

федеральное государственное автономное образовательное учреждение
высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ
К ЛАБОРАТОРНЫМ ЗАНЯТИЯМ**

по дисциплине

ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

направление подготовки

**02.03.02 - Фундаментальная информатика и информационные
технологии**

Составитель

к.т.н., доцент кафедры
программных систем
Попова-Коварцева
Дарья Александровна

Самара 2023

Оглавление

Введение	3
Лабораторная работа № 1	5
Лабораторная работа № 2	16
Лабораторная работа № 3	21
Лабораторная работа № 4	30
Лабораторная работа № 5	34
Описание вариантов заданий предметной области	47
Литература	67

Введение

Данный курс лабораторных работ предназначен для студентов специальности 020302 - фундаментальная информатика и информационные технологии.

Целями лабораторного практикума являются:

- приобретение практических навыков анализа и моделирования предметной области;
- ознакомление с работой специализированных CASE-средств проектирования баз данных;
- приобретение практических навыков использования структурированного языка запросов SQL и программируемых объектов баз данных.

Требования к организации и проведению лабораторных занятий

На лабораторных занятиях студент выполняет персональное (индивидуальное) задание соответствующее варианту, что способствует более эффективному формированию практических умений, навыков и компетенций.

Задания выполняются *строго* в соответствии с утвержденным преподавателем графиком.

Выполнение всех заданий является необходимым условием допуска студента к экзамену.

Цикл лабораторных работ необходимо выполнять с использованием СУБД MySQL, в частности с использованием **MySQL Workbench**.

Программное обеспечение цикла лабораторных работ

СУБД **MySQL** — это система управления реляционными базами данных. Программное обеспечение MySQL — это ПО с открытым кодом. На веб-сайте MySQL (<http://www.mysql.com>) представлена самая последняя информация о программном обеспечении MySQL. Код написан на C и C++. СУБД может быть установлена на все операционные системы с работающими потоками Posix и компилятором C++. MySQL поддерживает большое число типов данных столбцов, практически все стандарты функций ANSI SQL, а также программный интерфейс доступа к базам данных ODBC. MySQL позволяет управлять очень большими базами данных (более 50 миллионов записей). Система основана на привилегиях и паролях, за счет чего обеспечивается гибкость и безопасность, а также возможность верификации с удаленного компьютера. Пароли являются защищенными, поскольку при передаче по сети при соединении с сервером они шифруются.

Очевидно, что в настоящее время создание баз данных невозможно без их тщательного проектирования с применением CASE-средств. Использование

современных CASE-средств для проектирования различного рода программных продуктов на сегодняшний день связано не просто с необходимостью облегчить труд разработчика ПО, но и с необходимостью иметь инструмент, позволяющий также стандартизировать процесс разработки, получить наглядные графические модели разрабатываемого ПО, повторно использовать разработки и иметь возможность разделения огромных по объему задач между разработчиками.

При разработке баз данных часто в связи с большим объемом информации, необходимой для хранения, возникают сложности даже с построением логической модели, не говоря уже о физической модели, в которой необходимо учесть возможности конкретной системы управления баз данных (СУБД). В связи с этим многие СУБД создают свои CASE-средства, позволяющие разработчику в значительной степени автоматизировать процессы моделирования, проектирования и реализации баз данных. В настоящий момент в состав пакета разработчика помимо СУБД MySQL входит CASE-средство Workbench, которое также может быть получено на сайте (<http://www.mysql.com>) и установлено. Его использование в значительной мере упрощает и делает более наглядным процесс проектирования баз данных.

MySQL Workbench — инструмент для визуального проектирования баз данных, который позволяет проектировать, моделировать, создавать и эксплуатировать БД. С его помощью можно представить модель базы данных в графическом виде, устанавливая связи между таблицами, восстанавливать структуры уже существующей на сервере БД (обратный инжиниринг), использовать редактор SQL запросов, позволяющий сразу же отправлять их на сервер и получать ответ в виде таблицы, редактировать данные в таблице в визуальном режиме.

Следует заметить, что основные принципы работы с MySQL Workbench остаются неизменными вне зависимости от того, установлена ли СУБД на ОС Windows или ОС Linux. Различия в большей степени касаются коммерческой и свободно распространяемой версий. Некоторые возможности доступны только для коммерческого варианта. Также более новые версии могут незначительно отличаться от предыдущих, например иметь небольшие расхождения в наименовании пунктов меню или экранов. Однако основные подходы разработки и реализации не изменяются.

В цикле лабораторных работ будут рассмотрены основные возможности использования MySQL Workbench для администрирования, проектирования и реализации баз данных, в том числе реализация баз данных традиционным способом при помощи создания таблиц и связей между ними, а также при помощи построения ER-моделей.

Лабораторная работа № 1

Тема: логическое проектирование базы данных (ER-моделирование).

Цель: формирование навыков по исследованию заданной предметной области и умений выделить сущности и атрибуты, созданию ER-модели базы данных.в анализа предметной области и создания базы данных в СУБД MySQL. Приобретение навыков нормализации отношений.

Теоретические сведения

Общие сведения о ER-моделях

При построении баз данных CASE-средства чаще всего ориентированы на два уровня представления модели — логический и физический. Логическая модель данных является универсальной и никак не связана с конкретной реализацией СУБД.

Физическая модель данных, напротив, зависит от конкретной СУБД, фактически являясь отображением системного каталога. В физической модели содержится информация обо всех объектах БД. Поскольку стандартов на объекты БД не существует (например, хотя имеется стандарт основных типов данных, в каждой конкретной СУБД могут использоваться дополнительно и свои типы данных), физическая модель зависит от используемой СУБД. Следовательно, одной и той же логической модели могут соответствовать несколько различных физических моделей.

Создание модели данных, как правило, начинается с проектирования логической модели. Для проектирования могут использоваться различные инструменты, однако в последнее время наиболее часто применяются CASE-средства (Computer Aided Software Engineering) — средства для автоматизации проектирования программных продуктов. Как правило, CASE-средства предлагают построение ER-моделей.

Для создания ER-диаграмм обычно используют одну из двух наиболее распространенных нотаций.

- **IDEF1X** — усовершенствованная версия IDEF1 (методологии структурного анализа для проектирования сложных ИС, разработанный Т. Рэмей (T. Ramey)) и позволяющая разрабатывать концептуальную модель предметной области системы баз данных в форме одной или нескольких ER-диаграмм, эквивалентных отношениям в третьей нормальной форме. Помимо того, что эта нотация стала федеральным стандартом США, она также является стандартом в ряде международных организаций (например, Международный валютный фонд и др.);
- **Information Engineering (IE)**. Нотация, разработанная Мартином (Martin), Финкельштейном (Finkelstein) и др., используется преимущественно в

промышленности.

Заметим, что IDEF1X (см. приложение С) является методом для разработки реляционных баз данных и использует нотацию, специально разработанную для удобного построения концептуальной схемы — универсального представления структуры данных в рамках рассматриваемой предметной области, независимого от конечной реализации базы данных и аппаратной платформы. Использование метода IDEF1X наиболее целесообразно для построения логической структуры базы данных.

Основным преимуществом IDEF1X, по сравнению с другими многочисленными методами разработки реляционных баз данных, является жесткая и строгая стандартизация моделирования.

Как было сказано выше, IDEF1X используется для моделирования реляционных баз данных и имеет в США статус федерального стандарта. Стандарт входит в семейство методологий IDEF (методологии семейства ICAM (Integrated Computer-Aided Manufacturing) для решения задач моделирования сложных систем), позволяющих исследовать структуру, параметры и характеристики производственно-технических и организационно-экономических систем.

Создание ER-диаграммы в MySQLWorkbench

Среда *MySQLWorkbench* предназначена для визуального проектирования баз данных и управления сервером *MySQL*. Для построения моделей предназначена секция DataModeling:



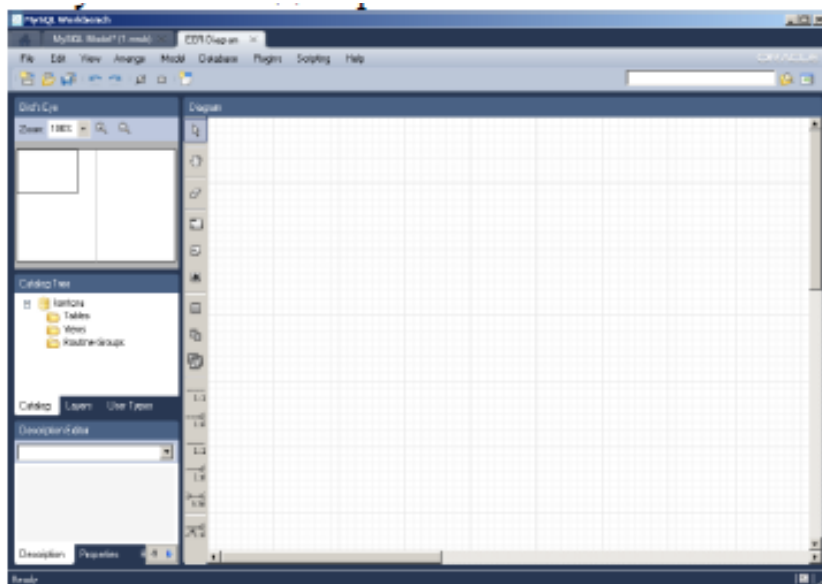
Выберем пункт **Create new EER Model**.


EERmodel расшифровывается как ExtendedEntity-RelationshipModel 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 **Editschema**. 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 **Adddiagram**, 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.

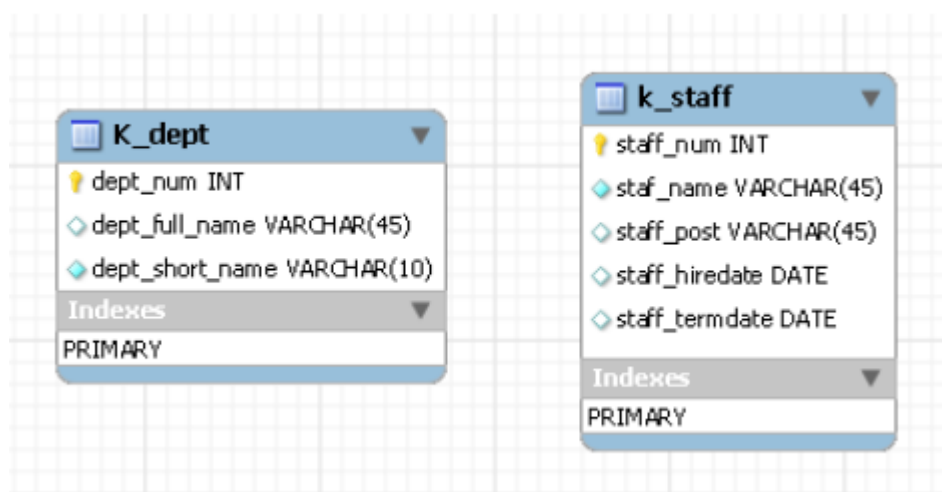
Таблица **Отделы** имеет следующий вид:

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:



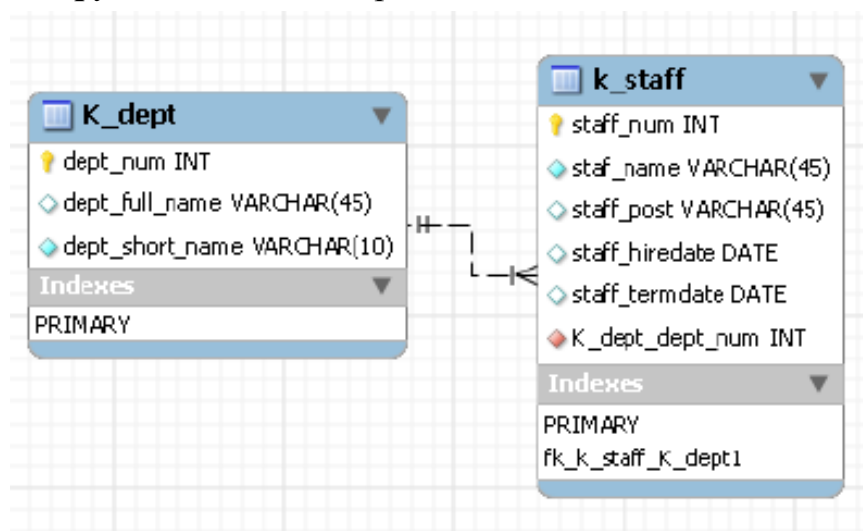
Обратите внимание, что при создании первичного ключа автоматически создается индекс по этому первичному ключу. Индекс представляет собой вспомогательную структуру, которая служит, прежде всего, для ускорения поиска и быстрого доступа к данным.

Теперь свяжем эти таблицы. Сначала создадим связь «Работает» между Сотрудником (дочерняя таблица) и Отделом (родительская таблица), степень связи




М:1. Для создания связей М: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.

