ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.В.ДВ.10.01 SQL-программирование

Направление подготовки *10.03.01 Информационная безопасность*

Профиль образовательной программы *Безопасность автоматизированных систем*

Форма обучения *очная*

СОДЕРЖАНИЕ

1. Конспект лекций……3

1.1 Лекция № 1, 2 *Определение структурированного языка запросов SQL*……3

1.2 Лекция № 3, 4 *Эффективное выполнение запросов для извлечения данных*……5

1.3 Лекция № 5, 6 *Построение нетривиальных запросов*……7

1.4 Лекция № 7, 8 *Запросы модификации данных в реляционной таблице* **.....**9

1.5 Лекция № 9, 10 *Понятие представлений* ……11

1.6 Лекция № 11, 12 *Определение функций пользователя, примеры их создания и использования*……13

1.7 Лекция № 13, 14 *Хранимые процедуры*……15

1.8 Лекция № 15, 16 *Триггеры*……17

2. Методические материалы по проведению практических занятий……19

2.1 Практическое занятие №1, 2, 3, 4 ПЗ-1, 2, 3, 4 Определение структурированного языка запросов SQL.……19

2.2 Практическое занятие №5, 6, 7, 8 ПЗ-5, 6, 7, 8 Эффективное выполнение запросов для извлечения данных.……23

2.3 Практическое занятие №9, 10, 11, 12 ПЗ-9, 10, 11, 12 Построение нетривиальных запросов.……27

2.4 Практическое занятие №13, 14, 15, 16 ПЗ-13, 14, 15, 16 Запросы модификации данных в реляционной таблице.……31

2.5 Практическое занятие №17, 18, 19, 20 ПЗ-17, 18, 19, 20 Понятие представлений. ……34

2.6 Практическое занятие №21, 22, 23, 24 ПЗ-21, 22, 23, 24 Определение функций пользователя, примеры их создания и использования.……37

2.7 Практическое занятие №25, 26, 27, 28 ПЗ-25, 26, 27, 28 Хранимые процедуры. ……40

2.8 Практическое занятие №29, 30, 31, 32 ПЗ-29, 30, 31, 32 Триггеры.……42

1. КОНСПЕКТ ЛЕКЦИЙ

1. 1 Лекция №1, 2 (4 часа).

Тема: «Определение структурированного языка запросов SQL»

1.1.1 Вопросы лекции:

1.Определение структурированного языка запросов SQL.
2. Реляционная база данных, СУБД.
3.Классификация команд SQL.

1.1.2 Краткое содержание вопросов:

1.Определение структурированного языка запросов SQL.
2. Реляционная база данных, СУБД.
3.Классификация команд SQL.

Основные понятия

Всякая профессиональная деятельность так или иначе связана с информацией, с организацией ее сбора, хранения, выборки. Можно сказать, что неотъемлемой частью повседневной жизни стали базы данных, для поддержки которых требуется некоторый организационный метод, или механизм. Такой механизм называется системой управления базами данных (СУБД). Итак, введем основные понятия.

База данных (БД) – совместно используемый набор логически связанных данных (и их описание), предназначенный для удовлетворения информационных потребностей организации.

СУБД (система управления базами данных) – программное обеспечение, с помощью которого пользователи могут определять, создавать и поддерживать базу данных, а также получать к ней контролируемый доступ.

Системы управления базами данных существуют уже много лет, многие из них обязаны своим происхождением системам с неструктурированными файлами на больших ЭВМ. Наряду с общепринятыми современными технологиями в области систем управления базами данных начинают появляться новые направления, что обусловлено требованиями растущего бизнеса, все увеличивающимися объемами корпоративных данных и, конечно же, влиянием технологий Internet.

Реляционные базы данных

Управление основными потоками информации осуществляется с помощью так называемых *систем управления реляционными базами данных*, которые берут свое начало в традиционных системах управления базами данных. Именно объединение *реляционных* баз данных и *клиент-серверных* технологий позволяет современному предприятию успешно управлять собственными данными, оставаясь конкурентоспособным на рынке товаров и услуг.

Реляционные БД имеют мощный теоретический фундамент, основанный на математической теории отношений. Появление теории реляционных баз данных дало толчок к разработке *ряда языков* запросов, которые можно отнести к *двум классам:*

- алгебраические языки, позволяющие выражать запросы средствами специализированных операторов, применяемых к отношениям;
- языки исчисления предикатов, представляющие собой набор правил для записи выражения, определяющего новое отношение из заданной совокупности существующих отношений.

Следовательно, *исчисление предикатов* есть метод определения того отношения, которое желательно получить как ответ на запрос из отношений, уже имеющихся в базе данных.

В реляционной модели объекты реального мира и взаимосвязи между ними представляются с помощью совокупности связанных между собой таблиц (отношений). Даже в том случае, когда функции СУБД используются для выбора информации из одной или нескольких таблиц (т.е. выполняется запрос), результат также представляется в табличном виде. Более того, можно выполнить запрос с применением результатов другого запроса.

Каждая таблица БД представляется как совокупность строк и столбцов, где строки (записи) соответствуют экземпляру объекта, конкретному событию или явлению, а столбцы (поля) – атрибутам (признакам, характеристикам, параметрам) объекта, события, явления.

В каждой таблице БД необходимо наличие

первичного ключа – так именуют поле или набор полей, однозначно идентифицирующий каждый экземпляр объекта или запись.

Значение первичного ключа в таблице БД должно быть уникальным, т.е. в таблице не допускается наличие двух и более записей с одинаковыми значениями первичного ключа. Он должен быть минимально достаточным, а значит, не содержать полей, удаление которых не отразится на его уникальности.

Реляционные связи между таблицами баз данных

Связи между объектами реального мира могут находить свое отражение в структуре данных, а могут и подразумеваться, т.е. присутствовать на неформальном уровне. Между двумя или более таблицами базы данных могут существовать *отношения подчиненности*, которые определяют, что для каждой записи главной таблицы (**называемой еще родительской**) возможно наличие одной или нескольких записей в подчиненной таблице (**называемой еще дочерней**).

Выделяют три разновидности связи между таблицами базы данных:

- "один–ко–многим";
- "один–к–одному";
- "многие–ко–многим".

Отношение "один–ко–многим"

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

Отношение "один–к–одному"

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

Отношение "многие–ко–многим"

Отношение "**многие–ко–многим**" применяется в следующих случаях:

- одной записи в родительской таблице соответствует более одной записи в дочерней;
- одной записи в дочерней таблице соответствует более одной записи в родительской.

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

Стандарт и реализация языка SQL

Рост количества данных, необходимость их хранения и обработки привели к тому, что возникла потребность в создании стандартного языка баз данных, который мог бы функционировать в многочисленных компьютерных системах различных видов.

Одним из языков, появившихся в результате разработки реляционной модели данных, является язык **SQL (StructuredQueryLanguage)**, который в настоящее время получил очень широкое распространение и фактически превратился в стандартный язык реляционных баз данных.

С использованием любых стандартов связаны не только многочисленные и вполне очевидные преимущества, но и определенные недостатки.

Под **реализацией языка SQL** понимается программный продукт SQL соответствующего производителя.

В настоящее время язык SQL поддерживается многими десятками СУБД различных типов, разработанных для самых разнообразных вычислительных платформ, начиная от персональных компьютеров и заканчивая мейнфреймами.

Рассматриваемый язык **SQL ориентирован** на операции с данными, представленными в виде логически взаимосвязанных совокупностей *таблиц-отношений.*

Важнейшая **особенность его структур** – ориентация на конечный результат обработки данных, а не на процедуру этой обработки.

Язык SQL сам определяет, где находятся данные, индексы и даже какие наиболее эффективные последовательности операций следует использовать для получения результата, а потому указывать эти детали в запросе к базе данных не требуется.

1. 2 Лекция №3, 4 (4 часа).

Тема: «Эффективное выполнение запросов для извлечения данных.»

1.2.1 Вопросы лекции:

1.Эффективное выполнение запросов для извлечения данных.
2. Синтаксис оператора SELECT.
3.Построение условий выбора данных с применением операторов сравнения, логических операторов и логических связок.

1.2.2 Краткое содержание вопросов:

1.Эффективное выполнение запросов для извлечения данных.
2. Синтаксис оператора SELECT.
3.Построение условий выбора данных с применением операторов сравнения, логических операторов и логических связок.

Предложение SELECT

Оператор SELECT – один из наиболее важных и самых распространенных операторов SQL. Он позволяет производить выборки данных из таблиц и преобразовывать к нужному виду полученные результаты. Будучи очень мощным, он способен выполнять действия, эквивалентные операторам реляционной алгебры, причем в пределах единственной выполняемой команды. При его помощи можно реализовать сложные и громоздкие условия отбора данных из различных таблиц.

Оператор SELECT – средство, которое полностью абстрагировано от вопросов представления данных, что помогает сконцентрировать внимание на проблемах доступа к данным.

Примеры его использования наглядно демонстрируют один из основополагающих принципов больших (промышленных) СУБД: средства хранения данных и доступа к ним

отделены от средств представления данных. Операции над данными производятся в масштабе наборов данных, а не отдельных записей.

Оператор SELECT имеет следующий формат:

SELECT [ALL | DISTINCT] {*|[имя_столбца
[AS новое_имя]]} [,...n]
FROM имя_таблицы [[AS] псевдоним] [,...n]
[WHERE <условие_поиска>]
[GROUPBYимя_столбца [,...n]]
[HAVING <критерии выбора групп>]
[ORDER BY имя_столбца [,...n]]

Оператор SELECT определяет поля (столбцы), которые будут входить в результат выполнения запроса. В списке они разделяются запятыми и приводятся в такой очередности, в какой должны быть представлены в результате запроса. Если используется имя поля, содержащее пробелы или разделители, его следует заключить в квадратные скобки. Символом * можно выбрать все поля, а вместо имени поля применить выражение из нескольких имен. Если обрабатывается ряд таблиц, то (при наличии одноименных полей в разных таблицах) в списке полей используется полная спецификация поля, т.е. Имя_таблицы.Имя_поля.

Предложение FROM

Предложение FROM задает имена таблиц и представлений, которые содержат поля, перечисленные в операторе SELECT. Необязательный параметр псевдонима – это сокращение, устанавливаемое для имени таблицы.

Обработка элементов оператора SELECT выполняется в следующей последовательности:

- FROM – определяются имена используемых таблиц;
- WHERE – выполняется фильтрация строк объекта в соответствии с заданными условиями;
- GROUP BY – образуются группы строк , имеющих одно и то же значение в указанном столбце;
- HAVING – фильтруются группы строк объекта в соответствии с указанным условием;
- SELECT – устанавливается, какие столбцы должны присутствовать в выходных данных;
- ORDER BY – определяется упорядоченность результатов выполнения операторов.

Порядок предложений и фраз в операторе SELECT не может быть изменен. Только два предложения SELECT и FROM являются обязательными, все остальные могут быть опущены. SELECT – закрытая операция: результат запроса к таблице представляет собой другую таблицу. Существует множество вариантов записи данного оператора, что иллюстрируется приведенными ниже примерами.

Пример 4.1.*Составить список сведений о всех клиентах.*

SELECT * FROM Клиент

Пример 4.1. Список сведений о всех клиентах.

Параметр WHERE определяет критерий отбора записей из входного набора. Но в таблице могут присутствовать повторяющиеся записи (дубликаты). Предикат ALL задает включение в выходной набор всех дубликатов, отобранных по критерию WHERE. Нет необходимости указывать ALL явно, поскольку это значение действует по умолчанию.

Пример 4.2. Составить список всех фирм.

SELECT ALL Клиент.Фирма FROM Клиент
Или (что эквивалентно)
SELECT Клиент.Фирма FROM Клиент

Результат выполнения запроса может содержать дублирующиеся значения, поскольку в отличие от операций реляционной алгебры оператор SELECT не исключает повторяющихся значений при выполнении выборки данных.

Предикат DISTINCT следует применять в тех случаях, когда требуется отбросить блоки данных, содержащие дублирующие записи в выбранных полях. Значения для каждого из приведенных в инструкции SELECT полей должны быть уникальными, чтобы содержащая их запись смогла войти в выходной набор.

Причиной ограничения в применении DISTINCT является то обстоятельство, что его использование может резко замедлить выполнение запросов.

Откорректированный пример 4.2 выглядит следующим образом:
SELECT DISTINCT Клиент.Фирма
FROM Клиент

1. 3 Лекция №5, 6 (4 часа).

Тема: «Построение нетривиальных запросов.»

1.3.1 Вопросы лекции:

1.Построение нетривиальных запросов.
2. Способ построения подзапросов, возвращающих множественные и единичные значения с использованием операторов EXISTS, ALL, ANY.

1.3.2 Краткое содержание вопросов:

1.Построение нетривиальных запросов.
2. Способ построения подзапросов, возвращающих множественные и единичные значения с использованием операторов EXISTS, ALL, ANY.

Понятие подзапроса

Часто невозможно решить поставленную задачу путем одного запроса. Это особенно актуально, когда при использовании условия поиска в предложении **WHERE** значение, с которым надо сравнивать, заранее не определено и должно быть вычислено в момент выполнения оператора SELECT. В таком случае приходят на помощь законченные операторы SELECT, внедренные в тело другого оператора SELECT. Внутренний подзапрос представляет собой также оператор SELECT, а кодирование его предложений подчиняется тем же правилам, что и основного оператора SELECT.

Внешний оператор SELECT использует результат выполнения внутреннего оператора для определения содержания окончательного результата всей операции. Внутренние запросы могут быть помещены непосредственно после оператора сравнения (=, <, >, <=, >=, <>) в предложения WHERE и HAVING внешнего оператора SELECT – они получают название подзапросов или вложенных запросов. Кроме того, внутренние операторы SELECT могут применяться в операторах INSERT, UPDATE и DELETE.

Подзапрос – это инструмент создания временной таблицы, содержимое которой извлекается и обрабатывается внешним оператором. Текст подзапроса должен быть заключен в скобки.

