ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ СТАВРОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ ЗАДАНИЙ ПО МДК 11.01 ТЕХНОЛОГИЯ РАЗРАБОТКИ И ЗАЩИТЫ БАЗ ДАННЫХ

для специальности: 09.02.07 «Информационные системы и программирование»

Ставрополь, 2024

Критерии оценивания практических умений:

«Отлично» - ставится, если студент:

а) уверенно и правильно выполняет манипуляцию в точном соответствии с алгоритмом;

б) обнаруживает полное понимание целей выполняемой манипуляции, может обосновать свои действия, пользуясь профессиональной терминологией, правильно отвечает на дополнительные вопросы;

в) свободно владеет речью (демонстрирует связность и последовательность в изложении);

г) демонстрирует умение действовать в стандартных и нестандартных профессиональных ситуациях.

«Хорошо» - ставится, если студент обнаруживает практические умения, удовлетворяющие тем же требованиям, что и для отметки «отлично», но допускает единичные негрубые ошибки, которые сам же исправляет после замечания преподавателя.

«Удовлетворительно» - ставится, если студент обнаруживает практические умения, но:

а) допускает неточности при выполнении алгоритма, не приводящие к негативным последствиям, затрудняется обосновать свои действия, затрудняется при ответе на дополнительные вопросы;

б) излагает материал недостаточно связно и последовательно с частыми заминками и перерывами;

в) испытывает затруднения в действиях при нестандартных профессиональных ситуациях.

«Неудовлетворительно» - ставится, если студент допускает грубые нарушения алгоритма действий и ошибки, влекущие за собой возникновение последствий, отсутствие умения действовать в стандартных профессиональных ситуациях.

Оглавление

Практическая работа № 1

Знакомства с MS SQL Server

1. Цель работы

Изучить интерфейс среды управления SQL Server Management Studio.

2. Теоретическая часть

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).

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.

Microsoft SQL Server является одной из наиболее популярных систем управления базами данных в мире. Данная СУБД подходит для самых различных проектов: от небольших приложений до больших высоконагруженных проектов. SQL Server был создан компанией Microsoft. Первая версия вышла в 1987 году. SQL Server долгое время был исключительно системой управления базами данных для Windows, однако начиная с версии 16 эта система доступна и на Linux (и на Mac, с помощью Docker).

Центральным аспектом в MS SQL Server, как и в любой СУБД, является база данных. База данных представляет хранилище данных, организованных определенным способом. Нередко физически база данных представляет файл на жестком диске, хотя такое соответствие необязательно. Для хранения и администрирования баз данных применяются системы управления базами данных (database management system) или СУБД (DBMS). И как раз MS SQL Server является одной из такой СУБД.

Для организации баз данных MS SQL Server использует реляционную модель. Эта модель баз данных была разработана еще в 1970 году Эдгаром Коддом. А на сегодняшний день она фактически является стандартом для организации баз данных. Реляционная модель предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта.

Для взаимодействия с базой данных применяется язык SQL (Structured Query Language). Клиент (например, внешняя программа) отправляет запрос на языке SQL посредством специального API. СУБД должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения.

Язык SQL был разработан в компании IBM для системы баз данных, которая называлась System/R. При этом сам язык назывался SEQUEL, расшифровывалась как Structured English QUEry Language – «структурированный английский язык запросов». Хотя в итоге ни база данных, ни сам язык не были впоследствии официально опубликованы, по традиции сам термин SQL нередко произносят как "сиквел".

В 1989 году Американский Национальный Институт Стандартов (ANSI) кодифицировал язык и опубликовал его первый стандарт. После этого стандарт периодически обновлялся и дополнялся. Последнее его обновление состоялось в 2016 году (SQL:2016). Но несмотря на наличие стандарта нередко производители СУБД используют свои собственные реализации языка SQL, которые немного отличаются друг от друга. Практически в каждой СУБД применяется свой процедурный язык, в частности, в Oracle Database используется PL/SQL (поддерживается также в DB2 и Timesten), в Interbase и Firebird – PSQL, в DB2 – SQL PL, в Microsoft SQL Server и Adaptive Server Enterprise – Transact-SQL, в PostgreSQL – PL/pgSQL. В рамках текущей книги будет рассматриваться T-SQL.

Подмножества языка SQL:

DDL (Data Definition Language / Язык определения данных). К этому типу относятся различные команды, которые создают базу данных, таблицы, индексы, хранимые процедуры и т.д.

DML (Data Manipulation Language / Язык манипуляции данными). К этому типу относят команды на выбор данных, их обновление, добавление, удаление – в общем все те команды, с помощью которых можно управлять данными.

DCL (Data Control Language / Язык управления доступа к данным). К этому типу относят команды, которые управляют правами доступа к данным.

3. Практическая часть

3.1. Установка SQL Server Developer Edition

SQL Server Developer – обладающий полным набором функций бесплатный выпуск, лицензируемый, для использования в качестве базы данных, для разработки и тестирования, и не предназначенный для применения в рабочей среде.

