошибку, которую можно подавить, если использовать необязательное ключевое слово *IF EXISTS*.

Обработчики ошибок. При выполнении хранимых процедур могут возникать ошибки. MySQL позволяет каждой возникающей в хранимой процедуре ошибке

назначить свой обработчик, который в зависимости от ситуации и серьезности ошибки может как прекратить, так и продолжить выполнение процедуры.

Для объявления такого обработчика предназначен оператор

DECLARE именованного обработчика HANDLER FOR код_ошибки [, ...] выражение;

Выражение содержит SQL-запрос, который выполняется при срабатывании обработчика. Тип обработчика может принимать одно из трех значений:

- *CONTINUE* – выполнение текущей операции продолжается после выполнения оператора обработчика;
- *EXIT* – выполнение составного оператора *BEGIN ... END*, в котором объявлен обработчик, прекращается;
- *UNDO* – данный вид обработчика в текущей версии не поддерживается.

Обработчик может быть привязан сразу к нескольким ошибкам, для этого их коды следует перечислить через запятую. Код ошибки, для которой будет происходить срабатывание обработчика, может принимать следующие значения:

- *SQLSTATE [VALUE] значение* – значение *SQLSTATE* является пятисимвольным кодом ошибки в шестнадцатеричном формате (стандарт в SQL); примеры кодов – 'HY000', 'HY001', '42000' и т. д.; один код обозначает сразу несколько ошибок;
- *SQLWARNING* – любое предупреждение MySQL; это ключевое слово позволяет назначить обработчик для всех предупреждений; обрабатываются любые события, для которых код *SQLSTATE* начинается с 01;
- *NOT FOUND* – любая ошибка, связанная с невозможностью найти объект (таблицу, процедуру, функцию, столбец и т. п.); обрабатываются любые события, для которых код *SQLSTATE* начинается с 02;
- *SQLEXCEPTION* – ошибки, не охваченные ключевыми словами *SQLWARNING* и *NOT FOUND*;
- *mysql_error_code* – обычные четырехзначные ошибки MySQL, такие как 1020, 1232, 1324 и т. п.;
- именованное условие (см. ниже).

При указании кода ошибки можно использовать не только целочисленные коды, но и именованные условия, которые объявляются при помощи оператора

DECLARE именованное условие CONDITION FOR код ошибки;

Оператор объявляет именованное условие для кода ошибки. Так, для обрабатываемой ошибки 1062 (23000) – дублирование уникального индекса, оператор может выглядеть следующим образом:

DECLARE 'violation' CONDITION FOR SQLSTATE '23000';

DECLARE 'violation' CONDITION FOR 1062;

Первое объявление охватывает все ошибки со статусом 23000, второй вид ошибок более узкий и включает только дублирование уникального индекса.

Курсоры. Если результирующий запрос возвращает одну запись, поместить результаты в промежуточные переменные можно с помощью оператора *SELECT ... INTO ... FROM*. Однако результирующие таблицы чаще содержат несколько записей, и использование такой конструкции приводит к возникновению ошибки 1172: «Результат содержит более чем одну строку».

Избежать ошибки можно, добавив предложение *LIMIT 1* или назначив *CONTINUE*-обработчик ошибок. Однако такая процедура реализует не то поведение, которое ожидает пользователь. Кроме того, существуют ситуации, когда требуется обработать именно многострочную результирующую таблицу.

Например, пусть требуется вернуть записи одной таблицы, отвечающие определенному условию, и на основании этих записей создать новую таблицу. Решить эту задачу можно с помощью курсоров, которые позволяют в цикле просмотреть каждую строку результирующей таблицы запросов. Работа с курсорами похожа на работу с файлами – сначала открытие курсора, затем чтение и после закрытие.

Работа с курсорами происходит по следующему алгоритму:

1. При помощи инструкции *DECLARE курсор CURSORFOR* связывается имя курсора с выполняемым запросом.

2. Оператор *OPEN* выполняет запрос, связанный с курсором, и устанавливает курсор перед первой записью результирующей таблицы.

3. Оператор *FETCH* помещает курсор на первую запись результирующей таблицы и извлекает данные из записи в локальные переменные хранимой процедуры. Повторный вызов оператора *FETCH* приводит к перемещению курсора к следующей записи, и так до тех пор, пока записи в результирующей таблице не будут исчерпаны. Эту операцию удобно осуществлять в цикле.

4. Оператор *CLOSE* прекращает доступ к результирующей таблице и ликвидирует связь между курсором и результирующей таблицей.