К подзапросам применяются следующие правила и ограничения:

 фраза ORDER BY не используется, хотя и может присутствовать во внешнем подзапросе;

 список в предложении SELECT состоит из имен отдельных столбцов или составленных из них выражений – за исключением случая, когда в подзапросе присутствует ключевое слово EXISTS;
 по умолчанию имена столбцов в подзапросе относятся к таблице, имя которой указано в предложении FROM. Однако допускается ссылка и на столбцы таблицы, указанной во фразе FROM внешнего запроса, для чего применяются квалифицированные имена столбцов (т.е. с указанием таблицы);
 если подзапрос является одним из двух операндов, участвующих в операции сравнения, то запрос должен указываться в правой части этой операции.

Существует два типа подзапросов:

 Скалярный подзапрос возвращает единственное значение. В принципе, он может использоваться везде, где требуется указать единственное значение.
 Табличный подзапрос возвращает множество значений, т.е. значения одного или нескольких столбцов таблицы, размещенные в более чем одной строке. Он возможен везде, где допускается наличие таблицы.

Использование подзапросов, возвращающих единичное значение

Пример 7.1. Определить дату продажи максимальной партии товара.

```
SELECT Дата, Количество
        FROM Сделка
        WHERE Количество=(SELECT Max(Количество) FROM Сделка)
```

Во вложенном подзапросе определяется максимальное количество товара. Во внешнем подзапросе – дата, для которой количество товара оказалось равным максимальному. Необходимо отметить, что нельзя прямо использовать предложение *WHERE Количество=Max(Количество)*, поскольку применять обобщающие функции в предложениях WHERE запрещено. Для достижения желаемого результата следует создать подзапрос, вычисляющий максимальное значение количества, а затем использовать его во внешнем операторе SELECT, предназначенном для выборки дат сделок, где количество товара совпало с максимальным значением.

Пример 7.2. Определить даты сделок, превысивших по количеству товара среднее значение и указать для этих сделок превышение над средним уровнем.

```
SELECT Дата, Количество,
Количество-(SELECT Avg(Количество)
                FROM Сделка) AS Превышение
FROM Сделка
WHERE Количество>
                (SELECT Avg(Количество)
        FROM Сделка)
```

В приведенном примере результат подзапроса, представляющий собой среднее значение количества товара по всем сделкам вообще, используется во внешнем операторе SELECT как для вычисления отклонения количества от среднего уровня, так и для отбора сведений о датах.

Пример 7.3. Определить клиентов, совершивших сделки с максимальным количеством товара.

```
SELECT Клиент.Фамилия
FROM Клиент INNERJOIN Сделка
ON Клиент.КодКлиента=Сделка.КодКлиента
WHERE Сделка.Количество=
```

(SELECT Max(Сделка.Количество)
FROM Сделка)

Здесь показан пример использования подзапроса при выборке данных из разных таблиц.

1. 4 Лекция №7, 8 (4 часа).

Тема: «Запросы модификации данных в реляционной таблице»

1.4.1 Вопросы лекции:

1.Запросы модификации данных в реляционной таблице.
2. Целостность данных.
3.Целостность сущностей и ссылочная целостность.

1.4.2 Краткое содержание вопросов:

1.Запросы модификации данных в реляционной таблице.
2. Целостность данных.
3.Целостность сущностей и ссылочная целостность.

Язык SQL ориентирован на выполнение операций над группами записей, хотя в некоторых случаях их можно проводить и над отдельной записью.
Запросы действия представляют собой достаточно мощное средство, так как позволяют оперировать не только отдельными строками, но и набором строк. С помощью запросов действия пользователь может добавить, удалить или обновить блоки данных.
Существует три вида запросов действия:
 INSERT INTO – запрос добавления;
 DELETE – запрос удаления;
 UPDATE – запрос обновления.

Запрос добавления

Оператор INSERT применяется для добавления записей в таблицу. Формат оператора:
<оператор_вставки>::=INSERT INTO <имя_таблицы>
[(имя_столбца [,...n])]
{VALUES (значение[,...n])|
<SELECT_оператор>}

Здесь параметр имя_таблицы представляет собой либо имя таблицы базы данных, либо имя обновляемого представления.

Первая форма оператора INSERT с параметром VALUES предназначена для вставки единственной строки в указанную таблицу. Список столбцов указывает столбцы, которым будут присвоены значения в добавляемых записях. Список может быть опущен, тогда подразумеваются все столбцы таблицы (кроме объявленных как счетчик), причем в определенном порядке, установленном при создании таблицы. Если в операторе INSERT указывается конкретный список имен полей, то любые пропущенные в нем столбцы должны быть объявлены при создании таблицы как допускающие значение NULL, за исключением тех случаев, когда при описании столбца использовался параметр DEFAULT. *Список значений должен следующим образом соответствовать списку столбцов:*
 количество элементов в обоих списках должно быть одинаковым;
 должно существовать прямое соответствие между позицией одного и того же элемента в обоих списках, поэтому первый элемент списка значений должен относиться к первому столбцу в списке столбцов, второй – ко второму столбцу и т.д.

 типы данных элементов в списке значений должны быть совместимы с типами данных соответствующих столбцов таблицы.

Пример 8.1. Добавить в таблицу ТОВАР новую запись.

```
INSERT INTO Товар (Название, Тип, Цена)
        VALUES(" Славянский ", " шоколад ", 12)
```

Если столбцы таблицы ТОВАР указаны в полном составе и в том порядке, в котором они перечислены при создании таблицы ТОВАР, оператор можно упростить.

```
INSERT INTO Товар VALUES (" Славянский ",
" шоколад ", 12)
```

Вторая форма оператора INSERT с параметром SELECT позволяет скопировать множество строк из одной таблицы в другую. Предложение SELECT может представлять собой любой допустимый оператор SELECT. Вставляемые в указанную таблицу строки в точности должны соответствовать строкам результирующей таблицы, созданной при выполнении вложенного запроса. Все ограничения, указанные выше для первой формы оператора SELECT, применимы и в этом случае.

Поскольку оператор SELECT в общем случае возвращает множество записей, то оператор INSERT в такой форме приводит к добавлению в таблицу аналогичного числа новых записей.

Пример 8.2. Добавить в итоговую таблицу сведения об общей сумме ежемесячных продаж каждого наименования товара.

```
INSERT INTO Итог
        (Название, Месяц, Стоимость )
SELECT Товар.Название, Month(Сделка.Дата)
        AS Месяц, Sum(Товар.Цена*Сделка.Количество)
        AS Стоимость
FROM Товар INNER JOIN Сделка
        ON Товар.КодТовара= Сделка.КодТовара
GROUP BY Товар.Название, Month(Сделка.Дата)
```

1. 5 Лекция №9, 10 (4 часа).

Тема: «Понятие представлений»

1.5.1 Вопросы лекции:

1.Понятие представлений.
2.Роль представлений в вопросах безопасности данных.
3.Процесс управления представлениями: создание, изменение, применение, удаление представлений.

1.5.2 Краткое содержание вопросов:

1.Понятие представлений.
2.Роль представлений в вопросах безопасности данных.
3.Процесс управления представлениями: создание, изменение, применение, удаление представлений.

Определение представления

Представления, или просмотры (VIEW), представляют собой временные, производные (иначе - виртуальные) таблицы и являются объектами базы данных, информация в которых не хранится постоянно, как в базовых таблицах, а формируется динамически при обращении к ним. Обычные таблицы относятся к **базовым,** т.е. содержащим данные

и постоянно находящимся на устройстве хранения информации. Представление не может существовать само по себе, а определяется только в терминах одной или нескольких таблиц. Применение представлений позволяет разработчику базы данных обеспечить каждому пользователю или группе пользователей наиболее подходящие способы работы с данными, что решает проблему простоты их использования и безопасности. Содержимое представлений выбирается из других таблиц с помощью выполнения запроса, причем при изменении значений в таблицах данные в представлении автоматически меняются. **Представление** - это фактически тот же запрос, который выполняется всякий раз при участии в какой-либо команде. Результат выполнения этого запроса в каждый момент времени становится **содержанием** представления. У пользователя создается впечатление, что он работает с настоящей, реально существующей таблицей.

У СУБД есть две возможности реализации представлений. Если его определение простое, то система формирует каждую запись представления по мере необходимости, постепенно считывая исходные данные из базовых таблиц. В случае сложного определения СУБД приходится сначала выполнить такую операцию, как материализация представления, т.е. сохранить информацию, из которой состоит представление, во временной таблице. Затем система приступает к выполнению пользовательской команды и формированию ее результатов, после чего временная таблица удаляется.

Представление - это предопределенный запрос, хранящийся в базе данных, который выглядит подобно обычной таблице и не требует для своего хранения дисковой памяти. Для хранения представления используется только оперативная память. В отличие от других объектов базы данных представление не занимает дисковой памяти за исключением памяти, необходимой для хранения определения самого представления.
Создания и изменения представлений в стандарте языка и реализации в MS SQL Server совпадают и представлены следующей командой:

<определение_представления> ::=

{ CREATE| ALTER} VIEW имя_представления
[(имя_столбца [,...n])]
[WITH ENCRYPTION]
AS SELECT_оператор
[WITH CHECK OPTION]

Рассмотрим назначение основных параметров.

По умолчанию имена столбцов в представлении соответствуют именам столбцов в исходных таблицах. Явное указание имени столбца требуется для вычисляемых столбцов или при объединении нескольких таблиц, имеющих столбцы с одинаковыми именами. Имена столбцов перечисляются через запятую, в соответствии с порядком их следования в представлении.

Параметр WITH ENCRYPTION предписывает серверу шифровать SQL-код запроса, что гарантирует невозможность его несанкционированного просмотра и использования. Если при определении представления необходимо скрыть имена исходных таблиц и столбцов, а также алгоритм объединения данных, необходимо применить этот аргумент.

Параметр WITH CHECK OPTION предписывает серверу исполнять проверку изменений, производимых через представление, на соответствие критериям, определенным в операторе SELECT. Это означает, что не допускается выполнение изменений, которые приведут к исчезновению строки из представления. Такое случается, если для представления установлен горизонтальный фильтр и изменение данных приводит к несоответствию строки установленным фильтрам. Использование аргумента WITH CHECK OPTION гарантирует, что сделанные изменения будут отображены в представлении. Если пользователь пытается выполнить изменения, приводящие к

исключению строки из представления, при заданном аргументе WITH CHECK OPTION сервер выдаст сообщение об ошибке и все изменения будут отклонены.

Пример 10.1. Показать в представлении клиентов из Москвы.

Создание представления:

```
CREATE VIEW view1 AS
SELECT КодКлиента, Фамилия, ГородКлиента
FROM Клиент
WHERE ГородКлиента='Москва'
```

Выборка данных из представления:

```
SELECT * FROM view1
```

1. 6 Лекция №11, 12 (4 часа).

Тема: «Определение функций пользователя, примеры их создания и использования»

1.6.1 Вопросы лекции:

1. Определение функций пользователя, примеры их создания ииспользования.
2. Типы функций.
3.Встроенныефункции языка SQL.

1.6.2 Краткое содержание вопросов:

1. Определение функций пользователя, примеры их создания ииспользования.
2. Типы функций.
3.Встроенныефункции языка SQL.

Понятие функции пользователя

При реализации на языке SQL сложных алгоритмов, которые могут потребоваться более одного раза, сразу встает вопрос о сохранении разработанного кода для дальнейшего применения. Эту задачу можно было бы реализовать с помощью хранимых процедур, однако их архитектура не позволяет использовать процедуры непосредственно в выражениях, т.к. они требуют промежуточного присвоения возвращенного значения переменной, которая затем и указывается в выражении. Естественно, подобный метод применения программного кода не слишком удобен. Многие разработчики уже давно хотели иметь возможность вызова разработанных алгоритмов непосредственно в выражениях.

Возможность создания пользовательских функций была предоставлена в среде MS SQL Server 2000. В других реализациях SQL в распоряжении пользователя имеются только встроенные функции, которые обеспечивают выполнение наиболее распространенных алгоритмов: поиск максимального или минимального значения и др.

Функции пользователя представляют собой самостоятельные объекты базы данных, такие, например, как хранимые процедуры или триггеры. Функция пользователя располагается в определенной базе данных и доступна только в ее контексте.

В SQL Server имеются следующие классы функций пользователя:

 Scalar – функции возвращают обычное скалярное значение, каждая может включать множество команд, объединяемых в один блок с помощью конструкции BEGIN...END;

 Inline– функции содержат всего одну команду SELECT и возвращают пользователю набор данных в виде значения типа данных TABLE;

 Multi-statement – функции также возвращают пользователю значение типа данных TABLE, содержащее набор данных, однако в теле функции находится множество команд SQL (INSERT, UPDATE и т.д.).

Именно с их помощью и формируется набор данных, который должен быть возвращен после выполнения функции.

Пользовательские функции сходны с хранимыми процедурами, но, в отличие от них, могут применяться в запросах так же, как и системные встроенные функции.

Пользовательские функции, возвращающие таблицы, могут стать альтернативой просмотрам. Просмотры ограничены одним выражением SELECT, а пользовательские функции способны включать дополнительные выражения, что позволяет создавать более сложные и мощные конструкции.

Функции Scalar

Создание и изменение функции данного типа выполняется с помощью команды:

```
<определение_скаляр_функции>::=
{CREATE | ALTER } FUNCTION [владелец.]
    имя_функции
( [ { @имя_параметраскаляр_тип_данных
[=default]}[,...n]])
RETURNS скаляр_тип_данных
[WITH {ENCRYPTION | SCHEMABINDING}
    [,...n] ]
[AS]
BEGIN
<тело_функции>
RETURN скаляр_выражение
END
```

Рассмотрим назначение параметров команды.

Функция может содержать один или несколько входных параметров либо не содержать ни одного. Каждый параметр должен иметь уникальное в пределах создаваемой функции имя и начинаться с символа "@". После имени указывается тип данных параметра. Дополнительно можно указать значение, которое будет автоматически присваиваться параметру (DEFAULT), если пользователь явно не указал значение соответствующего параметра при вызове функции.