Для установки SQL Server с сайта https://www.microsoft.com/ru-ru/sql-server/sql-server-downloads выберите Developer Edition и нажмите кнопку «Скачать». После скачивание автоматически запускается установщик. Выберите тип установки – «Базовая» (рис. 1):

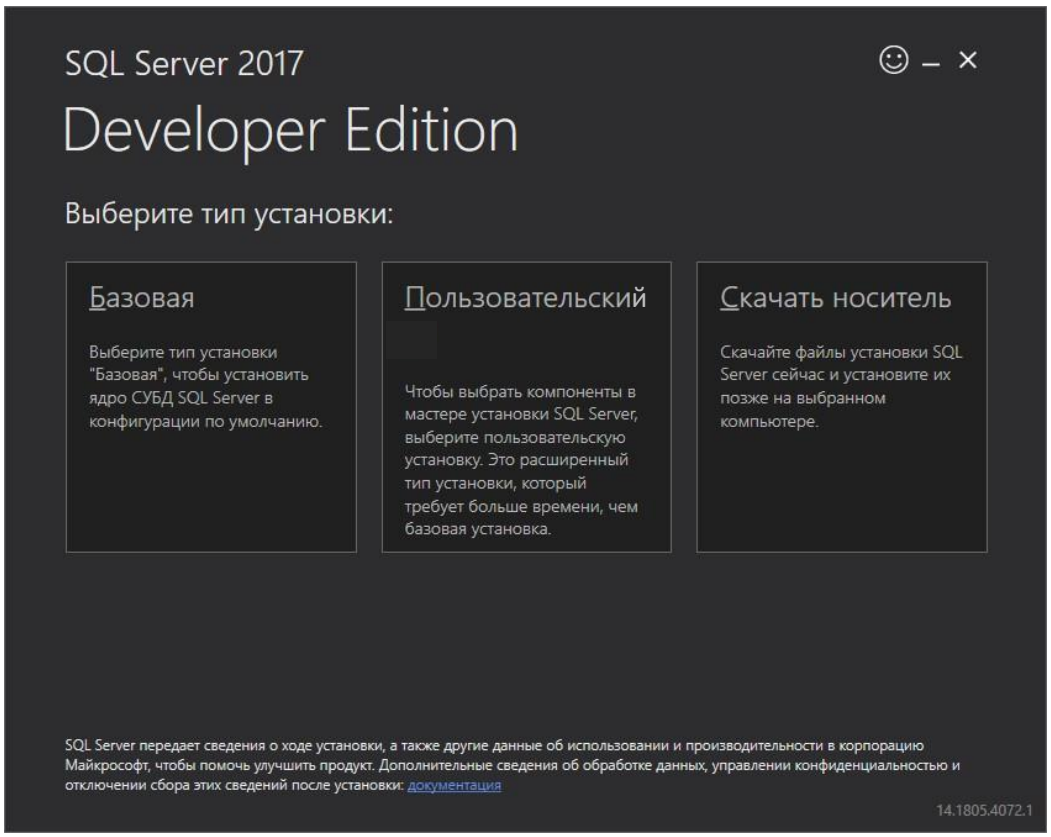

Рис. 1. Выбор тип установки

После принятия условия соглашения, укажите целевое расположение SQL Server.

3.2. Установка SQL Server Management Studio

SQL Server Management Studio (SSMS) – это интегрированная среда для управления любой инфраструктурой SQL, от SQL Server до баз данных SQL Azure. SSMS предоставляет средства для настройки, наблюдения и администрирования экземпляров SQL Server и баз данных. С помощью SSMS можно развертывать, отслеживать и обновлять компоненты уровня данных, используемые вашими приложениями, а также создавать запросы и скрипты.

Скачайте SQL Server Management Studio (SSMS) с сайта: https://docs.microsoft.com/ru-ru/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15

3.3. Открытие среды SQL Server Management Studio

- Запускайте SQL Server Management Studio.
- В диалоговом окне **Соединение с сервером** подтвердите заданные по умолчанию параметры и нажмите кнопку **Подключиться**. Для соединения необходимо, чтобы поле Имя

сервера содержало имя компьютера, на котором установлен SQL Server. Если компонент Database Engine представляет собой именованный экземпляр, то поле «Имя сервера» должно также содержать имя экземпляра в формате <имя_компьютера> \<имя_экземпляра>.

- Выбрать имя сервера (в данном случае это REM-DESKTOP).