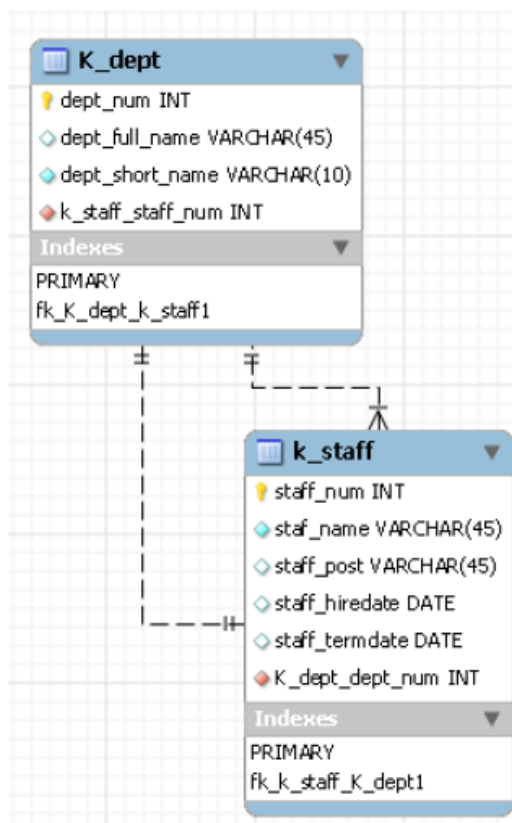
Итак, щелкнем на пиктограмме, затем щелкнем на дочерней таблице **Сотрудники**, затем на родительской таблице **Отделы**:



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.



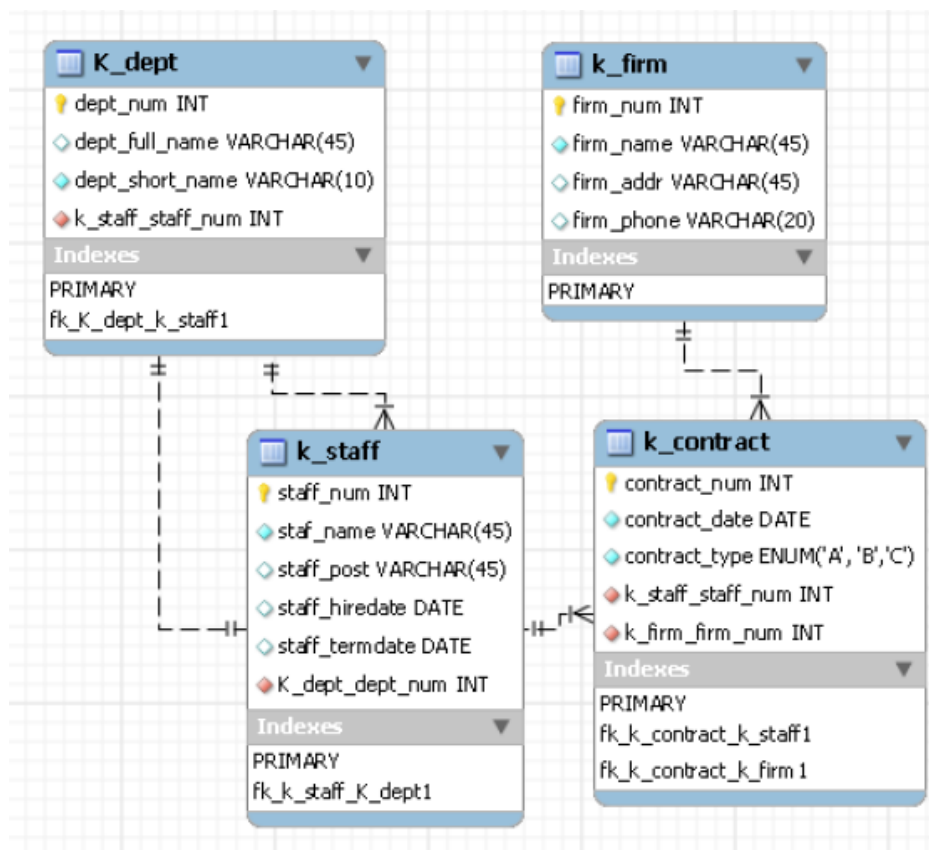
Создадим таблицу **Предприятия**:

k_firm						
Column Name	Datatype	PK	NN	UQ	BIN	UI
firm_num	INT	<input checked="" type="checkbox"/>	<input checked="" 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"/>
firm_addr	VARCHAR(45)	<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"/>

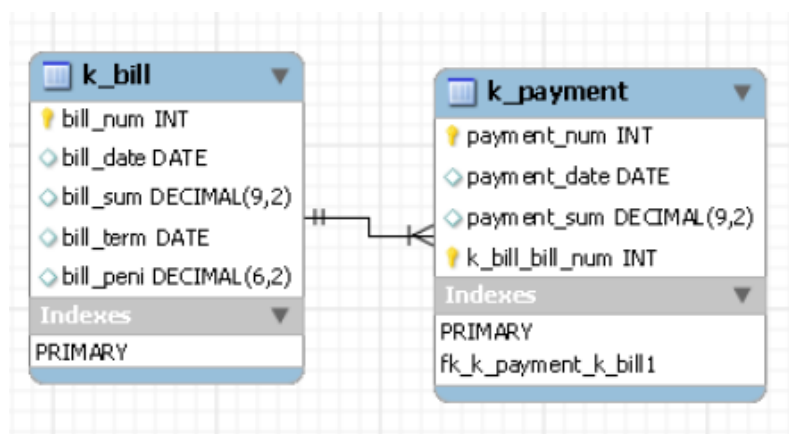
Создадим таблицу **Договоры**. У столбца **Тип_договора** зададим следующий формат: это буква из списка 'A', 'B', 'C'.


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"/>	

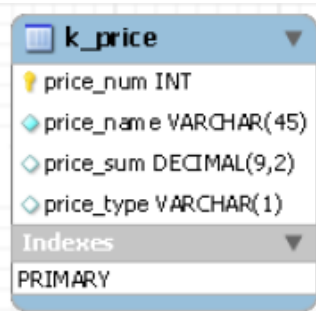
Свяжем **Договоры** с **Сотрудниками** и **Предприятиями** связями **М:1**.




Затем создадим *Счета и Платежи*:

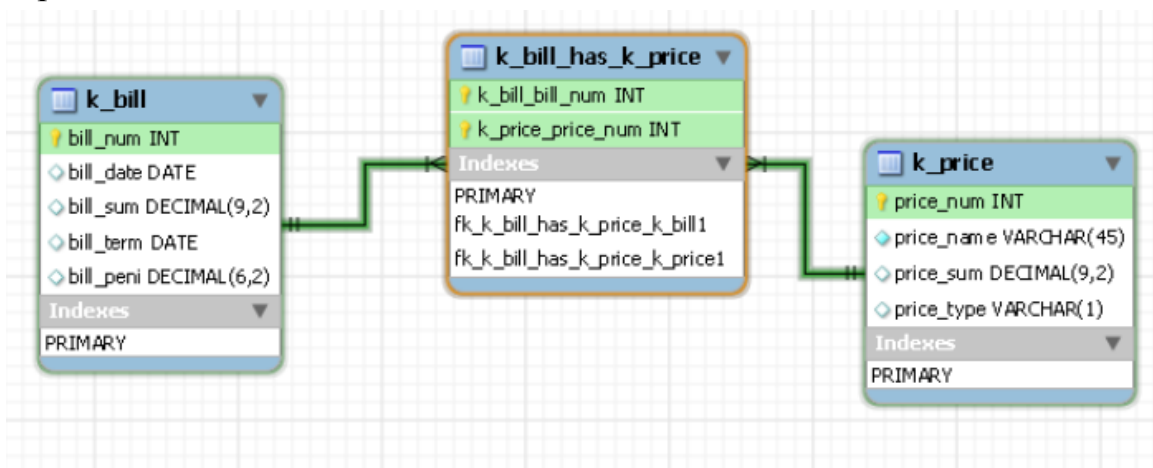


Поскольку сущность *Платеж* была «слабой», у нее нет полноценного первичного ключа, и каждый платеж однозначно идентифицируется группой атрибутов (номер_счета, номер платежа). Отметим в качестве ключевого поля *payment_num*, а затем создадим идентифицирующую связь между *Счетом* и *Платежом*. Идентифицирующая 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).

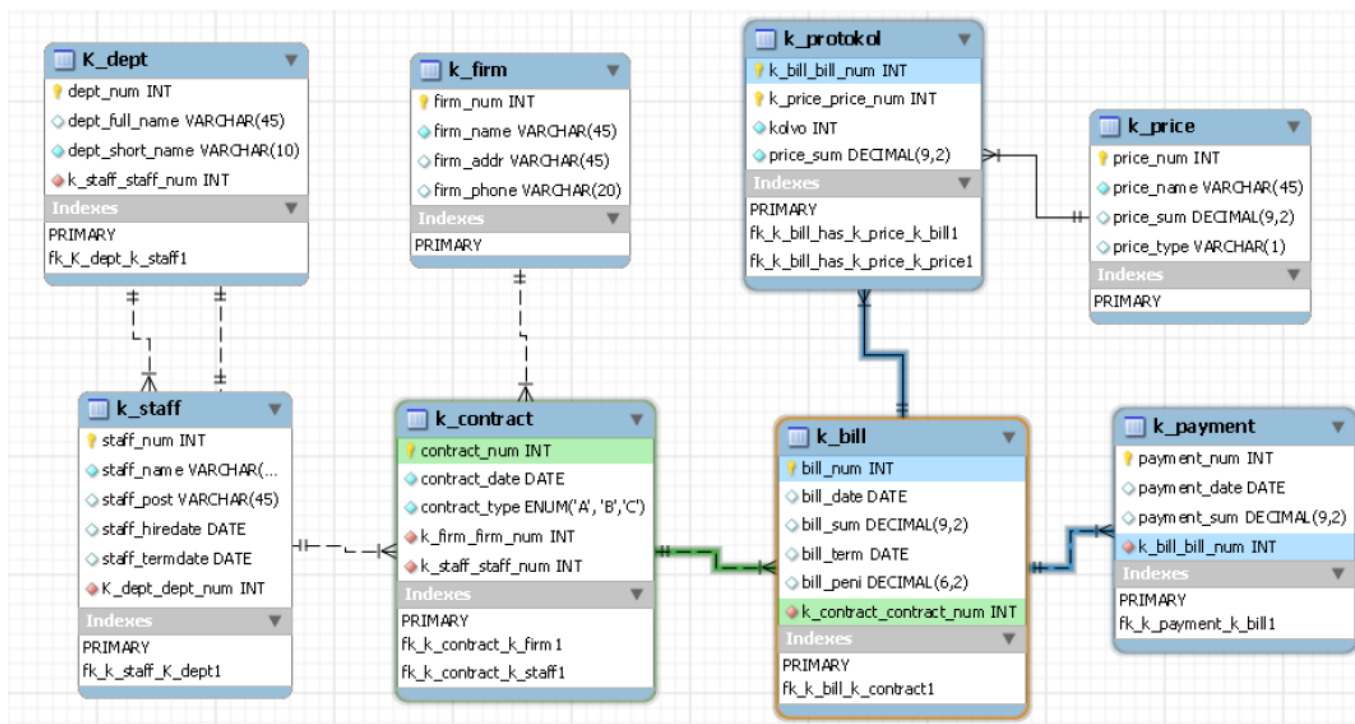
Между объектами **Счет** и **Прайс-лист** имеется связь «многие ко многим». Для создания этой связи нужно использовать пиктограмму . Следует щелкнуть мышью по этой пиктограмме, а затем последовательно щелкнуть по связываемым таблицам. Между ними появится новая таблица, обратите внимание на ее столбцы, 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:



Нормальные формы схем отношений

Нормализация процесс преобразования базы данных к виду, отвечающему нормальным формам. Целью нормализации является получение БД, где каждый факт появляется только один раз. т.е. исключена избыточность, причем к этому стремятся не столько ради экономии памяти, сколько для исключения возможной противоречивости данных.

Нормальная форма - совокупность требований, которым должно удовлетворять отношение

Типы нормальных форм:

- первая нормальная форма (1 NF);
- вторая нормальная форма (2 NF);
- третья нормальная форма (3 NF);
- нормальная форма Бойса-Кодла (BCNF);
- четвертая нормальная форма (4 NF);
- пятая нормальная форма (5 NF).

При решении практических задач в большинстве случаев третья нормальная форма является достаточной. Процесс проектирования реляционной базы данных, как правило, заканчивается приведением к 3 NF. Поэтому мы и рассмотрим подробнее первые три нормальные формы.

Первом нормальная форма (1 NF)

Отношение находится в 1 NF тогда и только тогда, когда все входящие в него атрибуты являются атомарными (неделимыми).

Вторая нормальная форма (2 NF)

Отношение находится в 2 NF, если оно находится в 1 NF и каждый неключевой атрибут функционально полно зависит от первичного ключа

Третья нормальная форма (3 NF)

Отношение находится в 3 NF в том и только в том случае, если оно находится в 2 NF и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Транзитивной зависимостью неключевых атрибутов от ключевых называется следующая: $A \rightarrow B$ и $B \rightarrow C$, где A - набор ключевых атрибутов (ключ). B и C - различные множества неключевых атрибутов.

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле является более ограниченной, но более лучшей, чем предыдущая;
- при переходе к следующей нормальной форме положительные свойства предыдущих нормальных свойств сохраняются.

Порядок выполнения работы №1:

1. Ознакомиться с описанием предметной области варианта задания.
2. Определить необходимые сущности и их атрибуты для предметной области.
3. Указать первичные ключи сущностей.
4. Определить связи между сущностями. При необходимости добавить в проект дополнительные сущности.
5. Привести созданную модель к третьей нормальной форме.
6. В рабочей области MySQL Workbench создать ER-модель в нотации IDEF1X.
7. Сохранить модель.