С помощью конструкции RETURNS скаляр_тип_данных указывается, какой тип данных будет иметь возвращаемое функцией значение. Дополнительные параметры, с которыми должна быть создана функция, могут быть указаны посредством ключевого слова WITH. Благодаря ключевому слову ENCRYPTION код команды, используемый для создания функции, будет зашифрован, и никто не сможет просмотреть его. Эта возможность позволяет скрыть логику работы функции. Кроме того, в теле функции может выполняться обращение к различным объектам базы данных, а потому изменение или удаление соответствующих объектов может привести к нарушению работы функции.

Чтобы избежать этого, требуется запретить внесение изменений, указав при создании этой функции ключевое слово SCHEMABINDING. Между ключевыми словами BEGIN...END указывается набор команд, они и будут являться телом функции. Когда в ходе выполнения кода функции встречается ключевое слово RETURN, выполнение функции завершается и как результат ее вычисления возвращается значение, указанное непосредственно после слова RETURN. Отметим, что в теле функции разрешается использование множества команд RETURN, которые могут возвращать различные значения. В качестве возвращаемого значения допускаются как обычные константы, так и сложные выражения. Единственное условие – тип данных возвращаемого значения должен совпадать с типом данных, указанным после ключевого слова RETURNS.

Пример 11.1. Создать и применить функцию скалярного типа для вычисления суммарного количества товара, поступившего за определенную дату. Владелец функции – пользователь с именем user1.

```
CREATE FUNCTION
    user1.sales(@data DATETIME)
```

```
RETURNS INT
AS
BEGIN
DECLARE @c INT
SET @c=(SELECT SUM(количество)
        FROM Сделка
        WHERE дата=@data)
RETURN (@c)
END
```

В качестве входного параметра используется дата. Функция возвращает значение целого типа, полученное из оператора SELECT путем суммирования количества товара из таблицы **Сделка.** Условием отбора записей для суммирования является равенство даты сделки значению входного параметра функции.

Проиллюстрируем обращение к функции пользователя: определим количество товара, поступившего за 02.11.01:

```
DECLARE @kol INT
SET @kol=user1.sales ('02.11.01')
SELECT @kol
```

1. 7 Лекция №13, 14 (4 часа).

Тема: «Хранимые процедуры.»

1.7.1 Вопросы лекции:

1.Понятие хранимых процедур.

2.Примеры создания, изменения и использования хранимых процедур с параметрами.

3.Определение входных и выходных параметров. 4.Примеры создания и вызова хранимых процедур.

1.7.2 Краткое содержание вопросов:

1.Понятие хранимых процедур.

2.Примеры создания, изменения и использования хранимых процедур с параметрами.

3.Определение входных и выходных параметров. 4.Примеры создания и вызова хранимых процедур.

Хранимые процедуры представляют собой группы связанных между собой операторов SQL, применение которых делает работу программиста более легкой и гибкой, поскольку выполнить хранимую процедуру часто оказывается гораздо проще, чем последовательность отдельных операторов SQL. Хранимые процедуры представляют собой набор команд, состоящий из одного или нескольких операторов SQL или функций и сохраняемый в базе данных в откомпилированном виде. Выполнение в базе данных хранимых процедур вместо отдельных операторов SQL дает пользователю следующие преимущества:

 необходимые операторы уже содержатся в базе данных;

 все они прошли этап синтаксического анализа и находятся в исполняемом формате; перед выполнением хранимой процедуры SQL Server генерирует для нее план исполнения, выполняет ее оптимизацию и компиляцию;

 хранимые процедуры поддерживают модульное программирование, так как позволяют разбивать большие задачи на самостоятельные, более мелкие и удобные в управлении части;

 хранимые процедуры могут вызывать другие хранимые процедуры и функции;

 хранимые процедуры могут быть вызваны из прикладных программ других типов;
 как правило, хранимые процедуры выполняются быстрее, чем последовательность отдельных операторов;
 хранимые процедуры проще использовать: они могут состоять из десятков и сотен команд, но для их запуска достаточно указать всего лишь имя нужной хранимой процедуры. Это позволяет уменьшить размер запроса, посылаемого от клиента на сервер, а значит, и нагрузку на сеть.

Хранение процедур в том же месте, где они исполняются, обеспечивает уменьшение объема передаваемых по сети данных и повышает общую производительность системы. Применение хранимых процедур упрощает сопровождение программных комплексов и внесение изменений в них. Обычно все ограничения целостности в виде правил и алгоритмов обработки данных реализуются на сервере баз данных и доступны конечному приложению в виде набора хранимых процедур, которые и представляют интерфейс обработки данных. Для обеспечения целостности данных, а также в целях безопасности, приложение обычно не получает прямого доступа к данным – вся работа с ними ведется путем вызова тех или иных хранимых процедур.

Подобный подход делает весьма простой модификацию алгоритмов обработки данных, тотчас же становящихся доступными для всех пользователей сети, и обеспечивает возможность расширения системы без внесения изменений в само приложение: достаточно изменить хранимую процедуру на сервере баз данных. Разработчику не нужно перекомпилировать приложение, создавать его копии, а также инструктировать пользователей о необходимости работы с новой версией. Пользователи вообще могут не подозревать о том, что в систему внесены изменения.

Хранимые процедуры существуют независимо от таблиц или каких-либо других объектов баз данных. Они вызываются клиентской программой, другой хранимой процедурой или триггером. Разработчик может управлять правами доступа к хранимой процедуре, разрешая или запрещая ее выполнение. Изменять код хранимой процедуры разрешается только ее владельцу или члену фиксированной роли базы данных. При необходимости можно передать права владения ею от одного пользователя к другому.

Хранимые процедуры в среде MS SQL Server

При работе с SQL Server пользователи могут создавать собственные процедуры, реализующие те или иные действия. Хранимые процедуры являются полноценными объектами базы данных, а потому каждая из них хранится в конкретной базе данных. Непосредственный вызов хранимой процедуры возможен, только если он осуществляется в контексте той базы данных, где находится процедура.

Типы хранимых процедур

В SQL Server имеется несколько типов хранимых процедур.

 Системные хранимые процедуры предназначены для выполнения различных административных действий. Практически все действия по администрированию сервера выполняются с их помощью. Можно сказать, что системные хранимые процедуры являются интерфейсом, обеспечивающим работу с системными таблицами, которая, в конечном счете, сводится к изменению, добавлению, удалению и выборке данных из системных таблиц как пользовательских, так и системных баз данных. Системные хранимые процедуры имеют префикс **sp_**, хранятся в системной базе данных и могут быть вызваны в контексте любой другой базы данных.
 Пользовательские хранимые процедуры реализуют те или иные действия. **Хранимые процедуры** – полноценный объект базы данных. Вследствие этого каждая хранимая процедура располагается в конкретной базе данных, где и выполняется.
 Временные хранимые процедуры существуют лишь некоторое время, после чего автоматически уничтожаются сервером.

Они делятся на **локальные** и **глобальные.**

Локальные временные хранимые процедуры могут быть вызваны только из того соединения, в котором созданы. При создании такой процедуры ей необходимо дать имя, начинающееся с одного символа **#.** Как и все временные объекты, хранимые процедуры этого типа автоматически удаляются при отключении пользователя, перезапуске или остановке сервера.

Глобальные временные хранимые процедуры доступны для любых соединений сервера, накотором имеется такая же процедура. Для ее определения достаточно дать ейимя, начинающееся с символов ##. Удаляются эти процедуры при перезапускеили остановке сервера, а также при закрытии соединения, в контексте которогоони были созданы.

Создание, изменение и удаление хранимых процедур

Создание хранимой процедуры предполагает решение следующих задач:

 определение типа создаваемой хранимой процедуры: **временная** или **пользовательская**. Кроме этого, можно создать свою собственную системную хранимую процедуру, назначив ей имя с префиксом **sp_** и поместив ее в системную базу данных. Такая процедура будет доступна в контексте любой базы данных локального сервера;

 планирование прав доступа. При создании хранимой процедуры следует учитывать, что она будет иметь те же права доступа к объектам базы данных, что и создавший ее пользователь;

 определение параметров хранимой процедуры. Подобно процедурам, входящим в состав большинства языков программирования, хранимые процедуры могут обладать входными и выходными параметрами;

 разработка кода хранимой процедуры. Код процедуры может содержать последовательность любых команд SQL, включая вызов других хранимых процедур.

Создание новой и изменение имеющейся хранимой процедуры осуществляется с помощью следующей команды:

```
<определение_процедуры>::=
{CREATE | ALTER } [PROCEDURE] имя_процедуры
    [;номер]
[{@имя_параметратип_данных } [VARYING ]
    [=default][OUTPUT] ][,...n]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE,
    ENCRYPTION }]
[FOR REPLICATION]
AS
    sql_оператор [...n]
```

1. 8 Лекция №15, 16 (4 часа).

Тема: «Триггеры»

1.8.1 Вопросы лекции:

1.Триггеры: создание и применение.
2.Определение триггера, область его использования, место и роль триггера в обеспечении целостности данных.
3.Типы триггеров.
4.Программирование триггера.

1.8.2 Краткое содержание вопросов:

1.Триггеры: создание и применение.
2.Определение триггера, область его использования, место и роль триггера в обеспечении целостности данных.
3.Типы триггеров.

4.Программирование триггера.

Определение триггера в стандарте языка SQL

Триггеры являются одной из разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора языка манипулирования данными (DML). Триггеры используются для проверки целостности данных, а также для отката транзакций.

Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных. Применение триггеров большей частью весьма удобно для пользователей базы данных. И все же их использование часто связано с дополнительными затратами ресурсов на операции ввода/вывода. В том случае, когда тех же результатов (с гораздо меньшими непроизводительными затратами ресурсов) можно добиться с помощью хранимых процедур или прикладных программ, применение триггеров нецелесообразно.

Триггеры – особый инструмент SQL-сервера, используемый для поддержания целостности данных в базе данных. С помощью ограничений целостности, правил и значений по умолчанию не всегда можно добиться нужного уровня функциональности. Часто требуется реализовать сложные алгоритмы проверки данных, гарантирующие их достоверность и реальность. Кроме того, иногда необходимо отслеживать изменения значений таблицы, чтобы нужным образом изменить связанные данные.

Триггеры можно рассматривать как своего рода фильтры, вступающие в действие после выполнения всех операций в соответствии с правилами, стандартными значениями и т.д.

Триггер представляет собой специальный тип хранимых процедур, запускаемых сервером автоматически при попытке изменения данных в таблицах, с которыми триггеры связаны. Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером. Создает триггер только владелец базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

Триггер представляет собой весьма полезное и в то же время опасное средство. Так, при неправильной логике его работы можно легко уничтожить целую базу данных, поэтому триггеры необходимо очень тщательно отлаживать. В отличие от обычной подпрограммы, триггер выполняется неявно в каждом случае возникновения триггерного события, к тому же он не имеет аргументов. Приведение его в действие иногда называют запуском триггера. *С помощью триггеров достигаются следующие цели:*

 проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;
 выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом;
 накопление аудиторской информации посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили;
 поддержка репликации.

Основной формат команды CREATE TRIGGER показан ниже:

<Определение_триггера>::=

CREATE TRIGGER имя_триггера

BEFORE | AFTER <триггерное_событие>

ON <имя_таблицы>
[REFERENCING
<список_старых_или_новых_псевдонимов>]
[FOR EACH { ROW | STATEMENT}]
[WHEN(условие_триггера)]
<тело_триггера>

2. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ
ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

2.1 Практическое занятие №1, 2, 3, 4 (8 часов).

Тема: «Определение структурированного языка запросов SQL»
Разработка базы данных. Основы работы в MySQL

2.1.1 Задание для работы:

Часть 1. Нормализация данных

Продажи
06.09.2011 г.

Клиент	Товар	Количество	Цена	Сумма
Иванов	Хлеб	2	24,50 р.	49,00 р.
Петров	Молоко	3	30,00 р.	90,00 р.
ОАО «Рога и копыта»	Хвосты	25	2,00 р.	50,00 р.
ЗАО «111»	Молоко	1	30,00 р.	30,00 р.
Сидоров	Хлеб	3	24,50 р.	73,50 р.

I нормальная форма

Продажи

Дата	Клиент	Товар	Количество	Цена	Сумма
06.09.2011 г.	Иванов	Хлеб	2	24,50 р.	49,00 р.
06.09.2011 г.	Петров	Молоко	3	30,00 р.	90,00 р.
06.09.2011 г.	ОАО «Рога и копыта»	Хвосты	25	2,00 р.	50,00 р.
06.09.2011 г.	ЗАО «111»	Молоко	1	30,00 р.	30,00 р.
06.09.2011 г.	Сидоров	Хлеб	3	24,50 р.	73,50 р.

II нормальная форма

Клиенты

Код клиента	Клиент
1	Иванов
2	Петров
3	ОАО «Рога и копыта»
4	ЗАО «111»
5	Сидоров

Товары

Код товара	Товар	Цена
1	Хлеб	24,50 р.
2	Молоко	30,00 р.
3	Хвосты	2,00 р.

Продажи

Код клиента	Код товара	Количество	Сумма	Дата
1	1	2	24,50 р.	06.09.2011 г.
2	2	3	30,00 р.	06.09.2011 г.
3	3	25	2,00 р.	06.09.2011 г.
4	2	1	30,00 р.	06.09.2011 г.
5	1	3	24,50 р.	06.09.2011 г.

III нормальная форма

Клиенты

Код клиента	Клиент
1	Иванов
2	Петров
3	ОАО «Рога и копыта»
4	ЗАО «111»
5	Сидоров

Товары

Код товара	Товар	Цена
1	Хлеб	24,50 р.
2	Молоко	30,00 р.
3	Хвосты	2,00 р.

Продажи

Код клиента	Код товара	Количество	Дата
1	1	2	06.09.2011 г.
2	2	3	06.09.2011 г.
3	3	25	06.09.2011 г.
4	2	1	06.09.2011 г.
5	1	3	06.09.2011 г.

Типы данных

MySQL поддерживает несколько типов столбцов, которые можно разделить на три категории: числовые типы данных, типы данных для хранения даты и времени и символьные (строковые) типы данных. Мы кратко рассмотрим основные типы данных. Более подробно ознакомиться с типами данных можно в дополнительном материале.