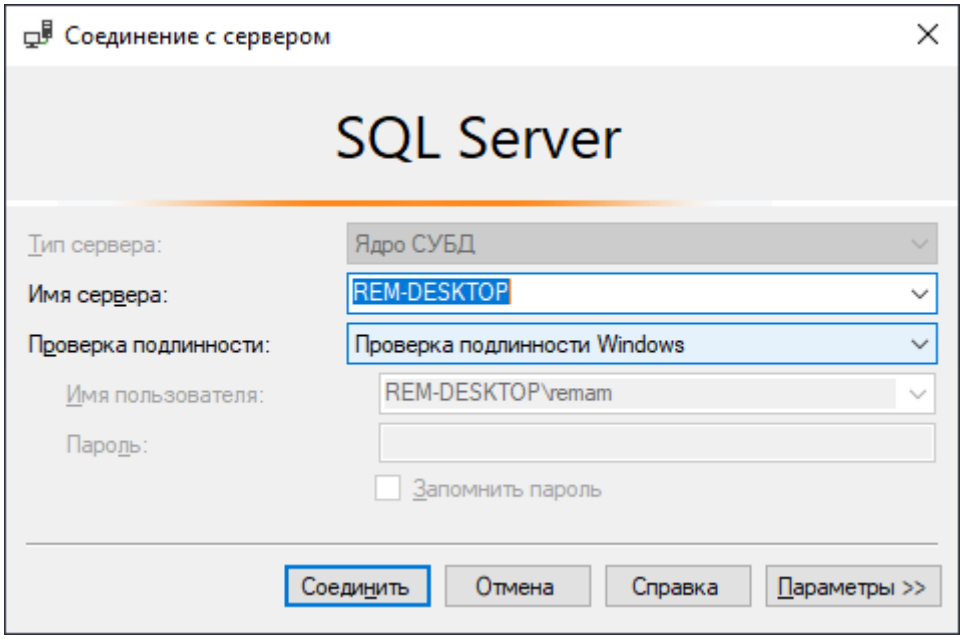

Рис. 2. Соединение с SQL Server

MS SQL Server использует два режима аутентификации: аутентификацию Windows и аутентификацию MS SQL Server (SQL Server authentication). Для аутентификации по умолчанию используется аутентификация Windows. Windows Authentification mode (Проверка подлинности Windows) – это режим, который использует для подключения к серверу только логины Windows (рис. 2). В этом случае пользователям нет необходимости вводить какие-то пароли при подключении к SQL Server, если они уже вошли в сеть Windows.

Для использования аутентификации SQL Server вначале необходимо создать учетную запись в самом SQL Server. В отличие от логинов Windows, логины SQL Server – это самостоятельные учетные записи со своими именами и паролями, информация о которых хранится в системной базе данных SQL Server. При подключении к серверу при помощи логина SQL Server вам придется указать имя логина и пароль.

3.4. Создание учетной записи

Создайте новую учетную запись с именем STUDENT. В Management Studio список учетных записей, сконфигурированных на сервере, содержится в папке **«\Безопасность\Имена для входа»**. Чтобы добавить новую учетную запись, необходимо выделить узел **«Имена для входа»** в контекстном меню и выбрать пункт **«Создать имя для входа…»**.

В открывшемся окне (рис. 3) в поле **«Имя для входа»** введите *STUDENT*. Далее выберите переключатель **«Проверка подлинности SQL Server»** и в поле **«Пароль»** наберите пароль. Снимите флажок **«Пользователь должен сменить пароль при следующем входе»**. Остальные поля оставьте без изменений.