Лабораторная работа № 2

Тема: создание физической модели базы данных из ER-модели (прямой инжиниринг) и построение SQL-запросов к ней.

Цель: формирование навыков по созданию базы данных в СУБД MySQL.

Порядок выполнения работы №2:

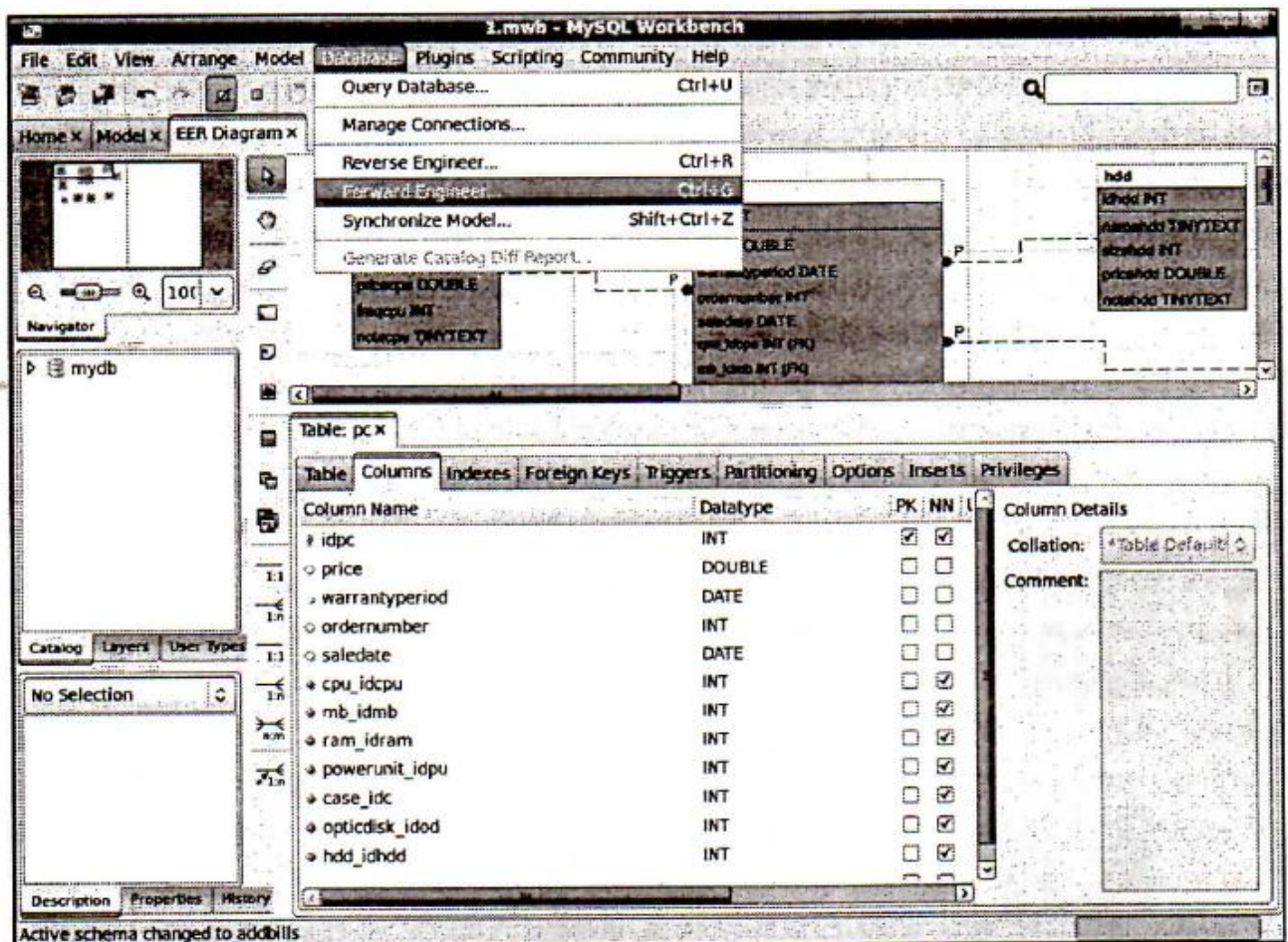
1. С использованием прямого инжиниринга экспортировать ER-модель на MySQL сервер.
2. Написать запросы добавления данных в таблицы (минимум 10 записей для каждой таблицы).
3. Создать SQL-запросы согласно вариантам задания, выполняющие основные требования к функциям системы.

Теоретические сведения

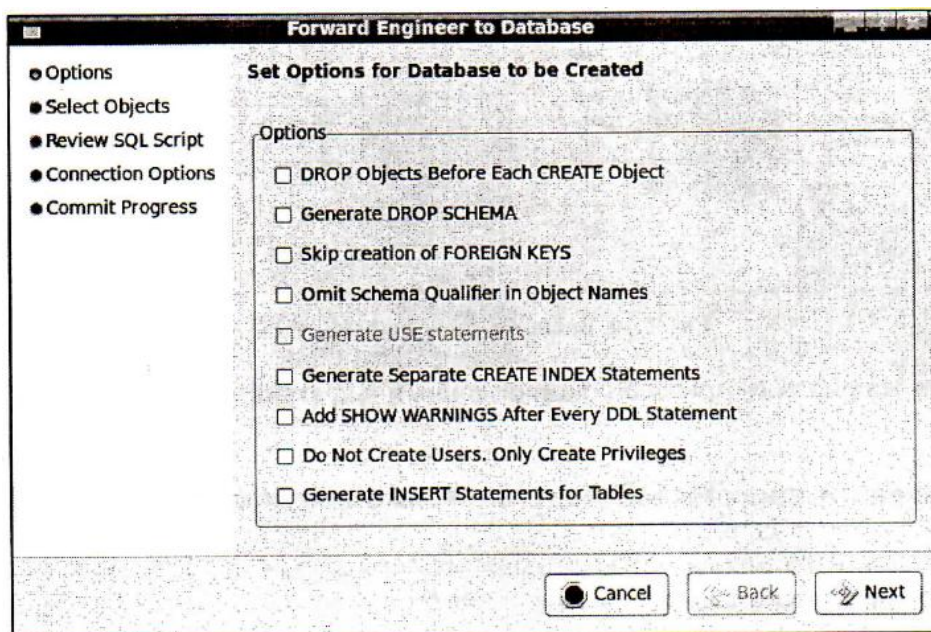
Следующий этап проектирования базы данных - переход к физической модели. Для того чтобы построить функционирующую модель, необходимо воспользоваться функцией прямого инжиниринга.

Прямой инжиниринг предназначен для экспорта ER-модели на MySQL сервер.

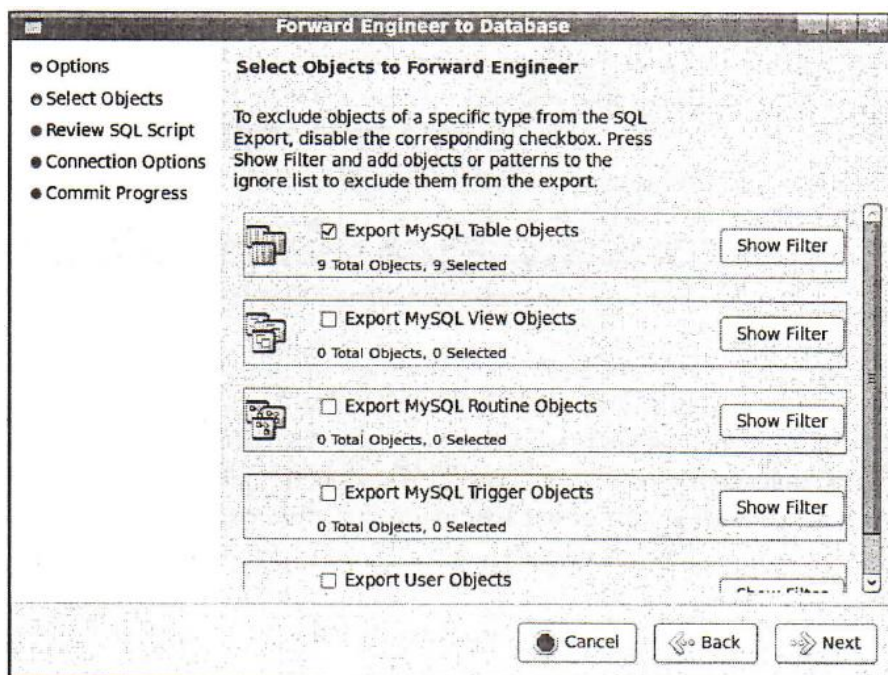
Выберем, как показано на рисунке, меню **Database - Forward Engineering**:



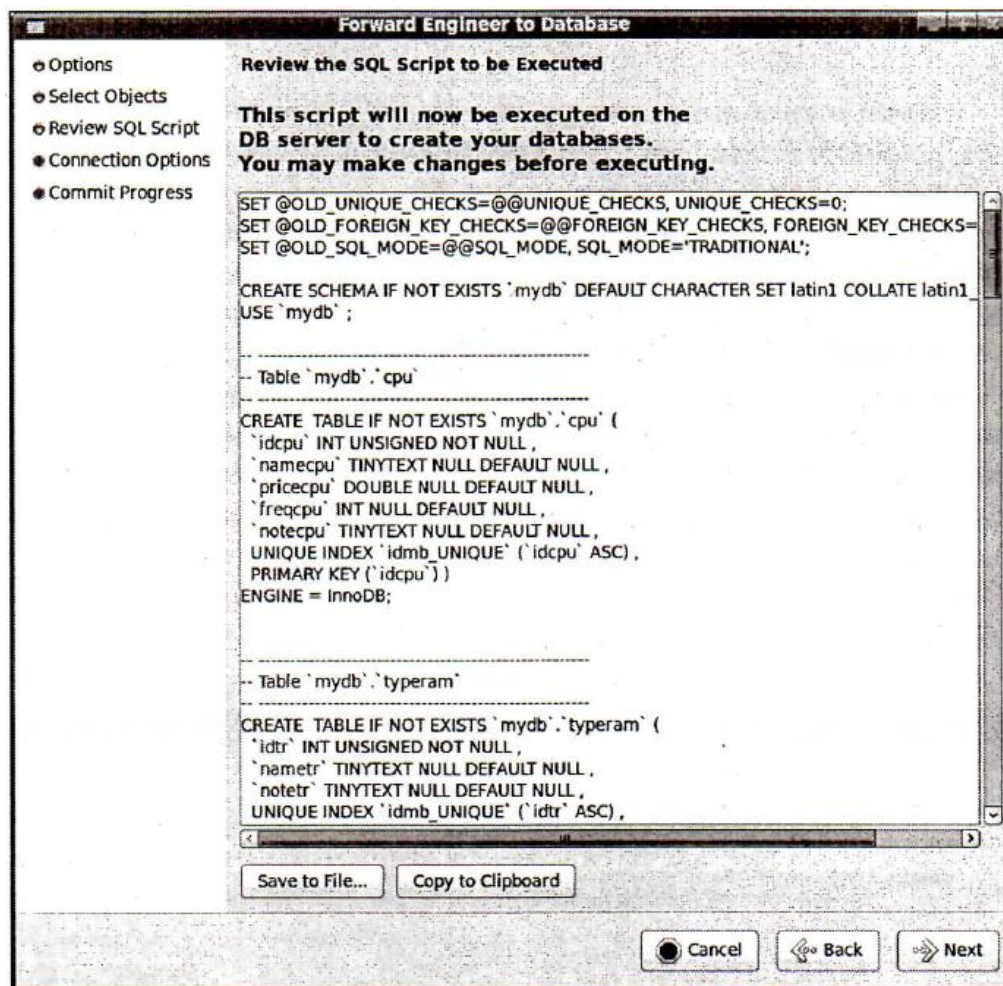
Далее будет открыто меню опций, в котором можно выбрать требуемые опции. При наведении на опции курсора возникает всплывающая подсказка, которая поясняет смысл выбираемой опции. Если нет необходимости, можно не выбирать опции, а сразу нажать на кнопку **Next**.