В описаниях используются следующие обозначения:

- M - указывает максимальный размер вывода. Максимально допустимый размер вывода составляет 255 символов.
- D - употребляется для типов данных с плавающей точкой и указывает количество разрядов, следующих за десятичной точкой. Максимально возможная величина составляет 30 разрядов, но не может быть больше, чем M-2.

Квадратные скобки ('[' и ']') указывают для типа данных группы необязательных признаков.

Заметьте, что если для столбца указать параметр ZEROFILL, то MySQL будет автоматически добавлять в этот столбец атрибут UNSIGNED.

- **INT**[(M)] [UNSIGNED] [ZEROFILL]

Целое число нормального размера. Диапазон со знаком от -2147483648 до 2147483647. Диапазон без знака от 0 до4294967295.

- **FLOAT**[(M,D)] [UNSIGNED] [ZEROFILL]

Малое число с плавающей точкой обычной точности. Допустимые значения: от -3,402823466E+38 до -1,175494351E-38, 0, и от 1,175494351E-38 до 3,402823466E+38. Если указан атрибут UNSIGNED, отрицательные значения недопустимы. Атрибут M указывает количество выводимых пользователю знаков, а атрибут D - количество разрядов, следующих за десятичной точкой. Обозначение FLOAT без указания аргументов или запись вида FLOAT(X), где X <=24справедливы для числа с плавающей точкой обычной точности.

- **DATE**

Дата. Поддерживается интервал от '1000-01-01' до '9999-12-31'. MySQL выводит значения DATE в формате 'YYYY-MM-DD', но можно установить значения в столбец DATE, используя как строки, так и числа.

- [NATIONAL] **CHAR**(M) [BINARY]

Строка фиксированной длины, при хранении всегда дополняется пробелами в конце строки до заданного размера. Диапазон аргумента M составляет от 0 до 255 символов (от 1 до 255 в версиях, предшествующих MySQL 3.23). Концевые пробелы удаляются при выводе значения. Если не задан атрибут чувствительности к регистру BINARY, то величины CHAR сортируются и сравниваются как независимые от регистра в соответствии с установленным по умолчанию алфавитом.

Атрибут NATIONAL CHAR (или его эквивалентная краткая форма NCHAR) представляет собой принятый в ANSI SQL способ указания, что в столбце CHAR должен использоваться установленный по умолчанию набор символов (CHARACTER).

- [NATIONAL] **VARCHAR**(M) [BINARY]

Строка переменной длины. **Примечание:** концевые пробелы удаляются при сохранении значения (в этом заключается отличие от спецификации ANSI SQL). Диапазон аргумента M составляет от 0 до 255 символов (от 1 до 255 в версиях, предшествующих MySQLVersion 4.0.2). Если не задан атрибут чувствительности к регистру BINARY, то величины VARCHAR сортируются и сравниваются как независимые от регистра.

Выработка данных – оператор SELECT

2.2.1 Задание для работы:

Теоретический материал:

Содержимое в таблицах в SQL просматривается с помощью оператора *SELECT*. Синтаксис его использования следующий:

```
SELECT <поля> FROM <таблица>
```

Вместо <поля> необходимо указать либо имя столбца, значения которого нужно просмотреть, либо имена нескольких столбцов через запятую, либо символ звездочки «*», означающий выбор всех столбцов таблицы.

Вместо <таблица>следует указать имя таблицы.

Пример 1. Просмотреть все столбцы из таблицы RODITELI.

```
SELECT * FROM RODITELI
```

Пример 2. Вывести фамилии родителей.

```
SELECT FIO_ROD FROM RODITELI
```

Пример 3. Вывести фамилии родителей, телефоны и место работы.

```
SELECT FIO_ROD, TEL, RABOTA FROM RODITELI
```

Для вывода полей из разных таблиц используются составные имена в виде Имя_таблицы.Имя_поля.

Пример 4. Вывести названия районов и городов.

Можно вывести данные двумя запросами:

```
SELECT NAZVANIE  FROM  REGION
SELECT NAZVANIE  FROM  GOROD
```

Либоиспользоватьсоставныеимена:

SELECT REGION.NAZVANIE, GOROD.NAZVANIE FROM REGION, GOROD

Для переименования выводимого поля используется конструкция **ASнов_имя_поля**, которая называется псевдонимом.

Пример 5. Вывести фамилии родителей, переименовав поле FIO_ROD в ФАМИЛИЯ.

SELECT FIO_ROD AS Фамилия FROM RODITELI

Для исключения повторяющихся записей используется функция **DISTINCT** (отличающиеся), для вывода всех записей используется функция **ALL** (все). Функция ALL используется по умолчанию.

Пример 6. Вывести должностей родителей.

SELECT DISTINCT (RABOTA) FROM RODITELI

Для вывода заданного количество строк и указания позиции используется команда **LIMITномер_позиции, количество_строк**.

Пример 7. Вывести с 2 по 4 строки таблицы DANNIE.

SELECT * FROM RODITELI LIMIT 1,3

Задания:

1) Вывести данные из таблицы DANNIE.
2) Вывести данные из таблицы DISCHIPLINA.
3) Вывести фамилии всех студентов.
4) Вывести названия всех групп.
5) Вывести фамилии, имена, телефоны, паспортные данные студентов.
6) Вывести фамилии родителей и телефоны.
7) Вывести названия городов, названия улиц.
8) Вывести названия предметов и фамилии преподавателей.
9) Вывести фамилии и дату рождения студентов, переименовав поле DATE_ROGNEN в ДЕНЬ_РОЖДЕНИЯ.
10) Вывести названия улиц, переименовав поле NAZVANIE в УЛИЦЫ.
11) Вывести список улиц, исключив повторяющиеся значения.
12) Вывести различные имена студентов.
13) Вывести первую в списке специальность.
14) Вывести с 6 по 10 строки таблицы RODITELI.

2.2 Практическое занятие №5, 6, 7, 8 (8 часов).

Тема: «Эффективное выполнение запросов для извлечения данных»
Уточнения запросов

2.3.1 Задание для работы:

Теоретический материал:

Использование в операторе SELECT предложения, определяемого ключевым словом WHERE (где), позволяет задавать выражение условия (предикат), принимающее значение истина или ложь для значений полей строк таблиц, к которым обращается оператор SELECT. Предложение WHERE определяет, какие строки указанных таблиц должны быть выбраны. В таблицу, являющуюся результатом запроса, включаются только те строки, для

которых условие (предикат), указанное в предложении WHERE, принимает значение истина.

В задаваемых в предложении WHERE условиях могут использоваться операции сравнения, определяемые операторами = (равно), > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), <> (не равно), а также логические операторы AND, OR и NOT.

Пример 1. Выбратьстудентов 3 группы.

SELECT * FROM DANNIE WHERE KOD_GRUPPY=3

Пример 2. Вывести родителей, работающих водителями и поварами.

SELECT FIO_ROD FROM RODITELI WHERE RABOTA='ВОДИТЕЛЬ' OR RABOTA='ПОВАР'

Оператор **GROUPBY** (группировать по) служит для группировки записей по значениям одного или нескольких столбцов. Он собирает записи в группы и упорядочивает группы по алфавиту (точнее по ASCII- кодам символов).

Пример 3. Вывести родителей студентов сгруппированных по месту работы.

SELECT RABOTA, FIO_ROD FROM RODITELI GROUP BY RABOTA

Оператор **HAVING** (имеющие, при условии) обычно применяются совместно с оператором группировки GROUPBY и задает фильтр записей в группах

Пример 4. Вывести родителей студентов сгруппированных по месту работы, если они водители или учителя.

SELECT RABOTA, FIO_ROD FROM RODITELI GROUP BY RABOTA HAVING (RABOTA='ВОДИТЕЛЬ') OR (RABOTA='УЧИТЕЛЬ')

При задании логического условия в предложении WHERE могут быть использованы операторы IN, BETWEEN, LIKE, IS NULL.

Операторы **IN** (равен любому из списка) и **NOT IN** (не равен ни одному из списка) используются для сравнения проверяемого значения поля с заданным списком. Этот список значений указывается в скобках справа от оператора IN.

Построенный с использованием IN предикат (условие) считается истинным, если значение поля, имя которого указано слева от IN, *совпадает* (подразумевается точное совпадение) с одним из значений, перечисленных в списке, указанном в скобках справа от IN.

Предикат, построенный с использованием NOT IN, считается истинным, если значение поля, имя которого указано слева от NOT IN, *не совпадает* ни с одним из значений, перечисленных в списке, указанном в скобках справа от NOT IN.

Пример 5. Вывести родителей, работающих водителями и поварами.

SELECT FIO_ROD FROM RODITELI WHERE RABOTA IN ('ВОДИТЕЛЬ', 'ПОВАР')

Пример 6. Вывести всех родителей, кроме работающих учителями и врачами.

SELECT FIO_ROD FROM RODITELI WHERE RABOTA NOT IN ('УЧИТЕЛЬ', 'ВРАЧ')

Задания:

1. Вывести фамилии студентов, обучающихся в группе с кодом 3.

2. Вывести название региона с кодом 1.
3. Вывести фамилию преподавателя с кодом 2.
4. Вывести информацию о студентах, обучающихся в группах с кодами 1 и 2.
5. Вывести названия дисциплин с кодами 2 и 3.
6. Вывести имена студентов, заглавные буквы которых находятся в диапазоне от «В» до «М».
7. Вывести данные о студентах, фамилии которых начинаются на букву «М».
8. Выбрать записи, содержащие пустые значения о номере квартире студентов (проживающих в доме).
9. Отсортировать фамилии студентов в алфавитном порядке.
10. Вывести данные о студентах, отсортировав номера телефонов по возрастанию.
11. Отсортировать студентов по номеру группы в порядке возрастания, а для одинаковых групп - фамилии в порядке, обратном алфавитному.
12. Выбрать студентов, фамилии которых заканчиваются на «а».
13. Отсортировать в порядке возрастания фамилии студентов, чьи отчества заканчиваются на «ич».
14. Отсортировать в порядке возрастания фамилии студентов, обучающихся в группе 3.
15. Вывести студентов, родившихся в 1990 году.
16. Отсортировать в алфавитном порядке фамилии студентов, у которых начальные буквы имени заключены в диапазоне от «И» до «Я».
17. Вывести данные о студентах с фамилиями «Петров», «Смелов».
18. Вывести в порядке, обратном алфавитному, все имена и фамилии студентов, родившихся не в 1991 году.
19. Вывести информацию о всех студентах, кроме Варечкина и Климовой.
20. Выбрать студентов, у которых номера телефонов связи МТС и начинаются с 3.

Агрегированные функции

2.4.1 Задание для работы:

Агрегированные функции используются подобно именам полей в предложении SELECT запроса, но с учетом того, что они берут имена полей в качестве аргумента. С SUM и AVG используются только числовые поля, а с COUNT, MAX, MIN могут использоваться числовые или символьные поля.

Функция **COUNT** производит подсчет количества строк или не-NULL значений полей, которые выбрал запрос.

Пример 1. Подсчитать количество записей в таблице DANNIE.

SELECTCOUNT(*) FROMDANNIE

В результате выполнения этого запроса появится столбец с заголовком COUNT(*), поэтому можно использовать оператор переименования.

Пример 2. Подсчитать количество записей в таблице DANNIE и назвать поле КОЛИЧЕСТВО.

SELECT COUNT(*) AS КОЛИЧЕСТВО FROM DANNIE

Функция **SUM** рассчитывает арифметическую сумму всех выбранных значений данного поля.

Пример 3. Вывести сумму оценок студентов сгруппированных по номеру группы.

SELECT KOD_GRUPPY, SUM(OCENKA) FROM DANNIE, USPEV GROUP BY KOD_GRUPPY

Функция**AVG** – производит усреднение всех выбранных значений данного поля.

Пример 4. Вывести среднее значения оценок.

SELECT AVG(OCENKA) FROM USPEV

Функция**MAX** – находит и возвращает наибольшее из всех выбранных значений данного поля.

Пример 5. Вывести максимальную оценку.

SELECTMAX(OCENKA) FROMUSPEV

Функция**MIN** – находит и возвращает наименьшее из всех выбранных значений данного поля.

Пример 6. Вывести минимальную оценку студентов.

SELECT MIN(OCENKA) FROM USPEV

Ключевое слово**GROUPBY** – указывает условие группировки строк.

Пример 7. Вывести среднее оценок, максимальную оценку, минимальную оценку студентов сгруппированных по номеру группы, с указанием имени у каждого столбца.

SELECT KOD_STUDENT, AVG(OCENKA) AS СРЕДНЯЯ, MAX(OCENKA) AS МАКСИМАЛЬНАЯ, MIN (OCENKA) AS МИНИМАЛЬНАЯ FROM USPEV GROUP BY KOD_STUDENT

Пример 8. Вывести коды и численность групп, в которых более 2 человек.

SELECT KOD_GRUPPY, COUNT(*) FROM DANNIE GROUP BY KOD_GRUPPY HAVING COUNT(*)>2

Задания:

Внимание: необходимо переименовать каждое вычисляемое поле.

1. Найти среднее значение оценок по каждому студенту.
2. Найти максимальную оценку по каждой дисциплине.
3. Найти среднюю оценку, выставленную каждым преподавателем.
4. Вывести минимальную оценку, выставленную каждым преподавателем.
5. Перевести каждую оценку в рейтинговый бал (оценка, большая 3 баллов, увеличивается в 2 раза).
6. Подсчитать количество разных групп.
7. Подсчитать количество различных квартир.
8. Вывести среднюю оценку, максимальную оценку, минимальную оценку для студента с кодом 3.
9. Подсчитать количество хороших оценок.
10. Подсчитать процент двоек, выставленных каждым преподавателем.
11. Посчитать количество и сумму 5-к и 4-к.
12. Подсчитать процент качества и процент успеваемости (общее количество оценок 26).

13. На скольких улицах проживают более 1 студента.

14. Вывести количество оценок, для которых выполняется условие «оценка*2+1>10».