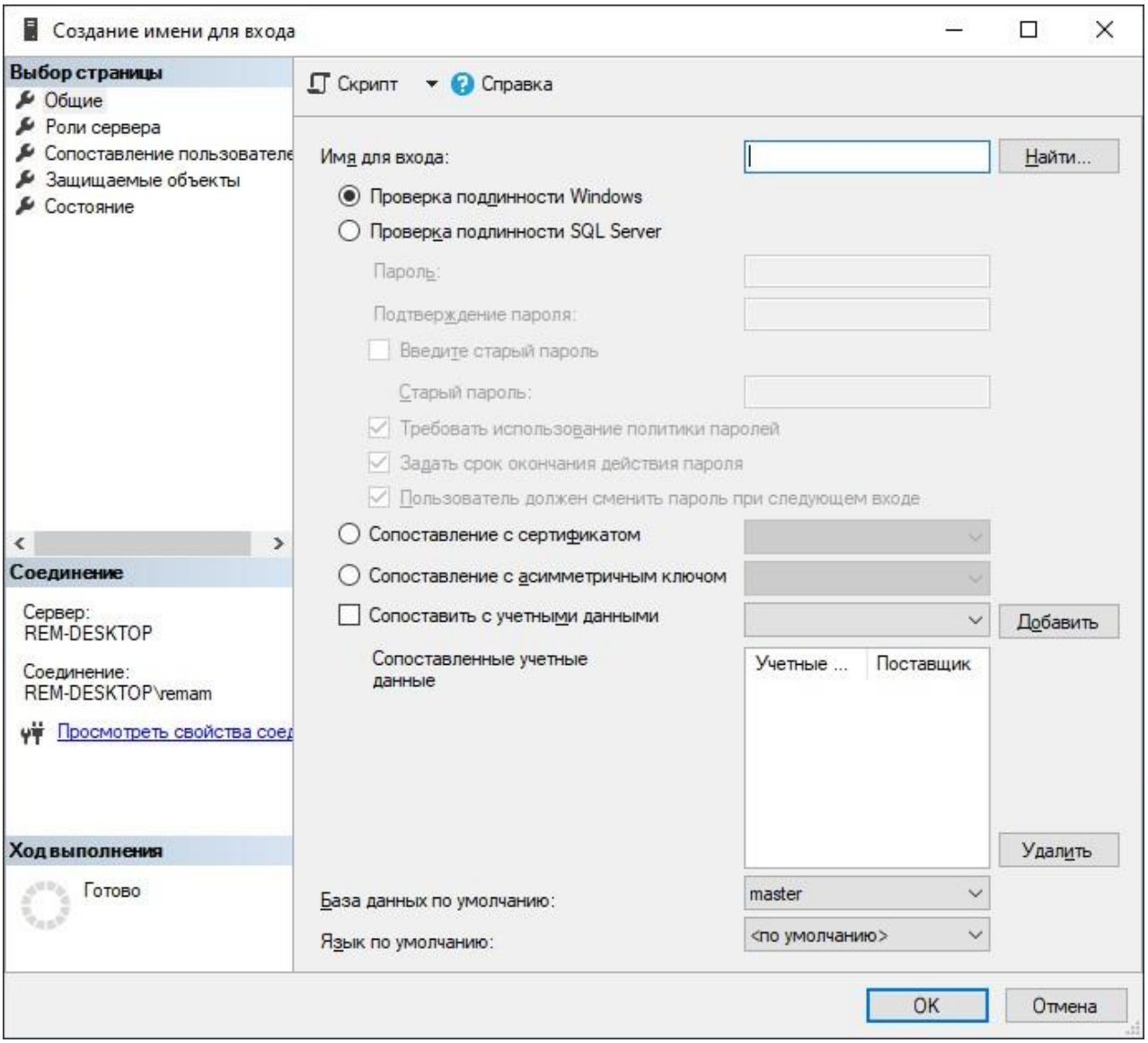

Рис. 3. Создание новой учетной записи

3.5. Создание базы данных

Базу данных может создать только пользователь с правами администратора. Выберите в контекстном меню папки **«Базы данных»** команду **«Создать базу данных…»** (рис. 4).

В поле **«Имя базы данных»** введите имя создаваемой базы данных (БД) – **«Учебная»** (рис. 5). Здесь также можно изменить путь сохранения созданной БД. *Обратите внимание, что создаётся пустая база данных (контейнер).*

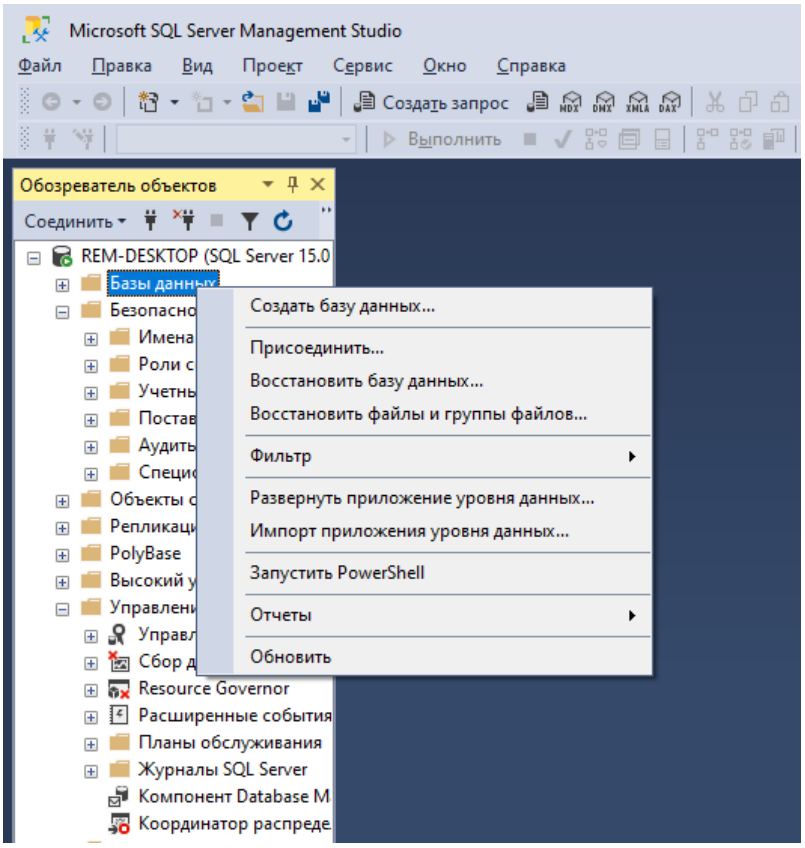

Рис. 4. Пункт контекстного меню «Создать базу данных…»