Следующим шагом является выбор объектов. Здесь можно выбрать опции, связанные с объектами, и посмотреть настройки фильтров. В данном случае выбирается только создание объектов-таблиц (9 штук), поскольку никаких других объектов в рамках ER-модели создано не было. В том случае, если были бы созданы иные объекты, например триггеры, следовало бы поставить галочку в соответствующем поле для их генерации при создании базы данных на сервере

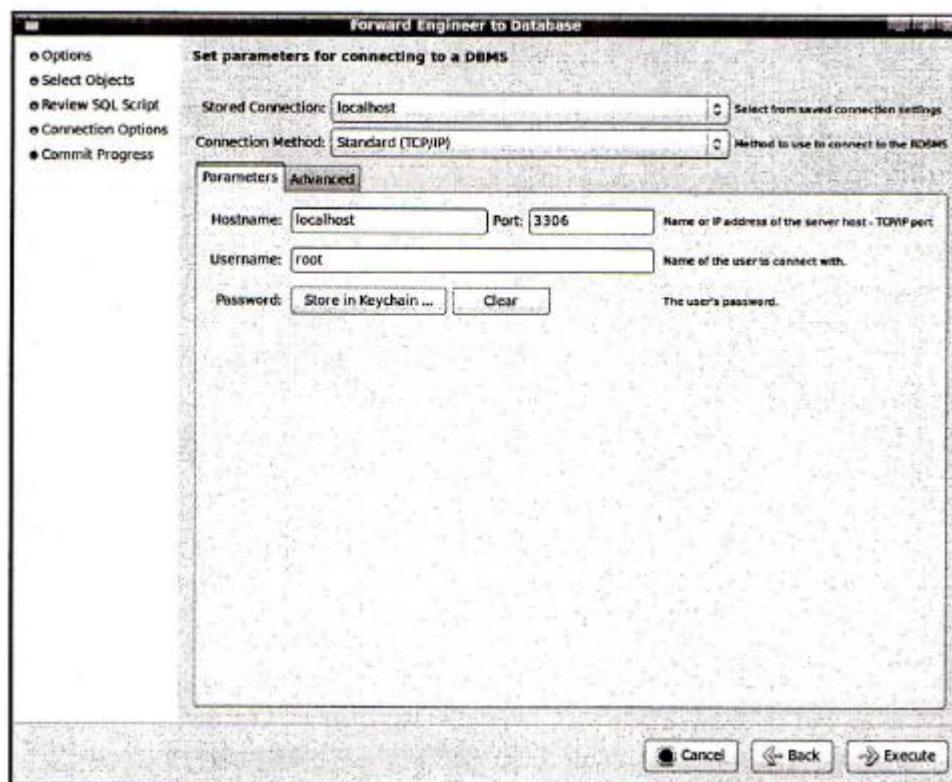


Далее можно осуществить предварительный просмотр SQL скрипта:

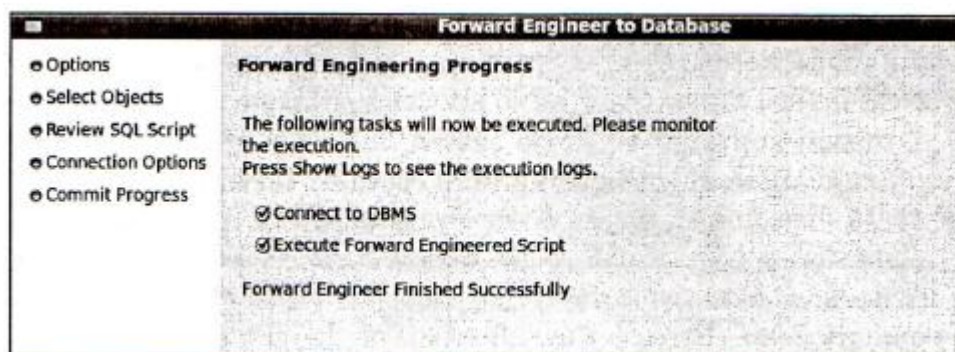


Если нет необходимости вносить изменения в сгенерированный код, тогда можно переходить на следующий этап, на котором по данному коду будет создана база данных. Не забудьте по мере построения модели периодически сохранять результаты работы. Первоначально, чтобы сохранить модель, необходимо выбрать пункт меню **Save as**, затем следует выбирать просто **Save**.

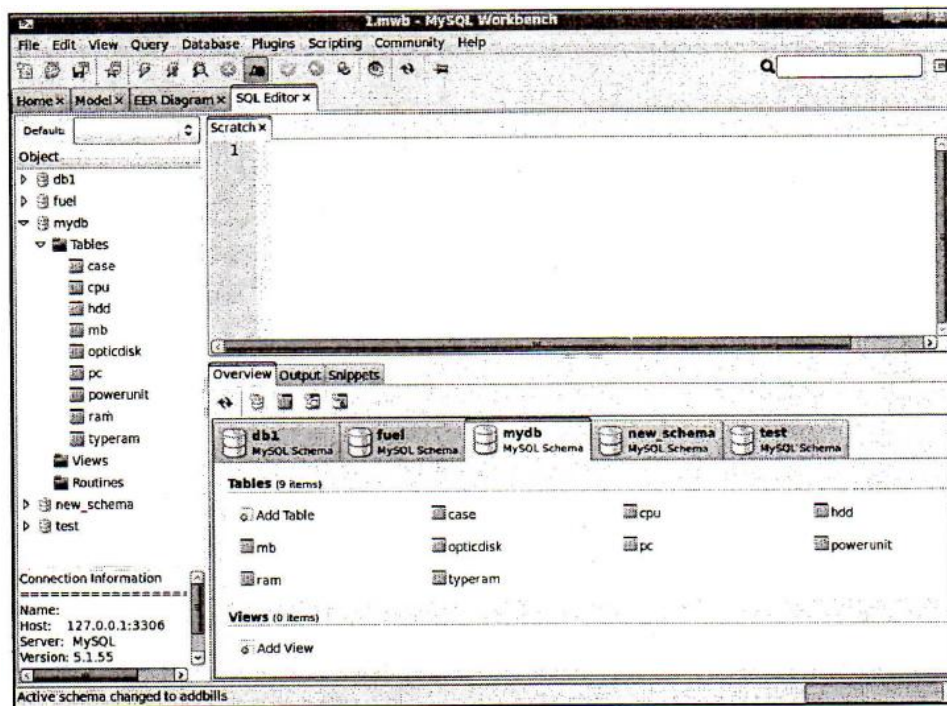
Следующий шаг заключается в настройке параметров соединения с сервером базы данных:



В данном случае следует выбрать соединение с локальным сервером — localhost. Также необходимо проверить, что настроен протокол TCP/IP. Остальные настройки, такие как имя пользователя и пароль, проводятся по необходимости. Если все настройки завершены, для окончания генерации базы данных необходимо нажать кнопку **Execute**. После ее нажатия появится окно процесса генерации:



Нажимаем кнопку **Close** внизу страницы. По завершении генерации новая база данных будет создана из ER-модели. Ее можно будет увидеть во вкладке объектов слева и во вкладке **Overview** внизу.



Лабораторная работа № 3

Тема: создание и использование программируемых объектов баз данных.

Цель: знакомство с возможностями по созданию средствами СУБД хранимых процедур, функций, курсоров.

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.

1. Повторное использование кода - после создания хранимой процедуры ее можно вызывать из любых приложений и SQL-запросов.
2. Сокращение сетевого трафика - вместо нескольких запросов экономнее послать серверу запрос на выполнение хранимой процедуры и сразу получить ответ.
3. Безопасность - действия не приведут к нарушению целостности данных, т.к. для выполнения хранимой процедуры пользователь должен иметь привилегию.
4. Простота доступа - хранимые процедуры позволяют инкапсулировать сложный код и оформить его в виде простого вызова.
5. 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

```
CREATE PROCEDURE 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 *END label*. 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 *//*:

```
1 DELIMITER $  
2 use book $  
3 CREATE PROCEDURE My_vers()  
4 BEGIN  
5 SELECT Version();  
6 END $
```

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

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* новое значение :

```
1 delimiter $  
2 create PROCEDURE set_x(in value int)  
3 BEGIN  
4 set @x=value;  
5 end $
```

Через параметр *value* процедуре передается числовое значение (например, 1234).

которое она присваивает пользовательской переменной `@x`. Модификатор `IN` сообщает, что данные передаются внутрь функции. Проверим корректность работы процедуры :

```
1 call set_x(1234);
2 select @x
```

+ Параметры

@x	
1234	

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 `numAvtor()`, которая подсчитывает число записей в таблице `Avtor` базы данных `book` :

```
1 DELIMITER $
2 CREATE PROCEDURE numAvtor(OUT total Int)
3 BEGIN
4 SELECT COUNT(*) INTO total FROM avtor;
5 END$
```

✓ MySQL вернула пустой результат (т.е. ноль строк). (Запрос занял 0,0000 сек.)

```
call numAvtor(@a)
```

[Построчное редактирование] [Изменить]

✓ Отображение строк 0 - 0 (1 всего, Запрос занял 0,0000 сек.)

```
select @a
```

☐ Профилирование [Построчное редактирование] [Изменить] [Анализ SQL запроса]

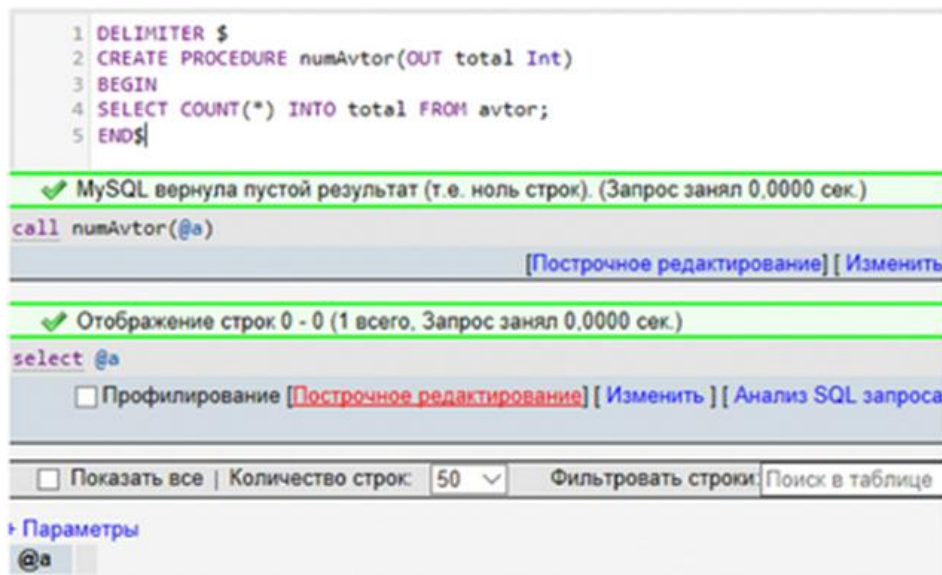
☐ Показать все | Количество строк: 50 | Фильтровать строки: Поиск в таблице

+ Параметры

@a	
----	--

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

Создадим хранимую процедуру `PRubrname()`, которая будет возвращать по шифру подрубрики `ID_PodRubr` ее название `PodRubrName`. Для этого потребуется определить параметр `id` с атрибутом `IN`, и `PRName` с атрибутом `OUT`:



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:

```

IF лог_выражение THEN оператор
    [ELSEIF лог_выражение THEN оператор] ...
    [ ELSE оператор]
END IF ;

```

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]
END CASE;

```


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
END WHILE [ 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 END LOOP [ 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 тип_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 составною 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.

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 CURSOR FOR* 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.

Порядок выполнения работы №3:

1. Создать хранимую процедуру, которая выполняет арифметическую операцию над полями таблицы по вводимому параметру процедуры.
2. Создать хранимую процедуру, которая возвращает связанные записи нескольких таблиц.
3. Создать хранимую процедуру, вычисляющую агрегированные характеристики записей таблицы (например, минимальное, максимальное и среднее значение некоторых полей).
4. Создать функцию, использующую конструкцию CASE (например, преобразование номера дня недели в текст), вывести результат выполнения функции в запросе.
5. Создайте курсор для вывода записей из таблицы, удовлетворяющих заданному условию.
6. Создать хранимую процедуру, которая записывает в новую таблицу все картежи из существующей таблицы по определенному критерию отбора. Предварительно необходимо создать новую пустую таблицу со структурой, аналогичной структуре существующей таблицы. Хранимая процедура должна использовать курсор, который в цикле читает данные из существующей таблицы и добавляет их в новую таблицу.

Лабораторная работа № 4

Тема: создание и использование триггеров; использование JSON в MySQL.

Цель: знакомство с возможностями по созданию средствами СУБД триггеров; изучение возможностей по использованию JSON в MySQL.

Теоретические сведения

ТРИГГЕРЫ

Рассмотрим следующие вопросы:

- понятие триггера;
- создание триггеров с помощью оператора *CREATE TRIGGER*;
- удаление триггеров с помощью оператора *DROP TRIGGER*.

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*.

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

JSON

Для работы с JSON-полями MySQL поддерживает тип данных JSON.

Добавление данных

Добавление записи в БД, содержащей json поле, может быть выполнено несколькими способами:

1. Для добавления записи, содержащей поле с json данными, достаточно добавить правильно сформированную json строку в значение этого поля в *INSERT* запросе.
2. Используя функцию *JSON_OBJECT* можно сформировать json строку в процессе добавления данных. Эта функция принимает список пар ключ-значение вида

JSON_OBJECT(key1, value1, key2, value2, ... key(n), value(n))

и возвращает JSON объект. При этом если значение является массивом, используется функция

JSON_ARRAY(value1, value2, ..., value(n)),

которая возвращает массив json объектов.

Чтение данных:

Для выбора нужных данных по json полю применяется функция JSON_EXTRACT(column, path), которая принимает в качестве аргумента поле таблицы с json данными и путь для перемещения по JSON объекту. Такие же действия выполняет конструкция вида “column->path”.

Например, запросы

1. SELECT * FROM product WHERE attr->'\$.screen' > 30

2. SELECT * FROM product WHERE JSON_EXTRACT(attr, '\$.screen') > 30

являются эквивалентными.

Обновление данных:

Для обновления JSON значений используются следующие функции: JSON_INSERT(column, path, value), добавляет новый ключ и соответствующее ему значение, JSON_REPLACE(column, path, value) заменяет значение уже существующего ключа, JSON_SET(column, path, value) заменяет существующие значения и добавляет несуществующие значения. Данные функции возвращают json объект с примененными изменениями.

Удаление данных:

Для удаления данных из json используется функция JSON_REMOVE(column, path), которая удаляет данные по указанному пути path возвращает измененный json.