2.3 Практическое занятие №9, 10, 11, 12 (8 часов).

Тема: «Построение нетривиальных запросов»
Строковые и числовые функции

2.5.1 Задание для работы:

Функция **UCASE** преобразует символы в верхний регистр.

Пример 1. Вывести фамилии студентов заглавными буквами, переименовав поле fam в familio.

SELECT UCASE(FAM) AS FAMILIO FROM DANNIE

Функция **UPPER** – переводит все символы указанной в параметре строки в верхний регистр (работает только с латиницей).

Функция **LCASE** преобразует символы в нижний регистр.

Пример 2. Вывести фамилии студентов строчными буквами, переименовав поле fam в familio.

SELECT LCASE(FAM) AS FAMILIO FROM DANNIE

Функция **LOWER** – переводит все символы указанной в параметре строки в нижний регистр (работает только с латиницей).

Функция **CONCAT(str1,str2...)** возвращает строку, созданную путем объединения аргументов (аргументы указываются в скобках - str1,str2...)., аргументами являются имена полей.

Пример 3. Вывести фамилию и имя студента в одном поле.

SELECT CONCAT(FAM, IMA) FROM DANNIE

Результатом будет строка, состоящая из фамилии и имени, не разделенных пробелом. Для добавления пробела запрос нужно изменить:

SELECTconcat(fam,' ', ima) FROM `dannie`

Функция**INSERT(str, pos, len, new_str)**возвращает строку str, в которой подстрока, начинающаяся с позиции pos и имеющая длину len символов, заменена подстрокой new_str.

Пример 4. Вывести фамилии студентов с 3 символа (вставить с 1 позиции 3 пробела).

SELECT INSERT(FAM, 1, 3, ' ') FROM DANNIE

Функция**LENGTH(str)** возвращает длину строки str.

Пример 5. Вывести фамилия студента и количество символов в ней.

SELECT FAM, LENGTH(FAM) FROM DANNIE

Функция**REPEAT(str, n)** возвращает строку str n-количество раз.

Пример 6. Вывести фамилию студента 3 раза в одном поле.
SELECT REPEAT(`fam`,3) FROM `dannie`

Функция**REPLACE(str, pod_str1, pod_str2)**возвращаетстрокуstr, вкоторойвсеподстроки pod_str1 замененыподстроками pod_str2.

Пример 7. В названиях городов заменить длинное 'Армавир' на короткое 'Ар'.
SELECT REPLACE(NAZVANIE,'АРМАВИР','АР') FROM GOROD

Функция**REVERSE(str)** возвращает строку str, записанную в обратном порядке.

Пример 8. Написать фамилии студентов в обратном порядке.
SELECT REVERSE(FAM) FROM DANNIE

Задания:

1.Вывести фамилии родителей заглавными буквами.

2.Вывести название улиц маленькими буквами.

3.Вывести названия факультетов, курс, группу (вырезать из полного названия группы название факультета, например, из МФ-МАТ-4-1 должно получиться МФ-4-1).

4.Вывести данные о родителях, разместив информацию о работе и телефоне в одном поле.

5.Вывести имя, отчество, телефон студента и количество символов в них.

6.Вывести номер телефона в обратном порядке.

7.Вывести дату рождения 5 раз в одном поле.

8.Заменить 1991 год рождения на 91.

Условные выражения с оператором CASE

2.6.1 Задание для работы:

В обычных языках программирования имеются операторы условного перехода, которые позволяют управлять вычислительным процессом в зависимости от того, выполняется или нет некоторое условие. В языке SQL таким оператором является CASE (случай, обстоятельство, экземпляр). Он имеет две основные формы.

Оператор CASE со значением имеет следующий синтаксис:

```
CASEпроверяемое_значение
      WHENзначение1THENрезультат1
      WHENзначение2THENрезультат2
      ….
      WHENзначениеNTHENрезультатN
ELSEрезультатX
END
```

В случае, когда *проверяемое_значение*равно *значение1*, оператор CASE возвращает значение *результат1*. В противном случае *проверяемое_значение* сравнивается с *значение2*, и если они равны, возвращается *результат2*и.т.д. Если *проверяемое_значение*не равно ни одному из таких значений, то возвращается значение *результатX*.

Ключевое слово ELSE не является обязательным. Если оно отсутствует и ни одно из значений, подлежащих сравнению, не равно проверяемому значению, то возвращается значение NULL.

Пример 1. Вывести название региона и код региона (Краснодарский край – 93, Ставропольский край – 73)

```
SELECT `nazvanie` ,
CASE `nazvanie`
WHEN 'Краснодарский край' THEN '93'
WHEN 'Ставропольский край' THEN '73'
ELSE `nazvanie`
END
AS Код_региона
FROM `region`
```

Вторая форма оператора CASE предполагает его использование при поиске в таблице тех записей, которые удовлетворяют определенному условию:

```
CASE
      WHEN условие1 THEN результат1
      WHEN условие2 THEN результат2
      ….
      WHEN условиеN THEN результатN
ELSE результатX
END
```

Оператор CASE проверяет, истинно ли *условие1* для первой записи в наборе, определенном оператором WHERE, или во всей таблице, если оператор WHERE отсутствует. Если да, то CASE возвращает значение *результат1*. В противном случае для данной записи проверяется *условие2* и т.д. Если ни одно из условий не выполняется, то возвращается значение *результатX*, указанное после ключевого слова ELSE.

Ключевое слово ELSE не является обязательным. Если оно отсутствует и ни одно из значений, подлежащих сравнению, не равно проверяемому значению, то возвращается значение NULL.

Порядок выполнения работы:

1. Перевести каждую оценку в рейтинговый бал (за оценку меньше 3 начисляется 0 баллов, от 3 до 4 – 1 балл, за оценку 5 – 2 балла).
2. Вывести список оценок и их буквенное обозначение (5 – «отлично», 4 – «хорошо», 3 – «удовлетворительно», 2 – «неудовлетворительно»).
3. Вывести список оценок и указать значение по системе «зачет-незачет» (для оценок 5 или 4 – «зачет», для остальных – «незачет»).
4. Вывести названия групп и названия специальностей («…ПИЭ…» - Прикладная информатика в экономике, «…Мат…» - Математика, «…Инф…» - Информатика, в случае другого обозначения повторить название группы).
5. Вывести фамилии студентов и место прохождения практики (студенты группы с кодом 1 проходят практику в «Банк УралСиб», с кодом 2 – «СберБанк», с кодом 3 – «Первомайский», с кодом 4 – «РосСельхозБанк»).

2.4 Практическое занятие №13, 14, 15, 16 (8 часов).

Тема: «Запросы модификации данных в реляционной таблице»
Разработка базы данных. Основы работы в MySQL

2.7.1 Задание для работы:

Подзапрос – это запрос на выборку данных, вложенный в другой запрос.

SELECT<*поля*>FROM<*таблица*>WHERE (HAVING) УСЛОВИЕ (SELECT<*поля*>FROM<*таблица*>WHERE<*условие*>)

В свою очередь подзапрос может содержать другой подзапрос, но в первую очередь выполняется подзапрос, имеющий самый глубокий уровень вложения.

Часто, но не всегда, внешний запрос обращается к одной таблице, а подзапрос - к другой.

Простые подзапросы характеризуются тем, что они формально никак не связаны с содержащими их внешними запросами, что позволяет сначала выполнить подзапрос, результат которого используется для выполнения внешнего запроса.

Три вида простых подзапросов:

- подзапросы, возвращающие единственное значение;
- подзапросы, возвращающие список значений, из одного столбца таблицы;
- подзапросы, возвращающие набор записей.

Подзапросы, возвращающие единственное значение

Пример 1. Вывести оценки, которые больше среднего значения всех оценок

SELECT * FROM USPEV WHERE OCENKA > (SELECT AVG(OCENKA) FROM USPEV)

Подзапросы, возвращающие список значений, из одного столбца таблицы

Пример 2. Определить коды родителей студента Маркова и Иванова.

SELECT KOD_RODITEL FROM RODDETI WHERE KOD_STUDENT IN (SELECT KOD_STUDENT FROM DANNIE WHERE FAM='МАРКОВ' OR FAM='ИВАНОВ')

Пример 3. Определить название дисциплин, которые сдавал студент с кодом 2.

SELECT NAZVANIE FROM DISCHIPLINA WHERE KOD_DISCHIPLINA IN (SELECT KOD_DISCHIPLINA FROM USPEV WHERE KOD_STUDENT='2')

Пример 4. Вывести фамилию, и место работы родителей студента с кодом 3.

SELECT FIO_ROD, RABOTA FROM RODITELI WHERE KOD_RODITEL IN (SELECT KOD_RODITEL FROM RODDETI WHERE KOD_STUDENT=3)

Порядок выполнения работы:

1. Вывести список оценок, которые получил студент Воркин.
2. Вывести все города Краснодарского края.
3. Вывести название группы студента Маркова.
4. Вывести название улицы, на которой живет студент Варечкин.
5. Определить фамилию преподавателя, поставившему студенту Климову оценку по программированию.

2.8.1 Задание для работы:

Пример 1. Вывести название дисциплин, по которым получена оценка 5.

SELECT NAZVANIE FROM DISCHIPLINA WHERE 5 IN (SELECT OCENKA FROM USPEV WHERE KOD_DISCHIPLINA= DISCHIPLINA.KOD_DISCHIPLINA)

Такой подзапрос отличается от простого подзапроса тем, что вложенный подзапрос не может быть обработан прежде, чем будет обрабатываться внешний подзапрос. Это связано с тем, что вложенный подзапрос зависит от значения DISCHIPLINA.KOD_DISCHIPLINA, а оно изменяется по мере того, как система проверяет различные строки таблицы DISCHIPLINA. Следовательно, с концептуальной точки зрения обработка осуществляется следующим образом:

1. Система проверяет первую строку таблицы DISCHIPLINA. Предположим, что это строка дисциплины с номером 1. Тогда значение DISCHIPLINA.KOD_DISCHIPLINA будет в данный момент имеет значение, равное 1, и система обрабатывает внутренний запрос:

(SELECT OCENKA FROM USPEV WHERE KOD_DISCHIPLINA= 1)

получая в результате множество (1, 4, 3, 5, 2, 4, 3, 3, 5). Теперь система может завершить обработку для дисциплины с номером 1. Выборка значения NAZVANIE для KOD_DISCHIPLINA= 1 (База данных) будет проведена тогда и только тогда, когда OCENKA=5 будет принадлежать этому множеству, что, очевидно, справедливо.

2. Далее система будет повторять обработку такого рода для следующей дисциплины и т.д. до тех пор, пока не будут рассмотрены все строки таблицы DISCHIPLINA.

Подобные подзапросы называются *связанными / соотнесенными / коррелированными*, так как их результат зависит от значений, определенных во внешнем подзапросе. Обработка связанного подзапроса, следовательно, должна повторяться для каждого значения извлекаемого из внешнего подзапроса, а не выполняться раз и навсегда.

Порядок выполнения работы:

1. Вывести фамилии преподавателей, которые поставили хотя бы одну двойку.
2. Вывести название предметов, средняя оценка по которым выше 3.
3. Вывести фамилии студентов, у которых имеются оценки 3 и 4 (одновременно).
4. Вывести фамилии студентов, которые получили хотя бы одну оценку, выше средней.
5. Вывести названия групп, в которых обучается 6 студентов.

2.5 Практическое занятие №17, 18, 19, 20 (8 часов).

Тема: «Понятие представлений»
Операции соединения

2.9.1 Задание для работы:

Декартовым произведением двух таблиц называется таблица, составленная из всевозможных сочетаний записей обеих таблиц.

Пример 1. Вывести декартово произведение таблиц DANNIE и USPEV.

SELECT * FROMDANNIE, USPEV

Общим столбцом этих таблиц является столбец kod_student.

В полученном декартовом произведении интересуют не все данные, а только те, в которых идентичные столбцы имеют одинаковые значения. Кроме того, в результативной таблице не нужны два идентичных столбца, достаточно лишь одного из них.

SELECT DANNIE.*, USPEV.OCENKA FROM DANNIE, USPEV WHERE DANNIE.KOD_STUDENT=USPEV.KOD_STUDENT

В результате выполнения этого запроса получается таблица естественным соединением.

Данный запрос можно записать, используя псевдонимы.

SELECT T1.*, T2.OCENKA FROM DANNIE T1, USPEV T2 WHERE T1.KOD_STUDENT=T2.KOD_STUDENT

Эквивалентный запрос можно составить с помощью оператора **NATURALJOIN**

SELECT T1.*, T2.OCENKA FROM DANNIE T1 NATURAL JOIN USPEV T2.

При естественном соединении, выполняемом с помощью NATURALJOIN, проверяется равенство всех одноимённых столбцов соединяемых таблиц.

Условное соединение (JOIN …ON)

Условное соединение похоже на соединение с условием равенства. В качестве условия может выступать любое логическое выражение, которое записывается после ключевого слова ON (при), а не WHERE. Если условие выполняется для текущей записи декартового произведения, то она входит в результативную таблицу.

Пример 2. Вывести информацию о студентах и их оценках по дисциплине с кодом 2.

SELECT * FROM DANNIE JOIN USPEV ON (DANNIE.KOD_STUDENT = USPEV.KOD_STUDENT) AND (USPEV.KOD_DISCHIPLINA=2)

Соединение по именам столбцов (JOIN …USING)

Соединение по именам столбцов похоже на естественное соединение. Отличие состоит в том, что можно указать, какие именно одноименные столбцы должны проверяться, а при естественном проверяются все одноименные столбцы.

Пример 3. Вывести в каком городе какие улицы.

SELECT * FROM GOROD JOIN ULICA USING (KOD_GOROD)

Данные таблицы содержат более одного идентичного поля, поэтому естественное соединение вернет пустой результат.

Пример 4. Запрос можно сформулировать иначе:

SELECT * FROM GOROD INNER JOIN ULICA ON (GOROD.KOD_GOROD= ULICA. KOD_GOROD)

Внешние соединения таблиц

Из таблицы, получаемой при внутреннем соединении, отбраковываются все записи, для которых нет соответствующей записи одновременно в обеих таблицах. При внешнем соединении такие несоответствующие записи сохраняются.