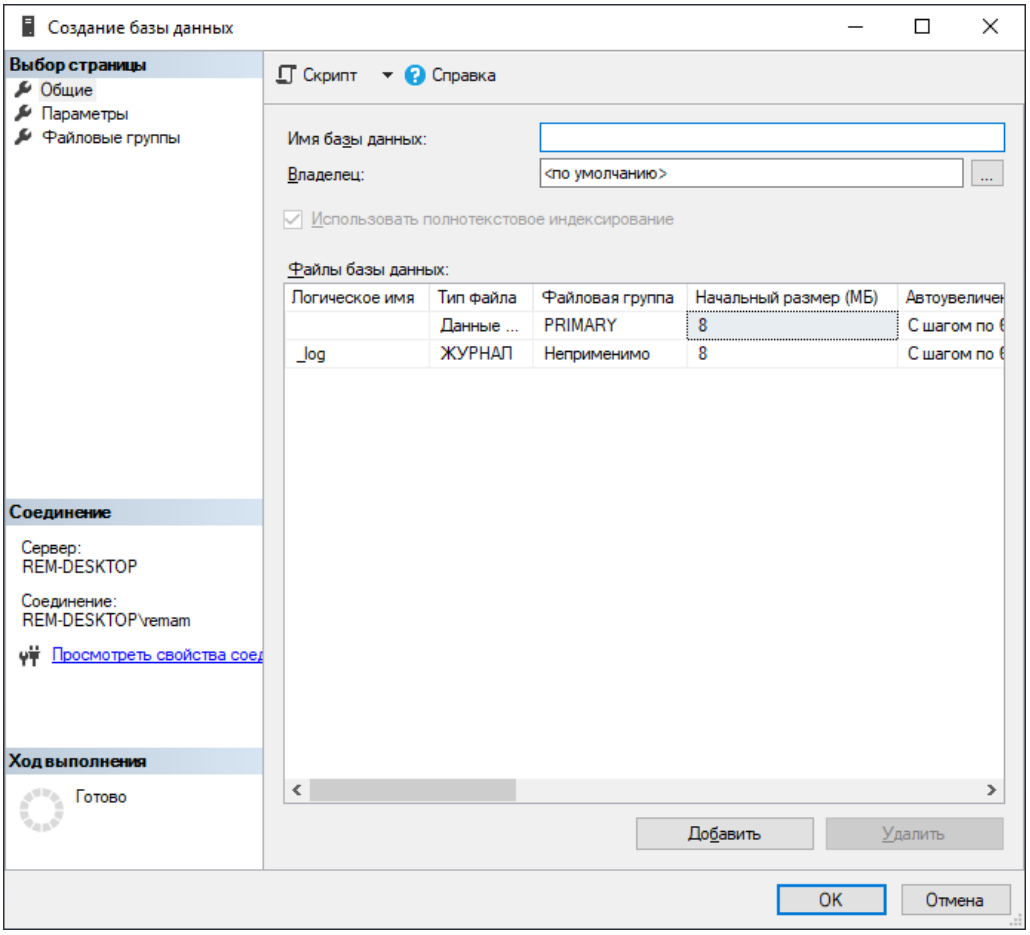

Рис. 5. Создание новой базы данных

После создания БД окно **«Обозреватель объектов»** обновится. В ветке **«Базы данных»** появится новая база данных **«Учебная»**. Эта база данных принадлежит пользователю по имени **sysadmin** (системный администратор).

3.6. Создание нового пользователя

Создание пользователя в SQL Server осуществляется посредством создания учетной записи. Пользователи БД – это специальные объекты, которые создаются на уровне базы данных и используются для предоставления разрешений в базе данных (на таблицы, представления, хранимые процедуры). Логины и пользователи БД – это совершенно разные объекты.

Чтобы добавить в базу данных **«Учебная»** нового пользователя, надо:

1. выделить узел **«\Базы данных\Учебная\Безопасность\Пользователи»** и в контекстном меню выбрать пункт **«Создать пользователя…»** (рис. 6).

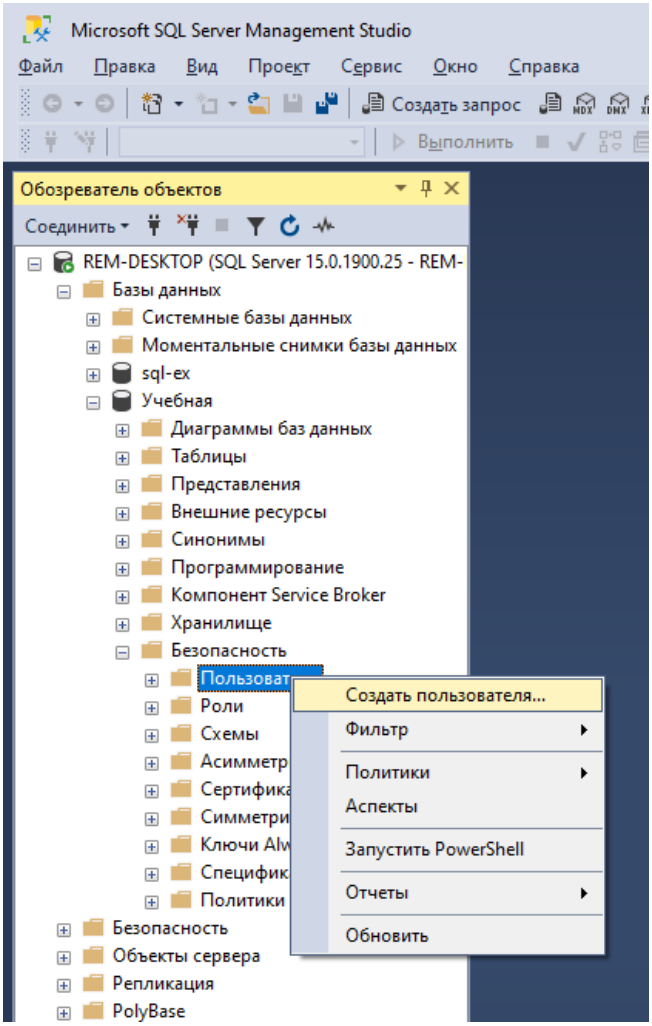

Рис. 6. Пункт контекстного меню «Создать пользователя...»

2. В открывшемся окне в поля **«Имя пользователя»** и **«Имя для входа»** ввести имя – *Student*.

3. На странице **«Собственные схемы»** в списке **«Схемы, принадлежащие данному пользователю:»** установить флажок **db_owner**→OK (рис. 7).