Порядок выполнения работы №4:

Триггеры

1. Создать триггер:
 - a. запрещающий вставку в таблицу новой строки с заданным параметром;
 - b. заполняющий одно из полей таблицы на основе вводимых данных.
2. Создать триггер ведения аудита изменения записей в таблицах.
3. Внести такие изменения в триггеры вставки и изменения записей таблиц, которые не позволят добавить или изменить записи с дублирующими названиями.

JSON

4. Создать новую таблицу или изменить существующую, добавив поле типа JSON, заполнить таблицу данными. Минимум одно из значений записи должно представлять из себя вложенную структуру, одно – массив.
5. Выполнить запрос, возвращающий содержимое данной таблицы, соответствующее некоторому условию, проверяющему значение атрибута вложенной структуры.
6. Выполнить запрос, изменяющий значение по некоторому существующему ключу в заданной строке таблицы.

Лабораторная работа № 5

Тема: разработка системы защиты информации на основе привилегий.

Цель: знакомство с возможностями СУБД по защите информации на основе привилегий.

Теоретические сведения

Учетные записи пользователей

Под учетной записью пользователя MySQL подразумевается строка в таблице user (Пользователь) системной базы данных mysql. Первичным ключом в этой таблице служат столбцы Host и User. Таким образом, в MySQL идентификация пользователя основана не только на имени пользователя, но и на комбинации имени пользователя и хоста, с которого подключается пользователь. Это означает, что вы не просто можете ограничить круг хостов, с которых разрешено подключаться данному пользователю; вы можете, например, создать *разные* учетные записи (а следовательно, назначить разные привилегии доступа) для пользователя anna, подключающегося с компьютера localhost, и для пользователя anna, подключающегося с компьютера somedomain.com.

Столбец User допускает значения длиной не более 16 символов. Значение этого столбца может быть пустым, что соответствует анонимному пользователю, но в этой книге мы не будем рассматривать такую возможность.

Столбец Host допускает следующие значения:

- конкретное имя компьютера или IP-адрес;
- маска подсети (например, 192.168.1.0/255.255.255.0);
- маска имени компьютера или маска IP-адреса, которая может содержать подстановочные символы:

В командах, управляющих учетными записями пользователей MySQL, используется *идентификатор пользователя* – значение первичного ключа учетной записи в формате

'<Значение столбца User>'['@'<Значение столбца Host>']

Например, 'anna'@'localhost'. Если значение столбца Host не указано, подразумевается маска %, так что идентификаторы 'anna' и 'anna'@ % эквивалентны.

При подключении пользователя к серверу MySQL происходит идентификация пользователя – поиск соответствующей ему учетной записи. Поиск начинается с тех строк таблицы user, в которых значение столбца Host не содержит подстановочных символов. Поэтому, например, если в таблице зарегистрированы две учетные записи с идентификаторами, соответственно, 'anna'@ % и 'anna'@'localhost', то при подключении пользователя с именем anna с локального компьютера будет выбрана

вторая из них.

После определения учетной записи выполняется аутентификация (проверка подлинности) пользователя, которая заключается в сравнении введенного пользователем пароля с паролем учетной записи, который хранится в столбце Password таблицы user (обратите внимание, что пароли хранятся и передаются только в зашифрованном виде). Если пароль указан правильно, то первый этап контроля доступа завершается успешно и устанавливается соединение клиентского приложения с сервером.

Чтобы *создать учетную запись пользователя*, выполните команду

```
CREATE USER <Идентификатор пользователя>  
[IDENTIFIED BY [PASSWORD] '<Пароль>'];
```

Обязательным параметром этой команды является идентификатор нового пользователя. Если не задан параметр IDENTIFIED BY, то будет использоваться пустой пароль.

Параметр PASSWORD необходимо указать в том случае, если вы вводите не реальный, а зашифрованный пароль (что позволяет избежать передачи незашифрованного пароля при отправке на сервер команды CREATE USER). Получить зашифрованное значение из реального пароля вы можете с помощью функции

```
PASSWORD('<Реальный пароль>')
```

Например, команда

```
CREATE USER 'anna' IDENTIFIED BY 'annapassword';
```

создает учетную запись для пользователя с именем anna, подключающегося с любого компьютера, и устанавливает для этой учетной записи пароль annapassword. Команда

```
CREATE USER 'anna'@'localhost' IDENTIFIED BY PASSWORD  
'*3C7F72EAE78BC95AAFBFD21F8741C24A0056C04B';
```

создает учетную запись для пользователя anna, подключающегося с локального компьютера, и устанавливает в качестве пароля значение annalocpassword (поскольку функция PASSWORD('annalocpassword') возвращает значение *3C7F72EAE78BC95AAFBFD21F8741C24A0056C04B).

Сразу после выполнения команды CREATE USER новый пользователь может подключаться к серверу MySQL.

В следующем подразделе мы обсудим, как изменить пароль пользователя, а также как восстановить забытый пароль пользователя root.

Для *установки пароля* предназначена команда

```
SET PASSWORD [FOR <Идентификатор пользователя>]  
= PASSWORD('<Пароль>');
```

Параметрами этой команды являются идентификатор учетной записи пользователя и новый пароль для этой записи. Если вы не укажете идентификатор пользователя, то измените свой пароль.

Вместо функции PASSWORD(), зашифровывающей реальный пароль, можно сразу ввести зашифрованный пароль. Например, команды

```
SET PASSWORD FOR 'anna'@'%' =  
PASSWORD('newannapassword');
```

и

```
SET PASSWORD FOR 'anna'@'%' =  
'*006B99DE1BDA1BE6E1FFF714E764A8FAB0E614DF';
```

устанавливают пароль newannapassword для пользователя anna, подключающегося с любого компьютера. Эти команды никак не влияют на другие учетные записи, например пароль учетной записи с идентификатором 'anna'@'localhost' не изменится.

Удалить учетную запись вы можете с помощью команды

```
DROP USER <Идентификатор пользователя>;
```

После удаления пользователь лишается возможности подключаться к серверу MySQL. Однако если на момент удаления пользователь был подключен к серверу, то соединение не прерывается.

Вместе с учетной записью удаляются все привилегии доступа для этой записи.

Для *получения информации о зарегистрированных пользователях* выполним запрос к таблице user (Пользователь) системной базы данных mysql, например

```
SELECT * FROM mysql.user;
```

Первые три столбца таблицы user нам уже знакомы – это Host, User и Password.

Далее следуют *столбцы глобальных привилегий*, которые мы рассмотрим в разделе «Система привилегий доступа», и, наконец, столбцы, в которых содержатся параметры безопасности соединения и сведения о ресурсах, предоставляемых соединению (эти столбцы остаются за рамками нашего обсуждения).

Система привилегий доступа

Данный раздел посвящается второму этапу контроля доступа пользователей – проверке привилегий доступа при выполнении каждой операции в базе данных.

Вы узнаете, какие привилегии предусмотрены в MySQL и как предоставить их пользователям.

Создание привилегии доступа в MySQL подразумевает определение следующих параметров:

- идентификатор учетной записи пользователя, которому предоставляется привилегия;
- тип привилегии, то есть тип операций, которые будут разрешены пользователю;
- область действия привилегии.

В MySQL используются следующие основные типы привилегий:

- ALL [PRIVILEGES] – предоставляет все привилегии, кроме GRANT OPTION, для указанной области действия;
- ALTER – разрешает выполнение команд ALTER DATABASE и ALTER TABLE;
- CREATE – разрешает выполнение команд CREATE DATABASE и CREATE TABLE;
- CREATE USER – разрешает выполнение команд CREATE USER, DROP USER, RENAME USER;
- DELETE – разрешает выполнение команды DELETE;
- DROP – разрешает выполнение команд DROP DATABASE и DROP TABLE;
- FILE – разрешает чтение и создание файлов на сервере с помощью команд SELECT... INTO OUTFILE и LOAD DATA INFILE;
- INDEX – разрешает выполнение команд CREATE INDEX и DROP INDEX;
- INSERT – разрешает выполнение команды INSERT;
- SELECT – разрешает выполнение команды SELECT;
- LOCK TABLES – разрешает выполнение команды LOCK TABLES при наличии привилегии SELECT для блокируемых таблиц;
- SHOW DATABASES – разрешает отображение всех баз данных при выполнении команды SHOW DATABASES (если эта привилегия отсутствует, то в списке будут отображены только те базы данных, по отношению к которым у пользователя есть какая-либо привилегия);

- RELOAD – разрешает выполнение команды FLUSH;
- SUPER – привилегия администратора сервера; в частности, разрешает выполнение команды SET GLOBAL;
- UPDATE – разрешает выполнение команды UPDATE;
- GRANT OPTION – разрешает назначать и отменять привилегии другим пользователям (эта возможность распространяется только на те привилегии, которые есть у самого пользователя для указанной области действия).

Областью действия привилегии могут быть:

- все базы данных (такие привилегии называются глобальными);
- отдельная база данных;
- таблица;
- столбец таблицы.

Каждый тип привилегии имеет свои допустимые области действия. Так, привилегии FILE, SHOW DATABASES, RELOAD, SUPER и CREATE USER могут быть только глобальными. Привилегия LOCK TABLES может применяться глобально или к отдельным базам данных, но не к отдельным таблицам. К отдельным столбцам таблицы применимы только привилегии SELECT, INSERT и UPDATE.

Чтобы получить разрешение на выполнение операции с каким-либо объектом базы данных, пользователю достаточно иметь привилегию соответствующего типа для какой-либо области действия, содержащей этот объект. Например, пользователь сможет выполнить запрос данных из столбца description (наименование) таблицы Products (Товары) базы данных SalesDept (Отдел продаж), если у него есть хотя бы одна из следующих привилегий:

- глобальная привилегия SELECT;
- привилегия SELECT для базы данных SalesDept;
- привилегия SELECT для таблицы Products;
- привилегия SELECT для столбца description.

Для выполнения некоторых операций может потребоваться несколько типов привилегий. Например, команда

UPDATE SalesDept.Products SET price='548.00' WHERE id=5;

доступна пользователю, если у него одновременно есть привилегия SELECT для таблицы Products (или для базы данных SalesDept, или глобальная) и привилегия UPDATE для столбца price (или для таблицы Products, или для базы данных SalesDept, или глобальная).

Теперь, получив общее представление о привилегиях доступа в MySQL, вы можете переходить к назначению привилегий пользователям.

Предоставление привилегий

Для предоставления привилегий пользователям используется команда

GRANT <Тип привилегии>

[(<Список столбцов>)] ON <Область действия>

TO <Идентификатор пользователя>

[WITH GRANT OPTION];

В качестве области действия вы можете указать одно из следующих значений:

- *.* – привилегия будет действовать глобально;
- <Имя базы данных>.* – привилегия будет действовать для указанной базы данных;
- * – привилегия будет действовать для базы данных, которая в момент выполнения команды GRANT являлась текущей;
- <Имя базы данных>.<Имя таблицы> или <Имя таблицы> – привилегия будет действовать для указанной таблицы (если имя базы данных не указано, подразумевается текущая база данных). Если требуется создать привилегию не для всей таблицы, а только для отдельных столбцов, необходимо перечислить эти столбцы в скобках перед ключевым словом ON.

Рассмотрим несколько примеров.

1) GRANT CREATE ON *.* TO 'anna'@'localhost'; Команда предоставляет пользователю anna'@'localhost привилегию на создание баз данных и таблиц в любой базе данных.

2) GRANT DROP ON SalesDept.* TO 'anna'@'localhost'; Команда предоставляет пользователю anna'@'localhost привилегию на удаление таблиц в базе данных SalesDept (Отдел продаж), а также на удаление самой базы данных SalesDept.

3) GRANT SELECT ON SalesDept.Products TO 'anna'@'localhost'; Команда предоставляет пользователю anna'@'localhost привилегию на получение данных из таблицы Products (Товары) базы данных SalesDept (Отдел продаж).

4) GRANT UPDATE (price) ON SalesDept.Products TO 'anna'@'localhost'; Команда предоставляет пользователю anna'@'localhost привилегию на изменение данных в столбце price (цена) таблицы Products (Товары).

При назначении привилегий необходимо иметь в виду следующие особенности.

- Если учетная запись с указанным идентификатором не существует, команда GRANT может создать такую запись. Отключить автоматическое создание учетной записи позволяет ключевое слово NO_AUTO_CREATE_USER в значении переменной sql_mode.

- Привилегию ALTER рекомендуется предоставлять с осторожностью: путем переименования таблиц и столбцов пользователь может изменить

настройки системы привилегий.

– Если при создании привилегии вы указываете параметр WITH GRANT OPTION, то пользователь получает возможность «делиться» не только созданной привилегией, но и *другими* своими привилегиями в рамках данной области действия. Например, после выполнения команд

```
GRANT SELECT ON *.* TO 'marina';  
GRANT INSERT ON SalesDept.* TO 'marina' WITH GRANT OPTION;  
GRANT DELETE ON SalesDept.Customers TO 'marina';
```

пользователь marina может предоставлять другим пользователям следующие привилегии:

– привилегии SELECT, INSERT и GRANT OPTION на уровне базы данных SalesDept (Отдел продаж). Хотя сам пользователь marina имеет глобальную привилегию SELECT, его возможности делегирования привилегий ограничены базой данных SalesDept;

– привилегию DELETE на уровне таблицы Customers (Клиенты) базы данных SalesDept, так как область действия этой привилегии входит в область действия привилегии GRANT OPTION.

Если вы предоставляете привилегию GRANT OPTION нескольким пользователям, эти пользователи могут «обмениваться» привилегиями, то есть объединить свои наборы привилегий.