Практическая работа

При выполнении лабораторной работы необходимо:

- Создать таблицы Студент(инд_номер, ФИО, стипендия, количество четверок), Преподаватель(таб_номер, ФИО), Предметы(инд_номер, название, количество часов, таб_номер_преподавателя)
- Создать хранимую процедуру, которая будет увеличивать стипендию на заданное входным параметром1 число процентов, при количестве четверок превышающем число заданное входным параметром2.
- Создать хранимую процедуру, которая будет подсчитывать количество предметов ведомых заданным преподавателем.
- составить отчет по лабораторной работе.

Пример выполнения работы

1. Создадим хранимую процедуру, которая выводит число заказов покупателя по вводимому в качестве параметра процедуры коду покупателя.

```
mysql> CREATE PROCEDURE num(OUT total INT, IN user_kod INT)
-> BEGIN
-> SELECT COUNT(*) INTO total FROM orders WHERE o_user_ID=user_kod;
-> END
-> //
Query OK, 0 rows affected (0.00 sec)
```

Параметр *total* является выходным, его значение равно числу заказов покупателя, код которого записывается во входной параметр *user_kod*. Процедура считает все строки, где код клиента совпадает с параметром *user_kod*.

До вызова процедуры присваиваем параметру процедуры значение кода клиента. Затем вызываем процедуру оператором *CALL*. Для вывода результата можно воспользоваться оператором *SELECT*.

```
mysql> SET @user_kod=3//
Query OK, 0 rows affected (0.00 sec)

mysql> CALL num(@total,@user_kod)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @total,@user_kod//
+-----+-----+
| @total | @user_kod |
+-----+-----+
| 2      | 3         |
+-----+-----+
1 row in set (0.00 sec)
```

2. Создадим хранимую процедуру, которая записывает в новую таблицу *fevral* все заказы, сделанные в феврале 2009 г. Предварительно необходимо создать новую пустую таблицу *fevral* со структурой, аналогичной структуре таблицы *orders*.

```
mysql> CREATE TABLE fevral(
-> f_order_ID int(6) NOT NULL,
-> f_o_user_ID int NOT NULL,
-> f_o_book_ID int NOT NULL,
-> f_o_time datetime NOT NULL,
-> f_o_number int(6) NOT NULL,
-> PRIMARY KEY (f_order_ID)
-> )TYPE=InnoDB//
Query OK, 0 rows affected, 1 warning (0.08 sec)
```

Хранимая процедура *ord_fevr()* использует курсор *curf*, который в цикле читает данные из таблицы *orders* и добавляет их в таблицу *fevral*.

```
mysql> CREATE PROCEDURE ord_fevr()
-> BEGIN
-> DECLARE id int;
-> DECLARE _end int DEFAULT 0;
-> DECLARE userID int;
-> DECLARE bookID int;
-> DECLARE tim datetime;
-> DECLARE num int;
-> DECLARE curf CURSOR FOR SELECT * FROM orders
-> WHERE o_time BETWEEN '2009.02.01' AND '2009.02.28';
-> DECLARE CONTINUE HANDLER FOR NOT FOUND SET _end=1;
-> OPEN curf;
-> wet: LOOP
-> FETCH curf INTO id,userID,bookID,tim,num;
-> IF _end THEN LEAVE wet;
-> END IF;
-> INSERT INTO fevral VALUES(id,userID,bookID,tim,num);
-> END LOOP wet;
-> CLOSE curf;
-> END
-> //
Query OK, 0 rows affected (0.03 sec)
```

Вызов процедуры осуществляется оператором *CALL*. Для просмотра результата выполнения процедуры используем полную выборку из таблицы *fevral*.

```
mysql> CALL ord_fevr//
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM fevral//
+-----+-----+-----+-----+-----+
| f_order_ID | f_o_user_ID | f_o_book_ID | f_o_time           | f_o_number |
+-----+-----+-----+-----+-----+
|          2 |          6 |          10 | 2009-02-10 09:40:29 |          2 |
|          3 |          1 |          20 | 2009-02-18 13:41:05 |          4 |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Требования к оформлению отчета

1. Титульный лист.
2. Название работы.
3. Тему, цель и задание к работе.
4. Снимки экрана (скриншоты) процесса разработки.
5. Выводы (что узнали, где можно применить полученные знания).

Министерство образования Нижегородской области
Государственное бюджетное образовательное учреждение
среднего профессионального образования
«Нижегородский радиотехнический колледж»

ОТЧЁТ

по лабораторной работе №_

Тема

Дисциплина **Основы проектирования баз данных**

Выполнил
студент(ка) группы

Проверил
преподаватель

ФИО

г. Арзамас
201_ г.