Таким образом, на основе учетной записи *Student* будет **создан новый пользователь** для БД «Учебная» с правами владельца этой базы данных.

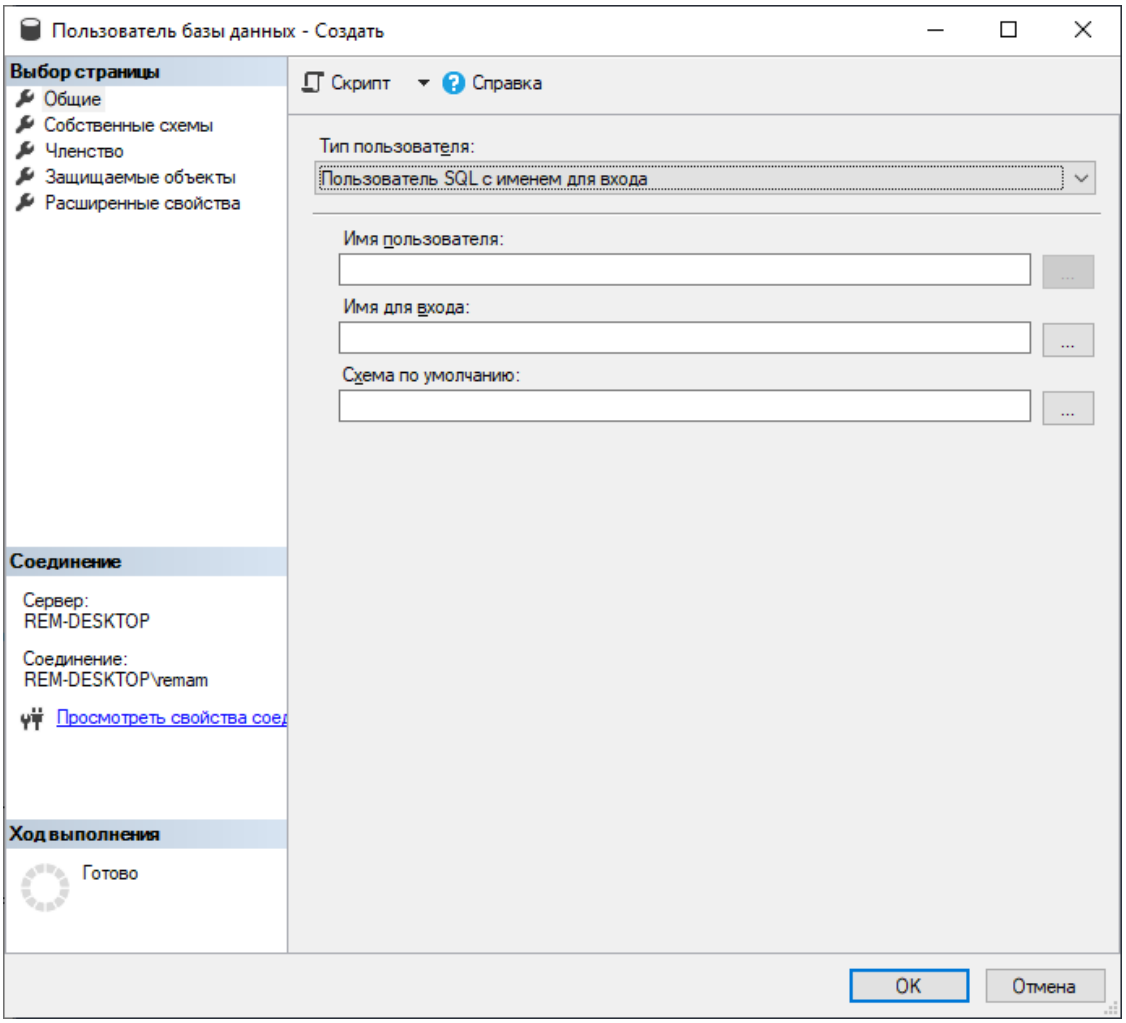

Рис. 7. Добавление в БД нового пользователя

3.7. Создание и заполнение таблиц

В «Обозревателе объектов» выберите базу данных «Учебная». На строке инструментов нажмите «Создать запрос» или используйте горячие клавиши <Ctrl>+<N> (рис. 8).