Привилегии, областью действия которых является таблица или столбец, вступают в силу немедленно – пользователь может сразу же начать выполнять SQL-команды, разрешенные ему новой привилегией. Привилегии, относящиеся к базе данных, начинают действовать после выполнения команды USE <имя базы данных>, то есть после выбора какой-либо базы данных в качестве текущей. Глобальные привилегии начинают применяться при следующем подключении пользователя к серверу MySQL.

Чтобы *удалить привилегию*, ранее назначенную пользователю, используется команда

```
REVOKE <Тип привилегии>  
[(<Список столбцов>)] ON <Область действия>  
FROM <Идентификатор пользователя>;
```

Например:

- REVOKE CREATE ON *.* FROM 'anna'@'localhost';

Команда отменяет глобальную привилегию пользователя 'anna'@'localhost', разрешавшую создание баз данных и таблиц.

- REVOKE DROP ON SalesDept.* FROM 'anna'@'localhost'; Команда отменяет

привилегию пользователя 'anna'@'localhost' на удаление базы данных SalesDept (Отдел продаж) и таблиц в этой базе данных.

- REVOKE SELECT ON SalesDept.Products

FROM 'anna'@'localhost';

Команда отменяет привилегию пользователя 'anna'@'localhost' на получение данных из таблицы Products (Товары) базы данных SalesDept.

- REVOKE UPDATE (price) ON SalesDept.Products

FROM 'anna'@'localhost';

Команда отменяет привилегию пользователя 'anna'@'localhost' на изменение данных в столбце price (цена) таблицы Products (Товары).

Параметры команды REVOKE имеют тот же смысл, что и параметры команды GRANT. Аналогичны и правила вступления изменений в силу.

Отметим, что в MySQL при удалении баз данных, таблиц и столбцов связанные с ними привилегии не удаляются автоматически; для удаления таких привилегий требуется выполнить команду REVOKE.

Если же требуется узнать, к каким объектам имеет доступ конкретный пользователь, выполните команду

SHOW GRANTS [FOR идентификатор пользователя];

Команда SHOW GRANTS выводит сведения о привилегиях пользователя в виде набора команд GRANT, с помощью которых можно сформировать текущий набор привилегий пользователя. Если идентификатор пользователя не задан, вы увидите свои привилегии.

Индексы

Индекс – объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск — например, сбалансированное дерево.

Хотя индексы могут быть созданы для каждого столбца, используемого в запросах, ненужные индексы занимают дисковое пространство и увеличивают время выполнения запроса из-за необходимости оценки, какой индекс следует использовать. Индексы также увеличивают время выполнения запросов на

добавление, изменение и удаление данных из-за необходимости обновления индекса после выполнения соответствующих запросов.

Для индексации доступны как обычные типы данных, так и пространственные. Для столбцов типов CHAR и VARCHAR можно индексировать префикс столбца.

Для создания индекса используется команда CREATE INDEX, синтаксис которой представлен ниже:

```
CREATE INDEX index_name ON table_name(column_name);
```

index_name – название индекса; table_name – название таблицы, в котором содержится индексируемое поле; column_name – поле таблицы, для которого создается индекс.

MySQL поддерживает *уникальные* индексы. Они применяются для тех полей таблицы, значения в которых должны быть уникальными по всей таблице. Такие индексы улучшают эффективность выборки для уникальных значений. Для создания уникального индекса используется ключевое слово UNIQUE:

```
CREATE UNIQUE INDEX index_name ON table_name(column_name);
```

Индексы могут быть *составными*. Так как MySQL часто может использовать только один индекс для выполнения запроса, появляется необходимость в создании составных индексов. Рассмотрим запрос:

```
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;
```

Если по столбцам col1 и col2 существует составной индекс, то соответствующие строки могут выбираться напрямую. В случае, когда по столбцам col1 и col2 существуют отдельные индексы, оптимизатор пытается найти наиболее ограничивающий индекс путем определения, какой индекс найдет меньше строк, и использует данный индекс для выборки этих строк.

Если данная таблица имеет составной индекс, то любой крайний слева префикс этого индекса может использоваться оптимизатором для нахождения строк. Например, если имеется индекс по трем столбцам (col1,col2,col3), то существует потенциальная возможность индексированного поиска по (col1), (col1,col2) и (col1,col2,col3).

В MySQL нельзя использовать составной индекс, если столбцы не образуют крайний слева префикс этого индекса. Например, если индекс существует по (col1,col2,col3), то запрос:

```
SELECT * FROM tbl_name WHERE col2=val2;
```

не будет использовать индекс.

Последние версии MySQL поддерживают *функциональные* индексы, т.е. при создании индекса мы можем указать не просто столбец, а использовать некоторую функцию, принимающую данные столбца.

MySQL применяет индексы также для сравнений LIKE, если аргумент в выражении LIKE представляет собой постоянную строку, не начинающуюся с символа-шаблона. Например, следующие команды SELECT используют индексы:

```
SELECT * FROM tbl_name WHERE key_col LIKE "Patrick%";
```

```
SELECT * FROM tbl_name WHERE key_col LIKE "Pat%_ck%";
```

Следующая команда SELECT не будет использовать индексы:

```
SELECT * FROM tbl_name WHERE key_col LIKE "%Patrick%";
```

MySQL поддерживает *полнотекстовые* индексы. Эти индексы могут быть созданы в столбцах VARCHAR и TEXT во время создания таблицы командой CREATE TABLE или добавлены позже с помощью команд ALTER TABLE или CREATE FULLTEXT INDEX. Загрузка больших массивов данных в таблицу будет происходить намного быстрее, если таблица не содержит индекс FULLTEXT, который затем создается командой ALTER TABLE (или CREATE FULLTEXT INDEX). Загрузка данных в таблицу, уже имеющую индекс FULLTEXT, будет более медленной.

Полнотекстовый поиск выполняется с помощью конструкции MATCH(filelds)... AGAINST(words). Рассмотрим пример:

```
SELECT * FROM articles WHERE MATCH (title, body) AGAINST ('database');
```

В этом примере осуществляется поиск слова «database» в полях title и body таблицы articles. Полученная выборка будет автоматически отсортирована по релевантности (мера сходства между строкой поиска и текстом) – это происходит в случае указания конструкции MATCH-AGAINST внутри блока WHERE и не задано условие сортировки ORDER BY. Эта величина зависит от количества слов в полях title и body, того насколько близко данное слово встречается к началу текста, отношения количества встретившихся слов к количеству всех слов в поле и др. Например, релевантность будет не нулевая, если слово database встретится либо в title, либо body, но если оно встретится и там и там, значение релевантности будет выше, нежели если оно два раза встретится в body.

Полнотекстовый поиск позволяет выполнять поиск вхождения подстроки в строку, используя полнотекстовый индекс, в отличие от поиска с помощью оператора LIKE.

Для анализа эффективности выполнения запросов используется оператор EXPLAIN. Данный оператор используется перед оператором SELECT и возвращает таблицу. Колонка key показывает используемый индекс. Колонка possible_keys показывает все индексы, которые могут быть использованы для этого запроса. Колонка rows показывает число записей, которые пришлось прочитать базе данных для выполнения этого запроса.

Для выполнения лабораторной работы:

Для демонстрации эффективности использования индексов при выполнении лабораторной работы используется база данных с большим числом записей в таблицах - часть БД сервиса **discogs.com**. Discogs - веб-сайт с одной из крупнейших баз данных музыкальных исполнителей и их изданий от различных музыкальных компаний. В состав данной БД входят следующие таблицы:

- artist – содержит информацию об исполнителях (включая музыкальные коллективы).

- group – содержит информацию о принадлежности исполнителя к музыкальному коллективу
- namevaration – содержит информацию о различных вариациях имен (псевдонимов) исполнителей.
- release - содержит информацию о музыкальных релизах.
- releaseartist – описывает связь между исполнителем и релизом.
- style - содержит информацию о музыкальном стиле релизов.

Импорт БД доступен через меню «Server – Data import».

Порядок выполнения работы №5:

1. Часть 1. Привилегии

- 1) Создайте нового пользователя и предоставьте ему привилегии на выборку информации из созданных вами представлений и запуска процедур.
- 2) Соединитесь с СУБД от имени нового пользователя и проверьте возможность доступа к вашим данным через представления и запуск процедур.
- 3) Проверьте невозможность доступа к вашим данным через таблицы и хранимые процедуры.
- 4) Соединитесь с СУБД от своего имени и предоставьте новому пользователю привилегии на выборку, вставку, изменение и удаление данных из ваших таблиц.
- 5) Соединитесь с СУБД от имени нового пользователя и проверьте возможность изменения данных в созданных вами таблицах.
- 6) Соединитесь с СУБД от своего имени и отберите у нового пользователя все предоставленные ему привилегии.
- 7) Соединитесь с СУБД от имени нового пользователя и проверьте невозможность изменения данных в созданных Вами таблицах и запуск ваших хранимых процедур.

2. Часть 2. Индексы

- 1) Составляются и выполняются запросы:
 - a) Найти информацию по заданному исполнителю, используя его имя.
 - b) Найти всех участников указанного музыкального коллектива (по названию коллектива).
 - c) Найти всех исполнителей, в описании (профиле) которых встречается указанное выражение, с использованием полнотекстового запроса.
- 2) Оценивается время выполнения запросов.
- 3) Анализируется план выполнения запросов.
- 4) Создаются необходимые индексы для повышения быстродействия запросов. Выполнение запроса должно исключать полное сканирование таблицы (отсутствие Table Scan в анализе запроса).
- 5) Оценивается время выполнения тех же запросов при наличии созданных индексов.

Запросы:

1. Найти информацию по заданному исполнителю, используя его имя.
2. Найти всех участников указанного музыкального коллектива (по названию коллектива).

3. Найти все релизы заданного исполнителя и отсортировать их по дате выпуска. Вывести имя исполнителя, название релиза, дату выхода.
4. Найти все главные релизы, выпущенные в указанный год, с указанием стиля релиза. Релиз является главным, если поле `release.IS_MAIN_RELEASE = 1`.
5. Найти всех исполнителей, в описании (профиле) которых встречается указанное выражение, с использованием полнотекстового запроса.

Описание вариантов заданий предметной области

Задача 1

Летопись острова Санта-Белинда

Где-то в великом океане находится воображаемый остров Санта-Белинда. Вот уже триста лет ведется подробная летопись острова. В эту летопись заносятся и данные обо всех людях, какое-то время проживавших на острове. Записываются их имена, пол, даты рождения и смерти. Хранятся там и имена их родителей, если известно, кто они. У некоторых отсутствуют сведения об отце, у некоторых — о матери, а часть людей, судя по записям, — круглые сироты. Из летописи можно узнать, когда был построен каждый дом, стоящий на острове (а если сейчас его уже нет, то когда он был снесен), точный адрес и подробный план этого дома, кто и когда в нем жил.

Точно так же, как и столетия назад, на острове действуют предприниматели, занимающиеся, в частности, ловлей рыбы, заготовкой сахарного тростника и выращиванием табака. Большинство из них занимается своим промыслом самостоятельно, а некоторые нанимают работников, заключая с ними контракты разной продолжительности. Имеются записи и о том, кто кого нанимал, на какую работу, когда начался и закончился контракт. Собственно, круг занятий жителей острова крайне невелик и не меняется веками. Неудивительно поэтому, что в летописи подробно описывается каждое дело, будь то рыбная ловля или выпечка хлеба. Все предприниматели — уроженцы острова. Некоторые объединяются в кооперативы, и по записям можно установить, кто участвовал в деле, когда вступил и когда вышел из него, каким паем владел. Имеются краткие описания деятельности каждого предпринимателя или кооператива, сообщающие в том числе, когда было начато дело, когда и почему прекращено.

Предлагается сформировать систему нормализованных таблиц, в которых можно было бы хранить всю эту многообразную информацию. Подыщите выразительные имена для таблиц и полей, снабдив их при необходимости соответствующими пояснениями.

Написать запросы, осуществляющие следующие операции:

- 1) Вывести список всех снесенных домов на острове с указанием даты постройки и даты сноса и его адреса. Список упорядочит хронологически.
- 2) Найти дома, в которых за указанный период проживало более 2-х человек.
- 3) Найти 3-х сирот-предпринимателей, имеющих в подчинении наибольшее количество человек.
- 4) Выдать список всех прекративших деятельность кооперативов с указанием причины.
- 5) Найти предпринимателей, дела которых просуществовали более 10 лет.
- 6) Сформировать списки кооперативов по количеству человек в них, начиная с самого большого.

Задача 2

База данных «Скачки»

В информационной системе клуба любителей скачек должна быть представлена информация об участвующих в скачках лошадях (кличка, пол, возраст), их владельцах (имя, адрес, телефон) и жокеях (имя, адрес, возраст, рейтинг). Необходимо сформировать таблицы для хранения информации по каждому состязанию: дата, время и место проведения скачек (ипподром), название состязаний (если таковое имеется), номера заездов, клички участвующих в заездах лошадей и имена жокеев, занятые ими места и показанное в заезде время.

Написать запросы, осуществляющие следующие операции:

- 1) Для заданной лошади найти все соревнования в которых она принимала участие, указать номер заезда и результат ее заезда.
- 2) Выдать список владельцев, имеющих более 2-х лошадей.
- 3) Выдать список соревнований в которых принимали участие не менее 4-х жокеев с заданным уровнем рейтинга.
- 4) За указанный период выдать список состязаний в хронологическом порядке с указанием количества заездов в них.
- 5) Выдать список состязаний и номеров заездов, в которых лошадь «Черная стрела» заняла 1 место.