Левое соединение (LEFTOUTERJOIN)

При левом внешнем соединении несоответствующие записи, имеющиеся в левой таблице, сохраняются в результативной, а имеющиеся в правой – удаляются.

Рассмотренный в ***Примере 1*** запрос выводит не все записи, тот же самый запрос при внешнем левом объединении выглядит так:

SELECT T1.*, T2.OCENKA FROM DANNIE T1 LEFT OUTER JOIN USPEV T2 ON T1.KOD_STUDENT=T2.KOD_STUDENT

Правое соединение (RIGHTOUTERJOIN)

При правом внешнем соединении несоответствующие записи, имеющиеся в правой таблице, сохраняются в результативной, а имеющиеся в левой – удаляются.

Рассмотренный в ***Примере 1*** запрос выводит не все записи, тот же самый запрос при внешнем правом объединении выглядит так:

SELECT T1.*, T2.OCENKA FROM DANNIE T1 RIGHT OUTER JOIN USPEV T2 ON T1.KOD_STUDENT=T2.KOD_STUDENT

Полное соединение (FULLJOIN)

Полное соединение выполняет и левое, и правое внешние соединения.

SELECT * FROM DANNIE FULL JOIN USPEV

Задания:

1. Вывести названия регионов и соответствующие названия городов.
2. Вывести перечень специальностей и название групп.
3. Вывести названия дисциплин, по которым студенты получили 5.
4. Вывести фамилии преподавателей, поставивших 3.
5. Вывести фамилии студентов и названия соответствующих улиц.
6. Вывести фамилии студентов и названия соответствующих городов.
7. Вывести фамилии студентов и названия соответствующих регионов.
8. Вывести информацию о студентах и их родителях.

Добавление, удаление и изменение данных

2.10.1 Задание для работы:

Для добавления записи в таблицу служит оператор **INSERT**, который имеет несколько форм:

INSERTINTO<*таблица*>VALUES (<*список значений*>)

– вставляет пустую запись в указанную таблицу и заполняет эту запись значениями из списка, указанного за ключевым словом VALUES. При этом первое в списке значение вводится в первый столбец, второе – во второй и т.д. Порядок столбцов задается при создании таблицы. Данная форма не очень надежна, поскольку нетрудно ошибиться в порядке вводимых значений.

INSERTINTO<*таблица*> (<*список столбцов*>) VALUES (<*список значений*>)

– вставляет пустую запись в указанную таблицу и в заданные столбцы значения из указанного списка. При этом в первый столбец из *список столбцов* вводится первое значение из *список значений.* Порядок и количество имен столбцов в списке может отличаться от их порядка и количества, заданного при создании таблицы. Столбцы которые не указаны в списке, заполняются значением NULL.

Пример 1. Добавить в DANNIE Иванова Ивана Ивановича.

INSERT INTO DANNIE (FAM, IMA, OTCH) VALUES ('ИВАНОВ', 'ИВАН', 'ИВАНОВИЧ')

Возможна работа со значениями типа запись. Это позволяет за ключевым словом VALUES указывать несколько наборов значений в круглых скобках.

Пример 2. Добавить в таблицу Города записи НОВОКУБАНСК, КУРГАНИНСК, КРОПОТКИН.

INSERT INTO GOROD (NAZVANIE) VALUES ('НОВОКУБАНСК'), ('КУРГАНИНСК'), ('КРОПОТКИН')

INSERTINTO<*таблица*> (<*список столбцов*>) <*запрос на выборку*>

– вставляет в указанную таблицу записи, возвращаемые запросом на выборку.

Пример 3. Названия городов добавить в названия регионов.

INSERT INTO REGION(NAZVANIE) SELECT NAZVANIE FROM GOROD

Удаление записей

Для удаления записей из таблицы применяют запрос:

DELETE FROM <*таблица*> WHERE <*условие*>

Данный оператор удаляет из указанной таблицы записи (а не отдельные значения полей), которые удовлетворяют указанному условию.

В операторе WHERE может находиться подзапрос на выборку данных.

Операция удаления записей является необратимой, чтобы избежать удаления нужных данных рекомендуется сначала выполнять запросы, чтобы просмотреть какие записи будут удалены.

Порядок выполнения работы:

1. Добавить в таблицу о студентах одного студента.
2. Добавить в таблицу о родителях информацию о двух родителях студента.
3. Названия улиц добавить в названия городов.
4. Добавить названия дисциплин в названия специальностей.
5. Удалить из таблиц Города, названия улиц.
6. Удалить из таблицы Специальности названия дисциплин.
7. Удалить информацию о добавленном студенте.
8. Удалить информацию о родителях добавленного студента.
9. Изменить номера телефонов МТС на Мегафон (8918… на 8928…).
10. Изменить название дисциплины ‘Информатика’ на ‘Информатика и ИТ’.
11. Воркин Фома Григорьевич перевелся в группу с кодом 2. Внесите соответствующие изменения в таблицу DANNIE.

2.6 Практическое занятие №21, 22, 23, 24 (8 часов).

Тема: «Определение функций пользователя, примеры их создания и использования» Представления

2.11.1 Задание для работы:

Данные в представлениях выбираются из таблиц, т.е. представляются в том или ином виде. Они применяются, чтобы скрыть от пользователя некоторые столбцы, скомбинировать из нескольких таблиц одну, которая часто нужна пользователю, а запрос для неё очень сложен. Таким образом, представления используются как надстроечные средства для адаптации базы данных к различным категориям пользователей.

Представления создаются с помощью оператора ***CREATEVIEW*** (создать вид, представление).

CREATEVIEW<*имя представления*>AS<*запрос*>

Пример 1. Создать представление, которое выводит фамилии и соответствующие оценки студентов.

CREATE VIEW OCENKI AS

SELECTDANNIE.FAM, USPEV.OCENKA FROM DANNIE, USPEV WHERE DANNIE.KOD_STUDENT = USPEV. KOD_STUDENT.

Создана виртуальная таблица *OCENKI,* к которой можно обращаться с запросами как к обычной таблице.

Пример 2. Вывести из представления *OCENKI*, только фамилии и хорошие оценки.

*SELECT * FROM OCENKI WHERE OCENKA IN (4,5)*

Рассмотренное представление является многотабличным, поскольку создано на основе не одной, а двух таблиц. На практике используются более простые однотабличные представления, в которых скрываются некоторые столбцы и/или добавляются, значения которых вычисляются.

Пример 3. Создать представление Rod, в котором будут отображены фамилии родителей и их телефоны.

CREATE VIEW ROD AS SELECT FIO_ROD AS FIO, TEL FROM RODITELI.

Название представлений и таблиц не должны совпадать. Данное представление можно заменить обычным запросом

SELECT FIO_ROD AS FIO, TEL FROM RODITELI

Однако относительно полученного набора данных нельзя задать какой–нибудь запрос, поскольку этот набор является виртуальной таблицей, отличной от представления.

Оператор *CREATEVIEW*допускает и такую форму синтаксиса:

CREATEVIEW<*имя представления*> (<*столбец1*>, <*столбец2*>,…, <*столбецN*>) AS <*запрос*>

Пример 4. Создать представление, которое выводит фамилию и дату рождения студентов.

CREATE VIEW DATE (FIO, DATE) AS SELECT FAM, DATE_ROGNEN FROM DANNIE

В представлении указываются имена столбцов, которые могут быть отличны от имён столбцов в таблице.

Удаление представления осуществляется командой:

DROPVIEW<*имя представления*>

Порядок выполнения работы:

1. Создать представление DAN, которое выводит фамилию, паспортные данные студента, название улицы, на которой проживает студент.
2. В представлении DAN вывести отсортировать данные о студентах по улицам в алфавитном порядке.
3. В представлении DAN вывести студентов, проживающих на улицах, начинающих на букву К.
4. Создать представление MINMAX, которое выводит фамилию студента, минимальную, максимальную оценку.
5. В представлении MINMAX найти среднее минимальное и максимальное значение оценок.

Связь MYSQL и DELPHI

2.12.1 Задание для работы:

В первую очередь, создайте тестовую базу данных, к которой необходимо будет подключиться. Пусть база называется primer и содержит одну таблицу zarp с полями fio (varchar(20)) и zarp (int). Заполните ее данными, причем записи вносите как на русском, так и на латинском языке:

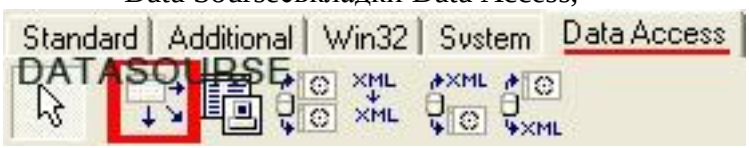
fio	zarp
Иванов	1000
Petrov	2000

Запустите Delphi.

На форме разместите компоненты:

- Button с вкладки Standart,

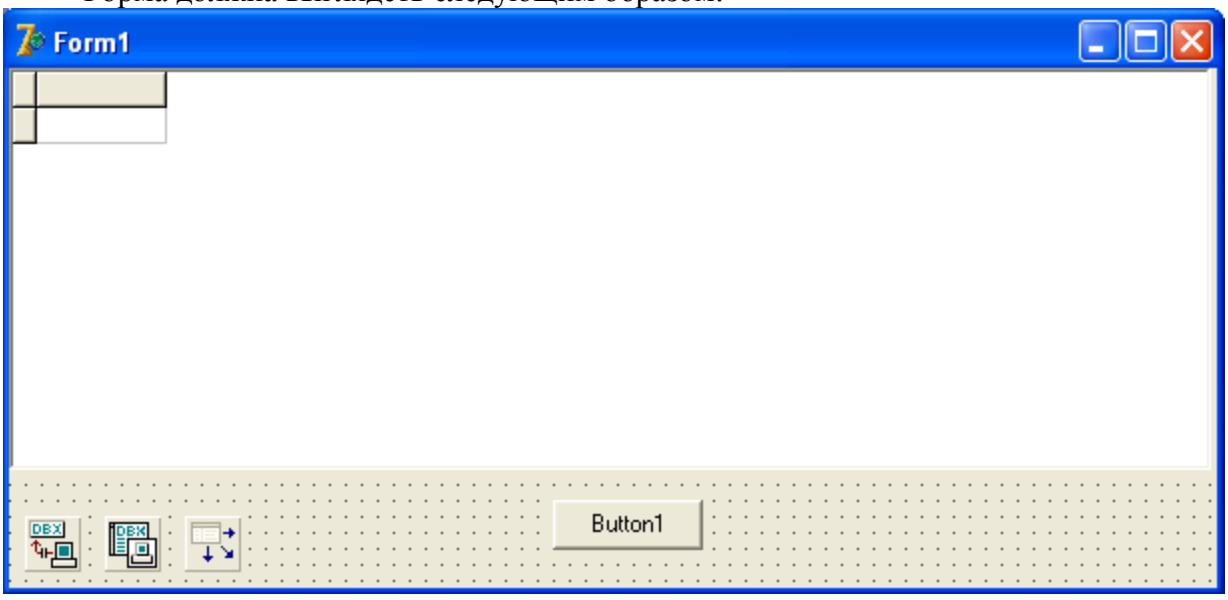
- Data Soursecвкладки Data Access,



- DBGridcвкладки Data Controls,

- SQLConnection, SimpleDataSetсвкладкиdbExpress.

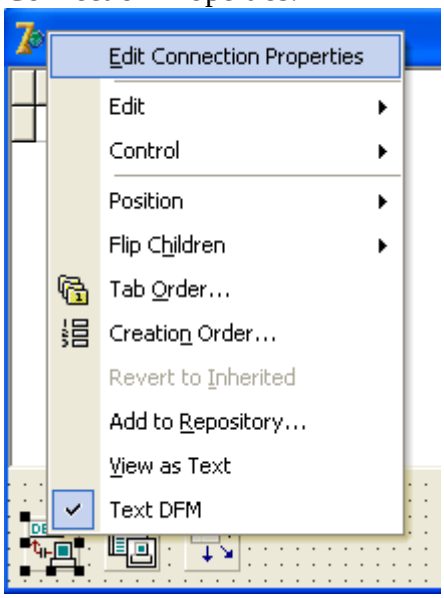
Форма должна выглядеть следующим образом:



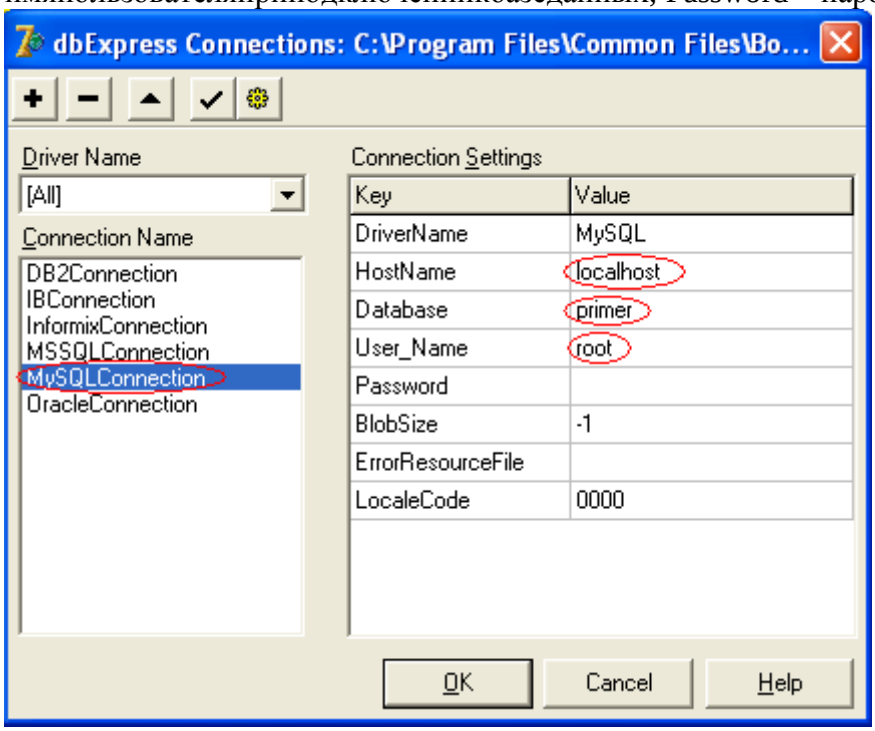
Сохраните проект в папку командой File->SaveProjectAs. В эту же папку поместите dll-библиотеку libmysql.dll.