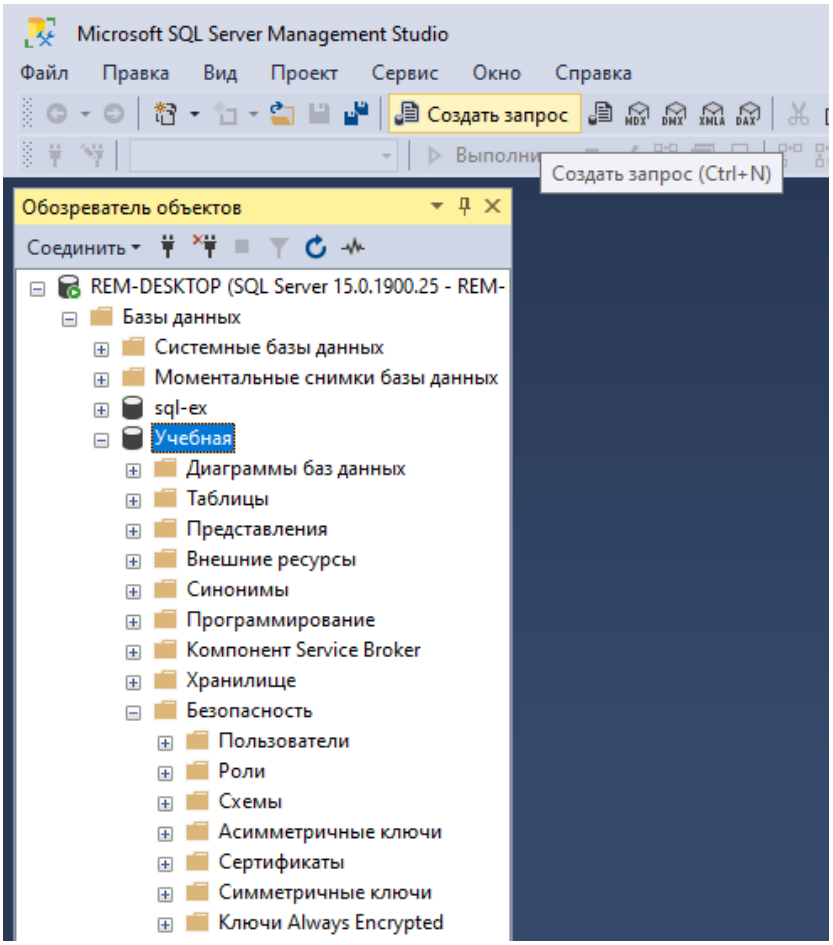

Рис. 8. Выбор БД «Учебная»

На открывшим окне редактора наберите код из приложения.

На строке инструментов нажмите «Выполнить» или используйте горячую клавишу <F5> для выполнения команд.

3.8. Резервное копирование БД

Сведения, хранящиеся в базах данных, являются стратегической ценностью любой компании, поэтому их сохранность очень важна.

1. Выберем в контекстном меню базы данных «Учебная» пункт «Задачи» → «Создать резервную копию» (рис. 9).

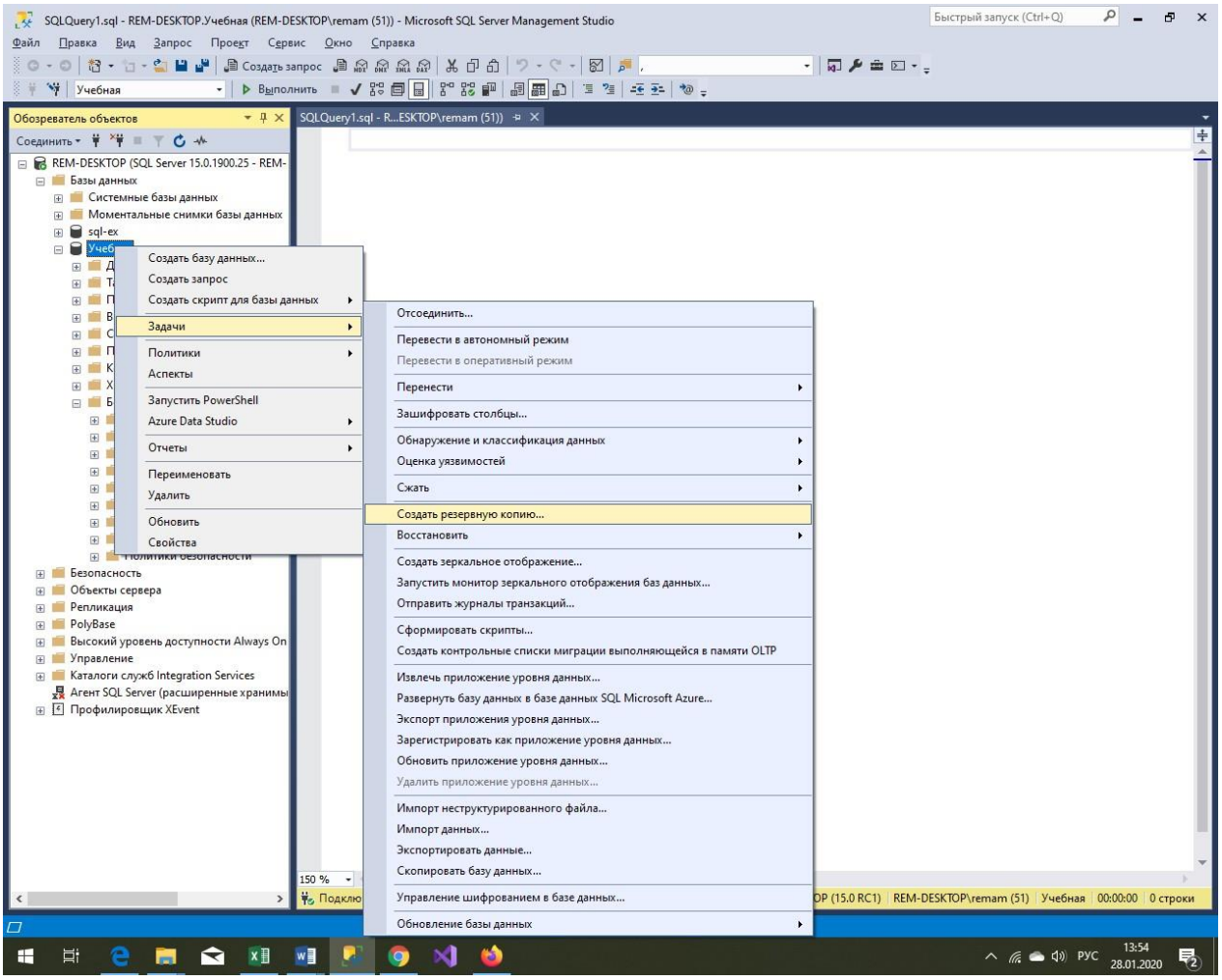

Рис. 9. Пункт контекстного меню «Создать резервное копию…»