Задача 3

База данных «Хроника восхождений» в альпинистском клубе

В базе данных должны записываться даты начала и завершения каждого восхождения, имена и адреса участвовавших в нем альпинистов, название и высота горы, страна и район, где эта гора расположена. Присвойте выразительные имена таблицам и полям для хранения указанной информации.

Предоставить возможность добавления новой вершины с указанием ее названия, высоты и страны местоположения. Предоставить возможность добавления нового альпиниста в состав указанной группы. Предоставить возможность добавления новой группы, указав ее название, вершину, время начала восхождения. Предоставить возможность изменения данных о вершине, если на нее не было восхождений.

Написать запросы, осуществляющие следующие операции:

1) Для введенного пользователем интервала дат показать список гор с указанием даты последнего восхождения. Для каждой горы сформировать в хронологическом порядке список групп, осуществлявших восхождение.

2) Показать список альпинистов, осуществлявших восхождение в указанный интервал дат. Для каждого альпиниста вывести список гор, на которые он осуществлял восхождения в этот период, с указанием названия группы и даты восхождения.

3) Показать информацию о количестве восхождений каждого альпиниста на каждую гору. При выводе список отсортировать по количеству восхождений.

4) Показать список восхождений (групп), которые осуществлялись в указанный пользователем период времени. Для каждой группы показать ее состав.

5) Предоставить информацию о том, сколько альпинистов побывало на каждой горе. Список отсортировать в алфавитном порядке по названию вершин.

Задача 4

База данных медицинского кооператива

Базу данных использует для работы коллектив врачей. В таблицы должны быть занесены имя, пол, дата рождения и домашний адрес каждого их пациента. Всякий раз, когда врач осматривает больного (пришедшего на прием или на дому), фиксируется дата и место проведения осмотра, симптомы, диагноз и предписания больному, проставляется имя пациента и имя врача. Если врач прописывает больному какое-либо лекарство, в таблицу заносится название лекарства, способ его приема, словесное описание предполагаемого действия и возможных побочных эффектов.

Написать запросы, осуществляющие следующие операции:

- 1) Выдать список пациентов к которым за указанный период врачи приходили для осмотра на дом
- 2) Выдать список врачей назначивших своим пациентам указанное лекарство.
- 3) По определенному врачу выдать информацию, которая будет содержать сведения о том, каких именно пациентов врач принял за указанный период (с указанием даты приема).
- 4) Выдать список врачей с указанием количества принятых им пациентов за последние 3 месяца.
- 5) Выбрать наиболее распространенные болезни за последний месяц.
- 6) Выбрать все диагнозы по определенному пациенту.

Задача 5

База данных «Городская Дума»

В базе хранятся имена, адреса, домашние и служебные телефоны всех членов Думы. В Думе работает порядка сорока комиссий, все участники которых являются членами Думы. Каждая комиссия имеет свой профиль, например, вопросы образования, проблемы, связанные с жильем, и так далее. Данные по каждой из комиссий включают: председатель и состав, прежние (за 10 предыдущих лет) председатели и члены этой комиссии, даты включения и выхода из состава комиссии, избрания ее председателей. Члены Думы могут заседать в нескольких комиссиях. В базу заносятся время и место проведения каждого заседания комиссии с указанием депутатов и служащих Думы, которые участвуют в его организации.

Предоставить возможность добавления нового члена комиссии. Предоставить возможность добавления новой комиссии с указанием ее председателя. Предоставить возможность добавления нового заседания с указанием присутствующих.

Написать запросы, осуществляющие следующие операции:

- 1) Показать список комиссий, для каждой — ее состав и председателя.
- 2) Для введенного пользователем интервала дат и названия комиссии показать в хронологическом порядке всех ее председателей.
- 3) Показать список членов Думы, для каждого из них — список комиссий, в которых он участвовал и/или был председателем.
- 4) Для указанного интервала дат и комиссии выдать список членов с указанием количества пропущенных заседаний.
- 5) Вывести список заседаний в указанный интервал в хронологическом порядке, для каждого заседания — список присутствующих.
- 6) По каждой комиссии показать количество проведенных заседаний в указанный период времени.

Задача 6

База данных рыболовной фирмы

Фирме принадлежит небольшая флотилия рыболовных катеров. Каждый катер имеет «паспорт», куда занесены его название, тип, водоизмещение и дата постройки. Фирма регистрирует каждый выход на лов, записывая название катера, имена и адреса членов команды с указанием их должностей (капитан, боцман и т.д.), даты выхода и возвращения, а также вес пойманной рыбы отдельно по сортам (например, трески). За время одного рейса катер может посетить несколько рыболовных мест (банок). Фиксируется дата прихода на каждую банку и дата отплытия, качество выловленной рыбы (отличное, хорошее, плохое). На борту улов не взвешивается.

Предоставить возможность добавления выхода катера в море с указанием команды. Предоставить возможность добавления новой банки с указанием данных о ней. Предоставить возможность пользователю изменять характеристики выбранного катера. Предоставить возможность добавления нового катера.

Написать запросы, осуществляющие следующие операции:

1) По указанному типу и интервалу дат вывести все катера, осуществлявшие выход в море, указав для каждого в хронологическом порядке записи о выходе в море и вес улова.

2) Для указанного интервала дат вывести для каждого сорта рыбы список катеров с наибольшим уловом.

3) Для указанного интервала дат вывести список банок с указанием среднего улова за этот период. Для каждой банки вывести список катеров, осуществлявших лов.

4) Для заданной банки вывести список катеров, которые получили улов выше среднего.

5) Вывести список сортов рыбы и для каждого сорта — список рейсов с указанием даты выхода и возвращения, величины улова. При этом список показанных рейсов должен быть ограничен интервалом дат.

6) Для выбранного пользователем рейса и банки добавить данные о сорте и количестве пойманной рыбы.

7) Для указанного интервала дат вывести в хронологическом порядке список рейсов за этот период времени с указанием для каждого рейса веса пойманной рыбы.

8) Для указанного сорта рыбы и банки вывести список рейсов с указанием количества пойманной рыбы. Список должен быть отсортирован в порядке уменьшения количества.

Задача 7

База данных фирмы, проводящей аукционы

Фирма занимается продажей с аукциона антикварных изделий и произведений искусства. Владельцы вещей, выставляемых на проводимых фирмой аукционах, юридически являются продавцами. Лица, приобретающие эти вещи, именуются покупателями. Получив от продавцов партию предметов, фирма решает, на каком из аукционов выгоднее представить конкретный предмет. Перед проведением очередного аукциона каждой из выставляемых на нем вещей присваивается отдельный номер лота. Две вещи, продаваемые на различных аукционах, могут иметь одинаковые номера лотов.

В книгах фирмы делается запись о каждом аукционе. Там отмечаются дата, место и время его проведения, а также специфика (например, выставляются картины, написанные маслом и не ранее 1900 г.). Заносятся также сведения о каждом продаваемом предмете: аукцион, на который он заявлен, номер лота, продавец, отправная цена и краткое словесное описание. Продавцу разрешается выставлять любое количество вещей, а покупатель имеет право приобретать любое их количество. Одно и то же лицо или фирма может выступать и как продавец, и как покупатель. После аукциона служащие фирмы, проводящей аукционы, записывают фактическую цену, уплаченную за проданный предмет, и фиксируют данные покупателя.

Предоставить возможность добавления факта продажи на указанном аукционе заданного предмета. Предоставить возможность добавления записи о проводимом аукционе (место, время). Предоставить возможность добавления и изменения информации о продавцах и покупателях.

Написать запросы, осуществляющие следующие операции:

1) Для указанного интервала дат вывести список аукционов в хронологическом порядке с указанием наименования, даты и места проведения. Для каждого из них показать список выставленных вещей.

2) Вывести список аукционов с указанием отсортированных по величине суммарных доходов от продажи.

3) Для указанного интервала дат вывести список проданных на аукционах предметов. Для каждого из предметов дать список аукционов, где выставлялся этот же предмет.

4) Для указанного интервала дат вывести список продавцов в порядке убывания общей суммы, полученной ими от продажи предметов в этот промежуток времени.

5) Вывести список покупателей и для каждого из них — список аукционов, где были сделаны приобретения в указанный интервал дат.

6) Для указанного места вывести список аукционов, отсортированных по количеству выставленных вещей.

7) Для указанного интервала дат вывести список продавцов, которые принимали участие в аукционах, с указанием для каждого из них списка выставленных предметов.

8) Вывести список покупателей с указанием количества приобретенных предметов в указанный период времени.

Задача 8

База данных музыкального магазина

Таблицы базы данных содержат информацию о музыкантах, музыкальных произведениях и обстоятельствах их исполнения. Несколько музыкантов, образующих единый коллектив, называют ансамблем. Это может быть классический оркестр, джазовая группа, квартет, квинтет и т.д. К музыкантам причисляют исполнителей (играющих на одном или нескольких инструментах), композиторов, дирижеров и руководителей ансамблей.

Кроме того, в базе данных хранится информация о компакт-дисках, которыми торгует магазин. Каждый компакт-диск, а точнее, его наклейка, идентифицируется отдельным номером, так что всем его копиям, созданным в разное время, присвоены одинаковые номера. На компакт-диске может быть записано несколько вариантов исполнения одного и того же произведения — для каждого из них в базе заведена отдельная запись. Когда выходит новый компакт-диск, регистрируется название выпускавшей его компании (например, ЕМІ), а также адрес оптовой фирмы, у которой магазин может приобрести этот компакт-диск. Не исключено, что компания-производитель занимается и оптовой продажей компакт-дисков. Магазин фиксирует текущие оптовые и розничные цены на каждый компакт-диск, дату его выпуска, количество экземпляров, проданных за прошлый год и в нынешнем году, а также число еще не проданных компакт-дисков.

Написать запросы, осуществляющие следующие операции:

- 1) Выдать список всех джазовых групп с указанием всех, принимающих в группе участие, музыкантов.
- 2) Выдать список всех имеющихся в базе данных ансамблей и указать количество однотипных коллективов.
- 3) Выдать список всех компакт-дисков, которые были выпущены определенной компанией.
- 4) Вывести список компакт-дисков, число непроданных экземпляров которых менее 15 штук.
- 5) Вывести список всех выпущенных дисков указанного коллектива.
- 6) Выдать список компакт-дисков разница между оптовой и розничной ценой которых более 100р.

Задача 9

База данных кегельной лиги

Ставится задача спроектировать базу данных для секретаря кегельной лиги небольшого городка, расположенного на Среднем Западе США. В ней секретарь будет хранить всю информацию, относящуюся к кегельной лиге, а средствами СУБД — формировать еженедельные отчеты о состоянии лиги. Специальный отчет предполагается формировать в конце сезона.

Секретарю понадобятся фамилии и имена членов лиги, их телефонные номера и адреса. Так как в лигу могут входить только жители городка, нет необходимости хранения для каждого игрока названия города и почтового индекса. Интерес представляют число очков, набранных каждым игроком в еженедельной серии из трех встреч, в которых он принял участие, и его текущая результативность (среднее число набираемых очков в одной встрече). Секретарю необходимо знать для каждого игрока название команды, за которую он выступает, и фамилию (и имя) капитана каждой команды. Помимо названия, секретарь планирует назначить каждой команде уникальный номер.

Исходные значения результативности каждого игрока необходимы как при определении в конце сезона достигшего наибольшего прогресса в лиге игрока, так и при вычислении гандикапа для каждого игрока на первую неделю нового сезона. Лучшая игра каждого игрока и лучшие серии потребуются при распределении призов в конце сезона.

Секретарь планирует включать в еженедельные отчеты информацию об общем числе набранных очков и общем числе проведенных игр каждым игроком, эта информация используется при вычислении их текущей результативности и текущего гандикапа. Используемый в лиге гандикап составляет 75% от разности между 200 и результативностью игрока, при этом отрицательный гандикап не допускается. Если результатом вычисления гандикапа является дробная величина, то она усекается. Перерасчет гандикапа осуществляется каждую неделю.

На каждую неделю каждой команде требуется назначать площадку, на которой она будет выступать. Эту информацию хранить в БД не нужно (соперники выступают на смежных площадках).

Наконец, в БД должна содержаться вся информация, необходимая для расчета положения команд. Команде засчитывается одна победа за каждую игру, в которой ей удалось набрать больше очков (выбить больше кеглей) (с учетом гандикапа), чем команде соперников. Точно так же команде засчитывается одно поражение за каждую встречу, в которой эта команда выбила меньшее количество кеглей, чем команда соперников. Команде также засчитывается одна победа (поражение) в случае, если по сравнению с командой соперников ею набрано больше (меньше) очков за три встречи, состоявшиеся на неделе. Таким образом, на каждой неделе разыгрывается 4 командных очка (побед или поражений). В случае ничейного результата каждая команда получает 1/2 победы и 1/2 поражения. В случае неявки более чем двух членов команды их команде автоматически засчитывается 4

поражения, а команде соперников — 4 победы. В общий результат команде, которой засчитана неявка, очки не прибавляются, даже если явившиеся игроки в этой встрече выступили, однако в индивидуальные показатели — число набранных очков и проведенных встреч — будут внесены соответствующие изменения.

Предоставить возможность добавления новой команды. Предоставить возможность заполнения результатов игры двух команд на указанной площадке.