Произведем настройку компонентов.

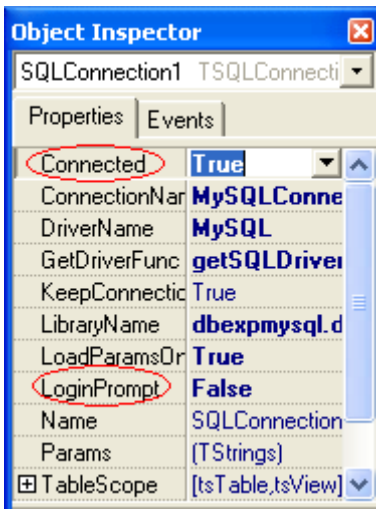
Выделите компонент SQLConnection. В контекстном меню выберите пункт Edit Connection Properties:



Выберитевполе Connection Name типMySQLConnection. Внастройкахподключения (Connection Settings) установитеследующиесвойства: HostName – адрессервера (localhost длялокальногоподключения), Database – имябазыданных, User_Name – имяпользователяприподключениикбазеданных, Password – пароль. Нажмите OK.



Проверьте, что выделен по-прежнему компонент SQLConnection, установите в инспекторе объектов (ObjectInspector) свойство LoginPrompt в false (это позволит отключить запрос пароля при каждом подключении к базе), а также Connected в true.



Теперь настроим компонент SimpleDataSet. В ObjectInspector для свойства Connection выберите из выпадающего списка значение SQLConnection1.

Далее раскройте свойство DataSet и в строке CommandText запишите запрос к базе данных. Например, Select * fromzarp.

2.7 Практическое занятие №25, 26, 27, 28 (8 часов).

Тема: «Хранимые процедуры»
Работа с SQLServer из VisuialStudio.NET

2.13.1 Задание для работы:

Взаимодействие с БД будем строить на основе технологии **ADO.NET**.

Запустите **Visual Studio.NET**.

Чтобы создать новый проект выберите **File**Æ**Newproject** или щелкните кнопкой мыши на **Newproject**.

Выберите **VisualBasicWindowsApplication**

Откроется окно, фрагмент которого показан на рисунке

*Если на экране нет окна **Обозревательсерверов**, найдите его во **View**Æ**Обозревательсерверов**.*

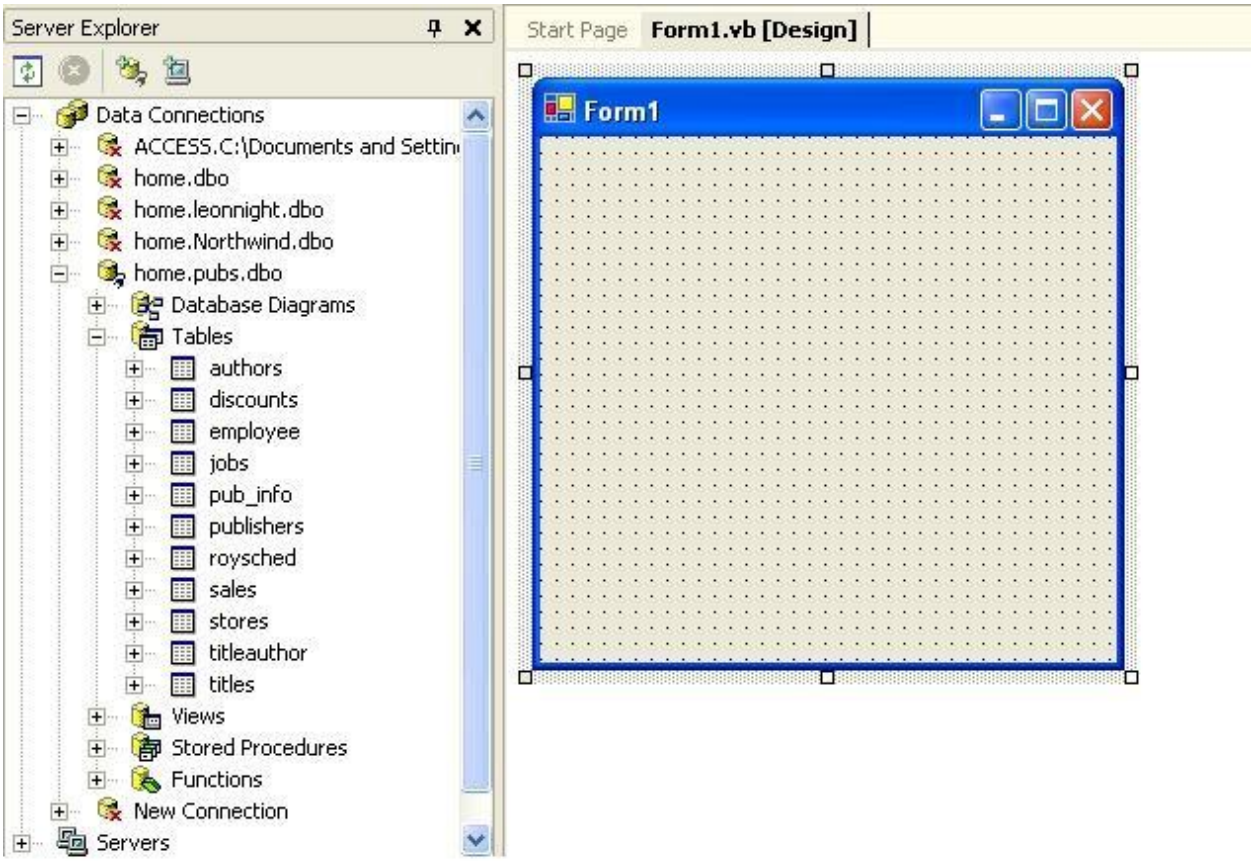


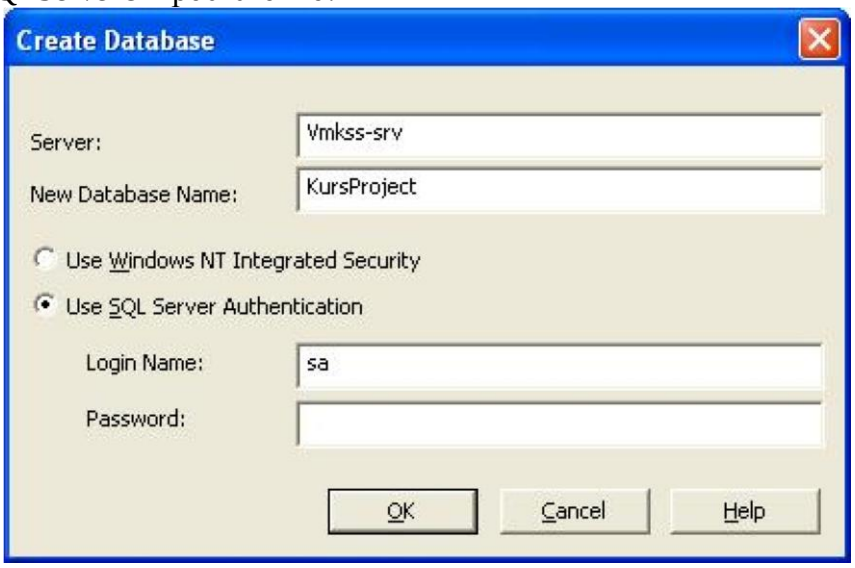
Рис. 1. Окноновогопроекта

Длясозданияновойбазыданныхна**SQLServer:**

Выберитепунктменю**Подключения данных**инажмитеправойкнопкоймыши. Откроетсяменю:

Обновить
Добавить подключение
Создать новую базу данных SQLServer

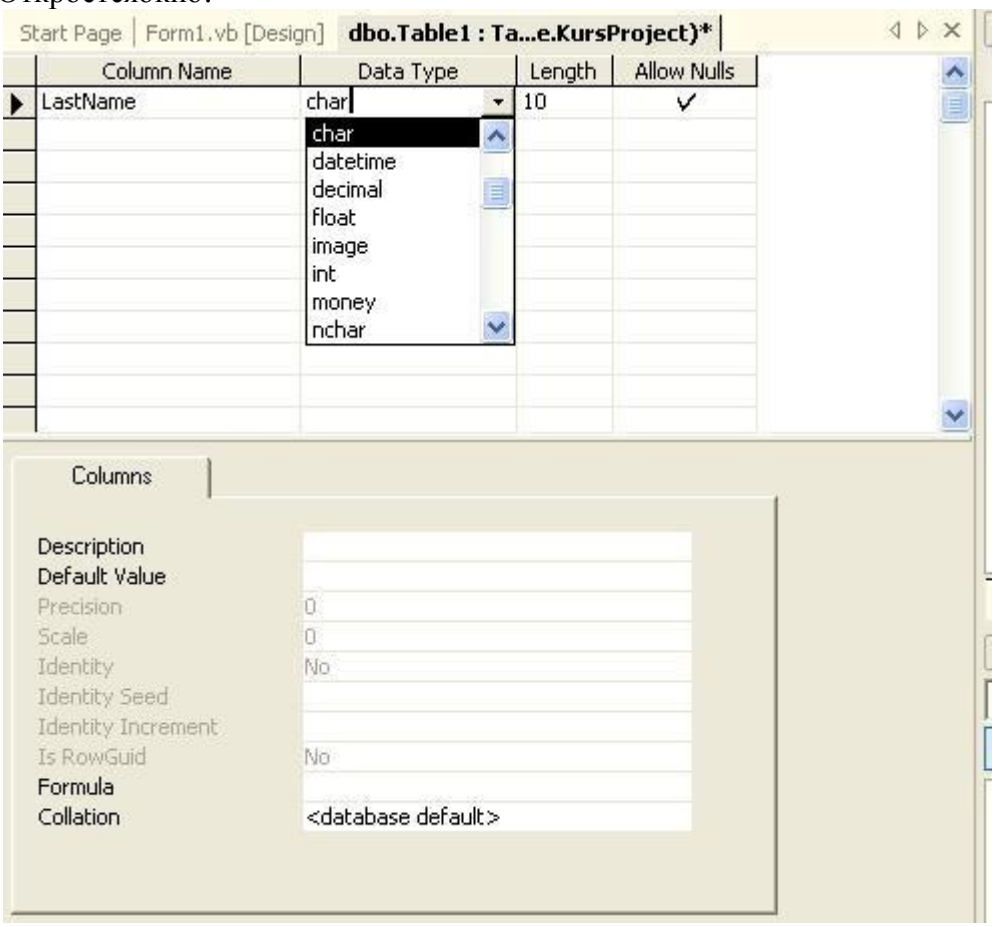
ВыберитеСоздать новую базу данных SQLServerОткроетсяокно:



Введитеимяновойбазыданных (KursProject)

Послещелчкаправойкнопкойна**Таблицы**, выберите**Новая Таблица**

Откроетсяокно:



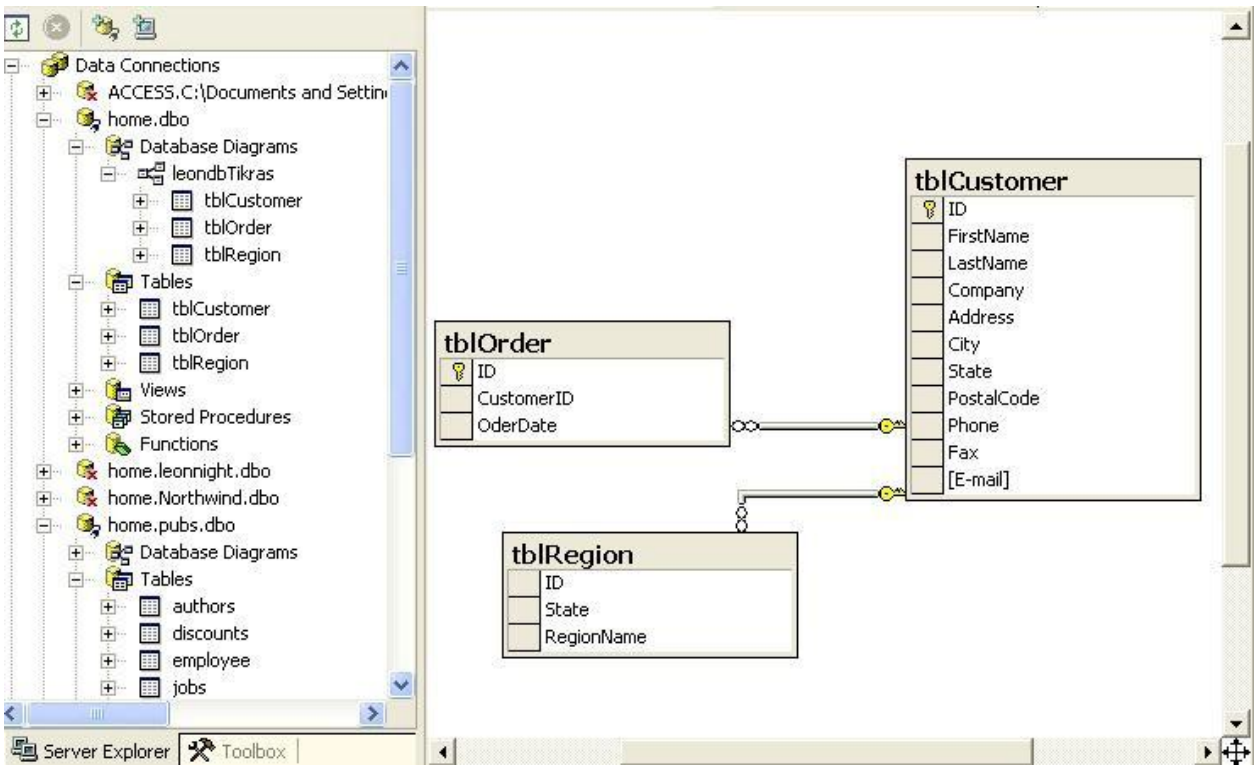
Работа с SQLServerиз VisualStudio.NET

2.14.1 Задание для работы:

Создайте схему базы данных. Правой кнопкой щелкните на **Схемы баз данных**

Добавить новую схему

Получится схема, похожая на эту:



Работа с базой данных

Построим приложение, позволяющееепросматриватьиредактироватьданныеизсозданнойвпредыдущемразделебазыданныхSQL Server.