Написать запросы, осуществляющие следующие операции:

- 1) Для указанного интервала дат показать список выступающих команд. Для каждой из них вывести состав и капитана команды.
- 2) Вывести список игровых площадок с указанием количества проведенных игр на каждой из них.
- 3) Для указанного интервала дат вывести список игровых площадок с указанием списка игравших на них команд.
- 4) Вывести список площадок с указанием суммарной результативности игроков на каждой из них.

Задача 10

База данных поддерживающая информационную систему аэропорта

Предметная область представляет собой информационную систему (ИС) аэропорта, направленную на сбор и обработку информации для предоставления услуг авиаперевозок. ИС аэропорта связывает поставщика услуг - аэропорт и их потребителей – пассажиров. Для оптимального взаимодействия этих сторон необходимо автоматизировать информационные процессы, это приведет к улучшению их качества и повышению быстродействия.

Проектируемая база данных (БД) предназначена для информационной системы (ИС) диспетчеров аэропорта и обслуживающего персонала, для управления и учета вылетов самолетов аэропорта. БД должна решать довольно узкий круг задач, таких как обслуживание, сопоставление расписания с фактическими вылетами самолетов по различным направлениям.

Основной задачей стоящей перед проектируемой БД является составление расписания вылетов самолетов аэропорта под управлением соответствующих экипажей.

Все самолеты имеют различные характеристики, такие как скорость, высота, взлетная масса, бортовой номер, тип самолета, количество посадочных мест, топливо, длина разбега. Наличие этих данных является важным для службы обеспечения аэропорта.

В зависимости от марки самолета выбирается экипаж, имеющий соответствующую группу допуска управления воздушным судном. Для диспетчерской службы важным является номер экипажа. Каждый экипаж состоит из нескольких человек, каждый из которых имеет личные данные и должность. На определенный срок диспетчеры составляют плановое расписание полетов самолетов. Расписание составляется не только для диспетчерской службы аэропорта, но и для информационного обеспечения потенциальных пассажиров. Чтобы обладать достаточной информативностью для пользователей в расписание должны входить следующие данные: номер рейса, название рейса, день вылета, время вылета, время прибытия.

Непосредственная динамическая информация о состоянии полета воздушных судов по заданным направлениям должна аккумулироваться и предоставляться пользователям в наглядной и информационно полной форме. Для предоставления таких услуг со стороны БД необходимо содержание в ней отношения «Вылеты», содержащего следующие атрибуты: Код вылета, код экипажа, номер рейса, код самолета, день вылета.

Написать запросы, осуществляющие следующие операции:

- 1) Выдать расписание аэропорта на определенную дату.
- 2) Выдать список самолетов, имеющих более 300 посадочных мест.
- 3) В связи с участившимися террористическими актами для охранных служб аэропорта создается запрос на выборку – «Продажа билетов», в котором отражается информация о том, сколько и кому каждый кассир продал билетов,

на какой рейс.

- 4) Выдать информацию об экипаже, назначенном на определенный рейс.
- 5) Выдать список самолетов, совершивших полеты в определенный день.
- 6) Выдать список всех направлений, по которым осуществляются авиаперевозки.

Задача 11

База данных для учета аудиторного фонда университета

База данных должна содержать следующую информацию об аудиторном фонде университета: наименование корпуса, в котором расположено помещение, номер комнаты, расположение комнаты в корпусе, ширина и длина комнаты в метрах, назначение и вид помещения, подразделение университета, за которым закреплено помещение. В базе данных также должна быть информация о высоте потолков в помещениях (в зависимости от места расположения помещений в корпусе). Следует также учитывать, что структура подразделений университета имеет иерархический вид, когда одни подразделения входят в состав других (факультет, кафедра, лаборатория).

Помимо SQL-запросов для создания таблиц базы данных, составьте запрос на создание представления, в котором, кроме приведенной выше информации, присутствовали бы данные о площадях и объемах каждого помещения.

Написать запросы, осуществляющие следующие операции:

- 1) Выдать полную информацию о помещениях, в которых высота потолков больше 3 м.
- 2) Подсчитать площадь всех помещений, принадлежащих определенному подразделению.
- 3) Выдать список подразделений с указанием видов помещений, которыми они обладают.
- 4) Выдать список кафедр, которые обладают компьютерными классами (указать номера/названия этих классов и их площадь).
- 5) Выдать наименование корпусов, к которым находится наибольшее количество комнат с площадью больше 20 м².

Задача 12

База данных регистрации происшествий

Необходимо создать базу данных регистрации происшествий. База должна содержать:

1. данные для регистрации сообщений о происшествиях (регистрационный номер сообщения, дата регистрации, тип происшествия, краткая фабула);
2. информацию о принятом по происшествию решении (отказано в возбуждении дела, удовлетворено ходатайство о возбуждении уголовного дела с указанием регистрационного номера заведенного дела, отправлено по территориальному признаку);
3. информацию о лицах, виновных или подозреваемых в совершении происшествия (регистрационный номер лица, фамилия, имя, отчество, адрес, количество судимостей), отношение конкретных лиц к конкретным происшествиям (виновник, потерпевший, подозреваемый, свидетель).

Написать запросы, осуществляющие следующие операции:

- 1) Выдать все происшествия, зарегистрированные в определенный день.
- 2) Выдать информацию о всех лицах, совершивших кражу за последние пол года.
- 3) Выдать список потерпевших от действий конкретного лица (по всем преступлениям этого лица).
- 4) Выдать информацию по всем делам по которым «отказано в возбуждении дела».
- 5) Выдать список типов происшествий за последние пол года с указанием количества эпизодов по каждому типу происшествия.
- 6) Выдать список всех свидетелей проходивших по всем делам за последний год.

Задача 13

База данных для обслуживания работы конференции

База данных должна содержать справочник персоналий участников конференции (фамилия, имя, отчество, ученая степень, ученое звание, научное направление, место работы, кафедра (отдел), должность, страна, город, почтовый индекс, адрес, рабочий телефон, домашний телефон, e-mail) и информацию, связанную с участием в конференции (докладчик или участник, дата рассылки первого приглашения, дата поступления заявки, тема доклада, отметка о поступлении тезисов, дата рассылки второго приглашения, дата поступления оргвзноса, размер поступившего оргвзноса, дата приезда, дата отъезда, потребность в гостинице).

Написать запросы, осуществляющие следующие операции:

- 1) Выдать список участников, которые указали потребность в гостинице.
- 2) Подсчитать количество участников с ученой степенью «Доцент».
- 3) Выдать список участников конференции без доклада.
- 4) Выдать список участников, имеющих одно и тоже место работы.
- 5) Посчитать количество участников из каждого города.
- 6) Выдать список участников , оплативших оргвзнос до указанной даты.

Задача 14

База данных для обслуживания склада

База данных должна обеспечить автоматизацию складского учета. В ней должны содержаться следующие данные:

1. информация о «единицах хранения» — номер ордера, дата, код поставщика, балансовый счет, код сопроводительного документа по справочнику документов, номер сопроводительного документа, код материала по справочнику материалов, счет материала, код единицы измерения, количество пришедшего материала, цена единицы измерения;
2. информация о хранящихся на складе материалах — справочник материалов — код класса материала, код группы материала, наименование материала;
3. информация о единицах измерения конкретных видов материалов — код материала, единица измерения (метры, килограммы, литры и т.п.).
4. информация о поставщиках материалов — код поставщика, его наименование, ИНН, юридический адрес (индекс, город, улица, дом), адрес банка (индекс, город, улица, дом), номер банковского счета.

Написать запросы, осуществляющие следующие операции:

- 1) Выдать список материалов, поставляемых определенным поставщиком.
- 2) Выдать информацию о «единицах хранения», поступивших на склад за последний месяц.
- 3) Посчитать количество однотипного материала, поступившего на склад за последний месяц, в том числе от разных поставщиков.
- 4) Выдать список материалов, хранящихся на складе, единица измерения которых - килограммы.
- 5) Выдать полную информацию о поставщиках, поставляющих на склад определенный вид материалов.

Задача 15

База данных бюро по трудоустройству

В базе данных должны храниться сведения о предприятиях с их данными, сведения от предприятия (вакансии), сведения о работниках ищущих работу, сведения о специальностях по которым могут работать работники для того, чтобы осуществить поиск по специальности.

Назначение информационной системы заключается в ведении базы данных (далее - БД) о клиентах, о предприятиях, поступающих вакансий от предприятий, подобранные вакансии. Основными функциями бюро по трудоустройству являются обслуживание клиентов и предприятий, регистрация новых клиентов и предприятий, подбор работников по вакансиям, подготовка отчетов для руководства.

Входной информацией будут служить заявления соискателей, анкеты, договора, личные данные соискателя. Выходной информацией являются отчеты, выводящиеся на печать на основании запросов, вывод необходимых данных на экран в виде таблиц, диаграмм, запросов.

Написать запросы, осуществляющие следующие операции:

- 1) Выдать список всех вакансий по определенной специальности.
- 2) Выдать список всех вакансий, которые разместил указанный работодатель.
- 3) Осуществить подбор претендентов на определенную вакансию.
- 4) Вывести список всех предприятий, которые разместили вакансии, с указанием количества вакансий от каждого предприятия.
- 5) Вывести список всех предприятий, которые разместили вакансии, с указанием всех специальностей по которым требуются претенденты.

Задача 16

База данных фирмы

Фирма отказалась от приобретения некоторых товаров у своих поставщиков, решив самостоятельно наладить их производство. С этой целью она организовала сеть специализированных цехов, каждый из которых принимает определенное участие в технологическом процессе.

Каждому виду выпускаемой продукции присваивается, как обычно, свой шифр товара, под которым он значится в файле товарных запасов. Этот же номер служит и шифром продукта. В записи с этим шифром указывается, когда была изготовлена последняя партия этого продукта, какова ее стоимость, сколько операций потребовалось.

Операцией считается законченная часть процесса производства, которая целиком выполняется силами одного цеха в соответствии с техническими требованиями, перечисленными на отдельном чертеже. Для каждого продукта и для каждой операции в базе данных фирмы заведена запись, содержащая описание операции, ее среднюю продолжительность и номер, по которому можно отыскать требуемый чертеж. Кроме того, указывается номер цеха, обычно производящего данную операцию.

В запись, связанную с конкретной операцией, заносятся потребные количества расходных материалов, а также присвоенные им шифры товара. Расходными называют такие материалы, как, например, электрический кабель, который нельзя использовать повторно. При выдаче расходного материала со склада в процессе подготовки к выполнению операции регистрируется фактически выданное количество, соответствующий шифр товара, номер служащего, ответственного за выдачу, дата и время выдачи, номер операции и номер наряда на проведение работ (о котором несколько ниже). Реально затраченное количество материала может не совпадать с расчетным (например, из-за брака).

Каждый из цехов располагает требуемым инструментарием и оборудованием. При выполнении некоторых операций их иногда недостаточно, и цех вынужден обращаться в центральную инструментальную за недостающими. Каждый тип инструмента снабжен отдельным номером, и на него заведена запись со словесным описанием. Кроме того, отмечается, какое количество инструментов этого типа выделено цехам и какое осталось в инструментальной. Экземпляры инструмента конкретного типа, например, гаечные ключи одного размера, различаются по своим индивидуальным номерам. На фирме для каждого типа инструмента имеется запись, содержащая перечень всех индивидуальных номеров. Кроме того, указаны даты их поступления на склад.

По каждой операции отмечают типы и количество инструментов тех типов, которые должны использоваться при ее выполнении. Когда инструменты действительно берутся со склада, фиксируется индивидуальный номер каждого экземпляра, указываются номер заказавшего их цеха и номер наряда на проведение работ. И в этом случае затребованное количество не всегда совпадает с заказанным.

Наряд на проведение работ по форме напоминает заказ на приобретение товаров, но, в отличие от последнего, направляется не поставщику, а в один из цехов. Оформляется наряд после того, как руководство фирмы сочтет необходимым выпустить партию некоторого продукта. В наряд заносятся шифр продукта, дата оформления наряда, срок, к которому должен быть выполнен заказ, а также требуемое количество продукта.

Разработайте структуру таблиц базы данных, подберите имена таблиц и полей, в которых могла бы разместиться вся эта информация.

Напишите SQL-запросы, осуществляющие следующие операции:

1) Для выбранного цеха выдать список выполняемых им операций. Для каждой операции показать список расходных материалов с указанием количества.

2) Показать список инструментов и предоставить возможность добавления нового.

3) Выдать список используемых инструментов, отсортированных по количеству их использования в различных нарядах.

4) Для указанного интервала дат вывести список нарядов в хронологическом порядке, для каждого из которых показать список используемых инструментов.

5) Показать список операций и предоставить возможность добавления новой операции.

6) Выдать список расходуемых материалов, отсортированных по количеству их использования в различных нарядах.

7) Выдать список товаров с указанием используемых при их изготовлении инструментов.

8) Показать список нарядов в хронологическом порядке и предоставить возможность добавления нового.

9) Выдать отчет о производстве товаров различными цехами, указав наименование цеха, название товара и его количество.

Литература

1. Проектирование и реализация баз данных в СУБД MySQL с использованием MySQL Workbench: учебное пособие/С.А. Мартишин, В.Л. Симонов, М.В. Храпченко.- Москва: ИД «Форум »:ИНФРА-М, 2021.-160с.
2. Базы данных: практикум/ А.С. Копырин. - Москва: ФЛИНТА. 2021. -106с.
3. MySQL 5.0. Библиотека программиста / В. Гольцман — «Питер», 2010.-268с.