ВзаимодействиесБДбудемстроитьнаосноветехнологии*ADO.NET*

Увеличьтеразмерыформы, растянувеемышьюзаправыйнижнийугол.

Построим приложение, работающее с базой данных на **SQL Server**.

СоединимсясСерверомспомощью ADO.NET.

Выберитепунктменю**Подключения данных**инажмитеправойкнопкоймыши.

Откроетсяменю:

Обновить
Добавить подключение
Создать новую базу данных SQLServer

Рис. 7.8. Выборпоставщикаданных (провайдера) приустановкесоединениясбазойданных

Выберите ***Добавитьподключение***

НавкладкеПодключениенажмитекнопкусправаотполяВыберите или введите имя базы данных, затемвоткрывшемсядиалоговомокненайдитеивыделитефайлсозданнойнамибазыданных Stud.mdb (рис. 7.9) инажмитекнопкуОткрыть — полныйпутькБДпоявитсявполе Выберите или введите имя базы данных (рис. 7.10).

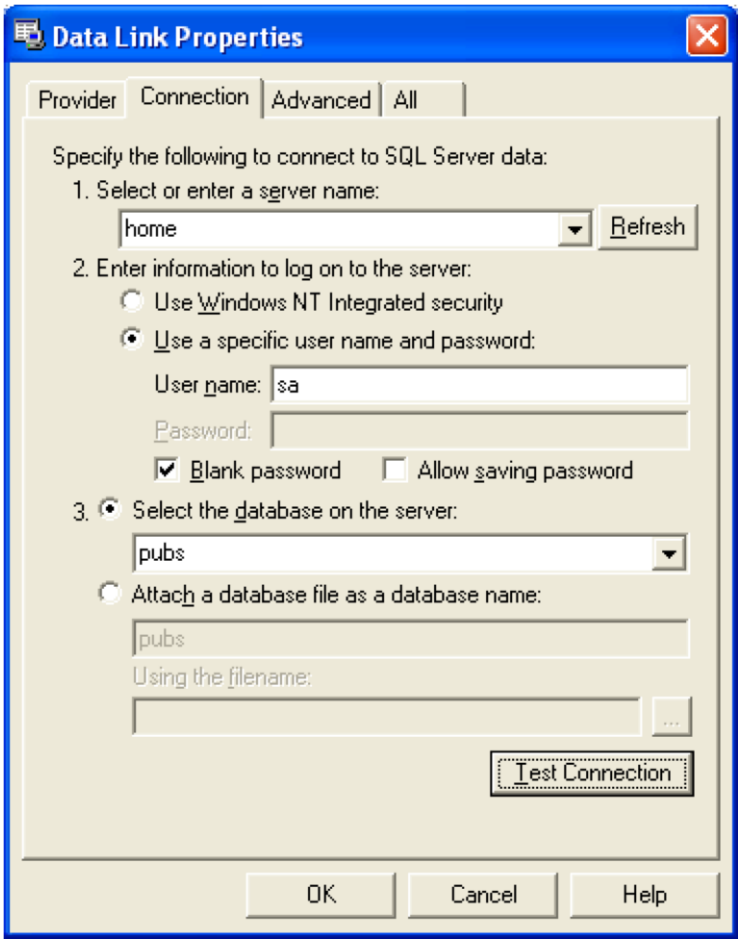


Рис. 7.10 Окно
Нажмите кнопку Проверить подключение. Появится сообщение об успешном соединении:

Нажмите ОК для закрытия диалоговых окон — на панели Обозреватель серверов появится новое соединение.

2.8 Практическое занятие №29, 30, 31, 32 (8 часов).

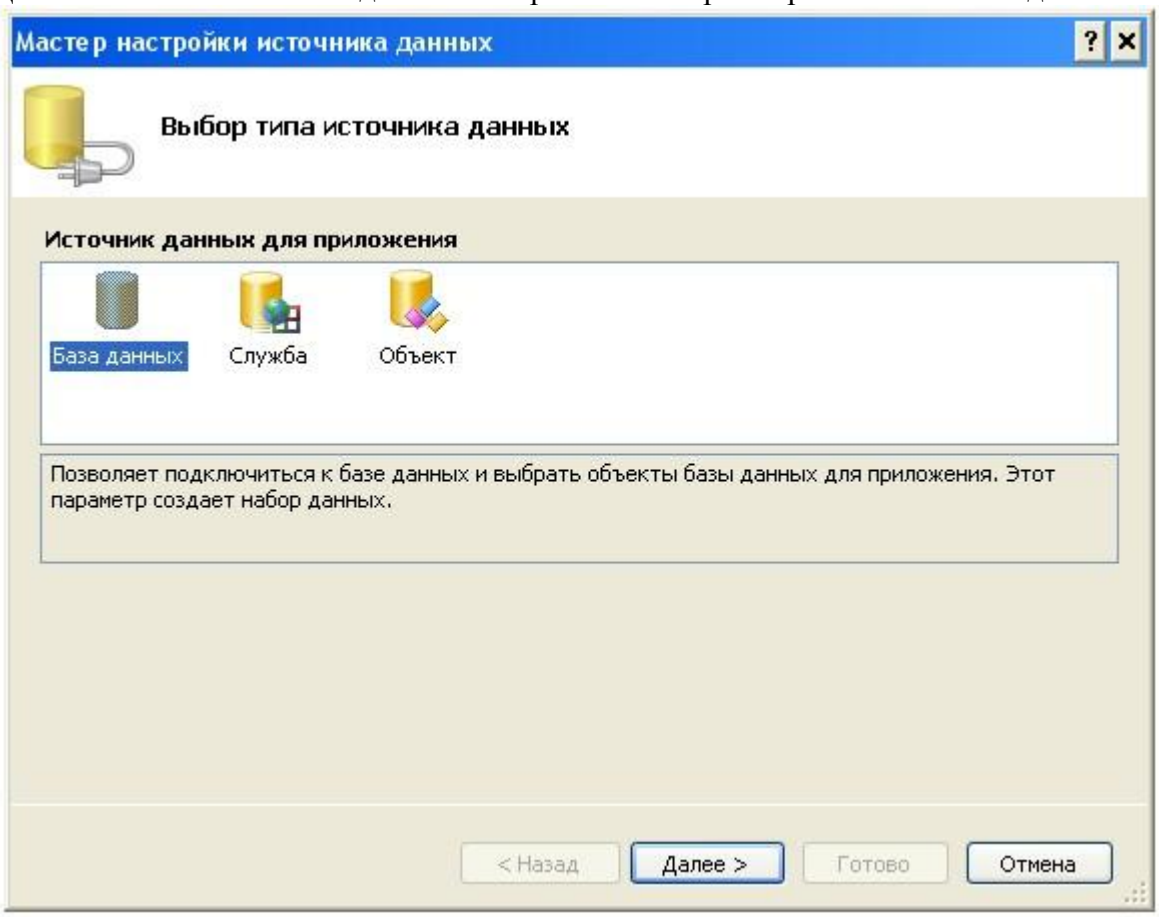
Тема: «Триггеры»
Работа с SQLServer из VisuialStudio.NET

2.15.1 Задание для работы:

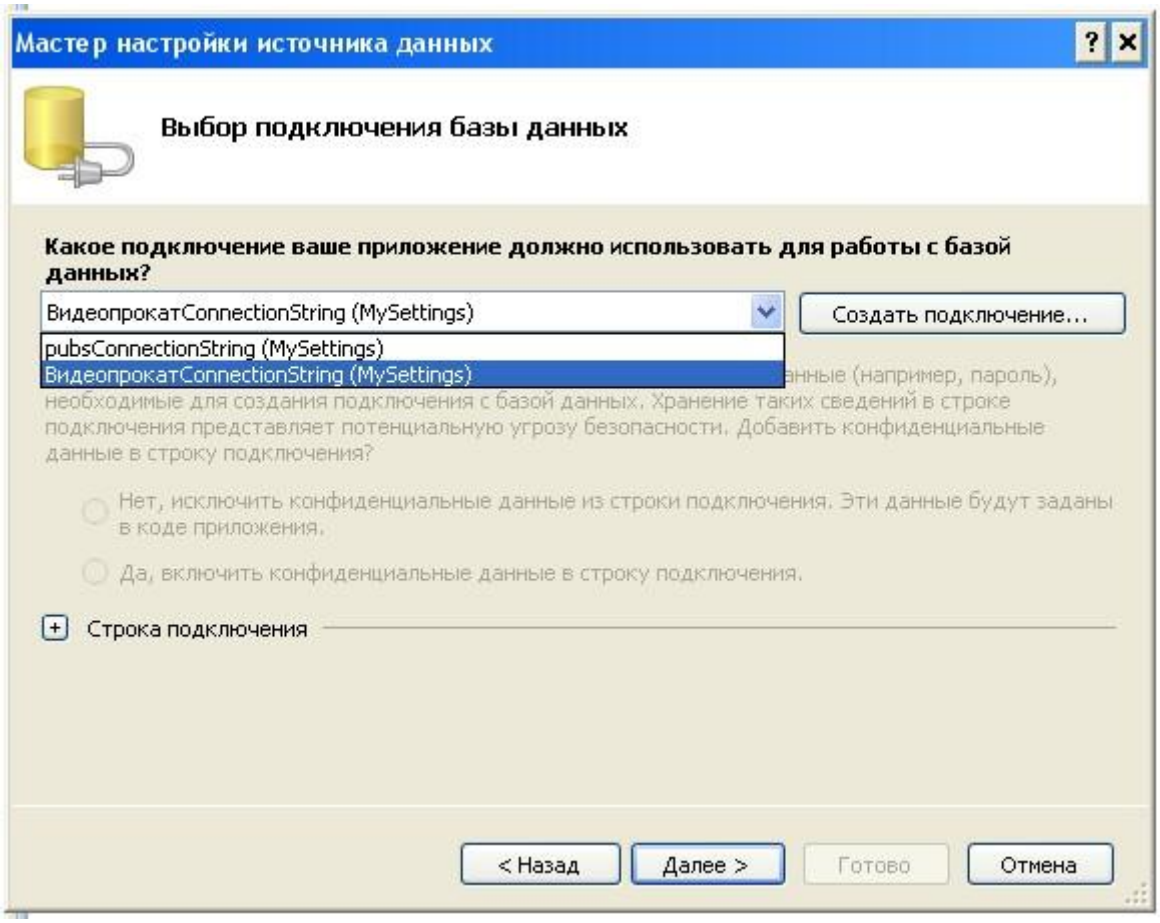
Для включения таблицы Authors из созданного соединения в проект:

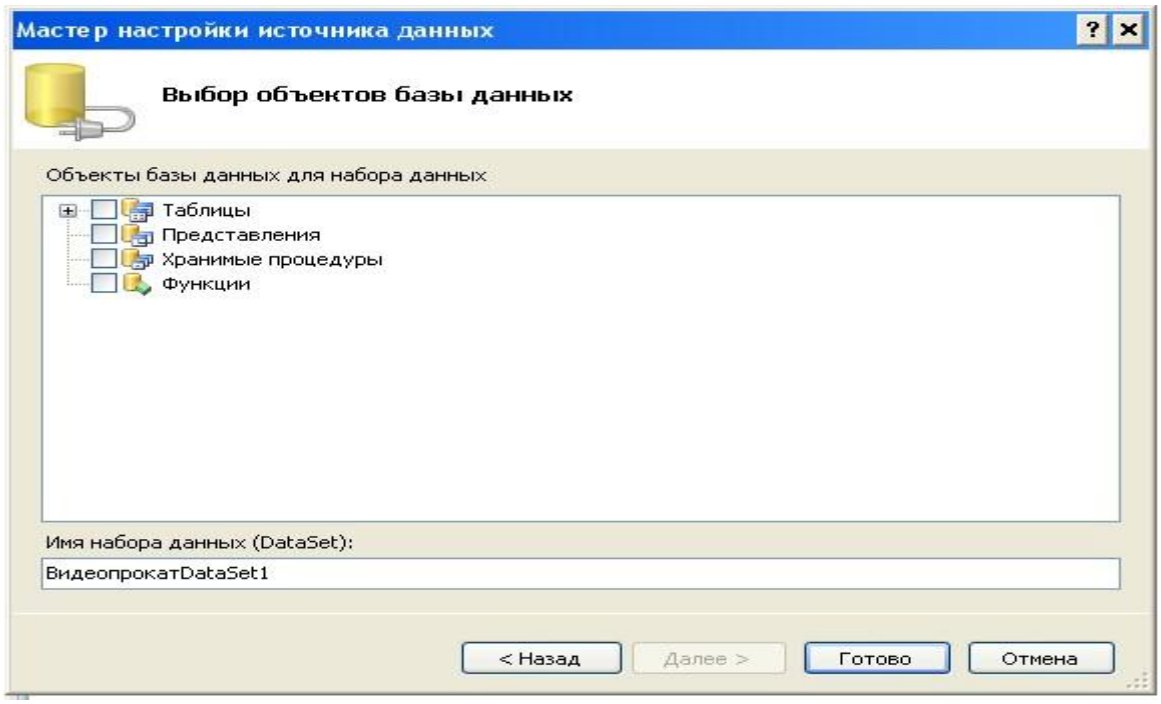
1. Выберите пункт главного меню Данные. В открывшемся списке выберите

Добавить новый источник данных. Откроется мастер настройки источника данных



Выберите База данных
В следующем окне выберите ваше подключение из списка. Нажмите далее.





Выберите все необходимые вам объекты (как минимум таблицы)
Нажимаем Готово.