2. В открывшемся диалоге «Резервное копирование базы данных – Учебная» (рис. 10) установите некоторые параметры:

- База данных – *Учебная* (это база данных для резервного копирования);
- Тип архивной копии – *Полная* (тип резервной копии – полный; для первого резервного копирования применяется только такой; если резервная копия уже имеется, и вы желаете не создавать заново полную копию, а внести изменения в старую, выберите *Разностная*);
- Компонент архивной копии – *База данных*;
- Архивировать в: – *Диск*.

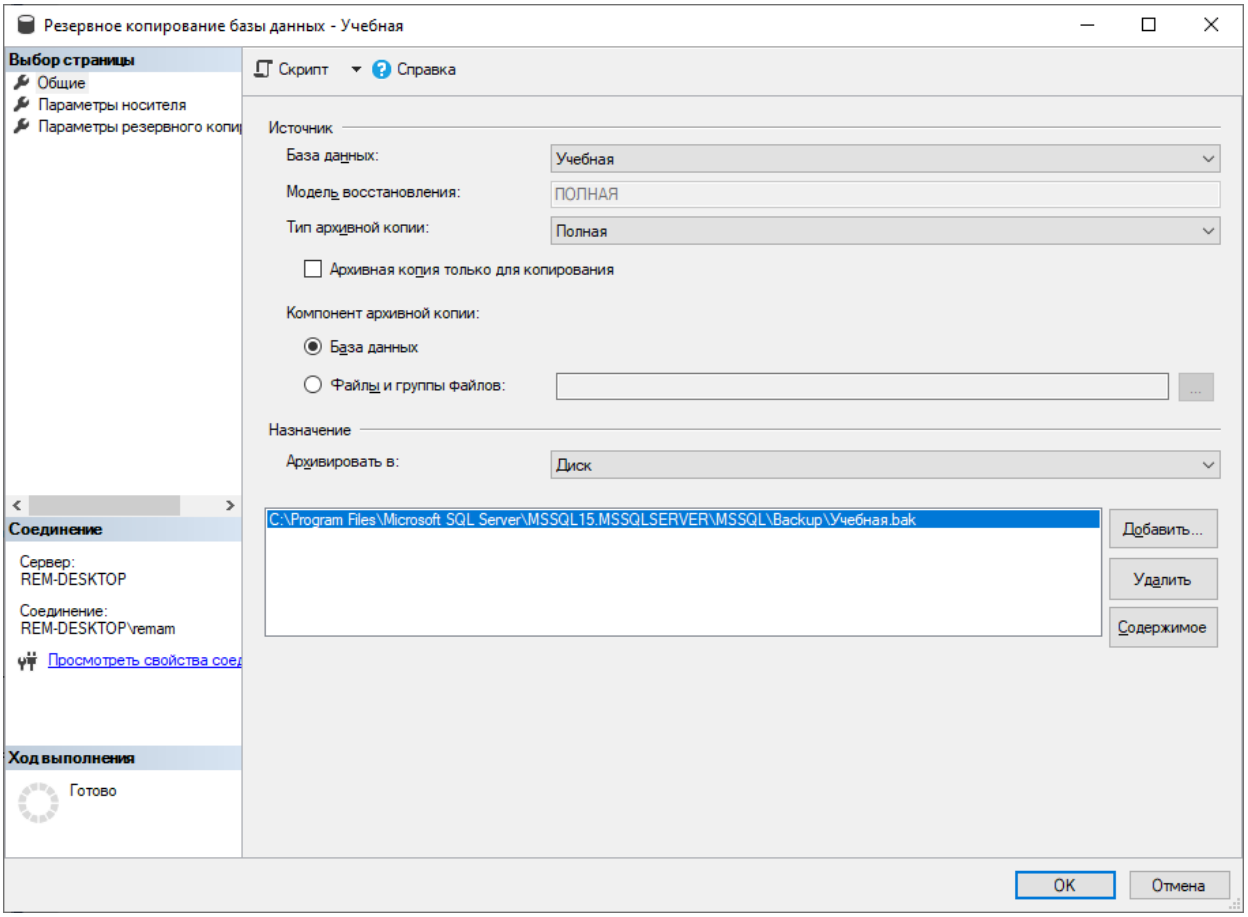

Рис. 10. Диалоговое окно Back Up Database

3. Нажмите кнопку «Добавить…» в этом диалоге. Откроется диалоговое окно «Выбор места расположения резервной копии» (рис. 11). В поле окна «Имя файла» введите имя файла, в который будет выполнено резервное копирование.

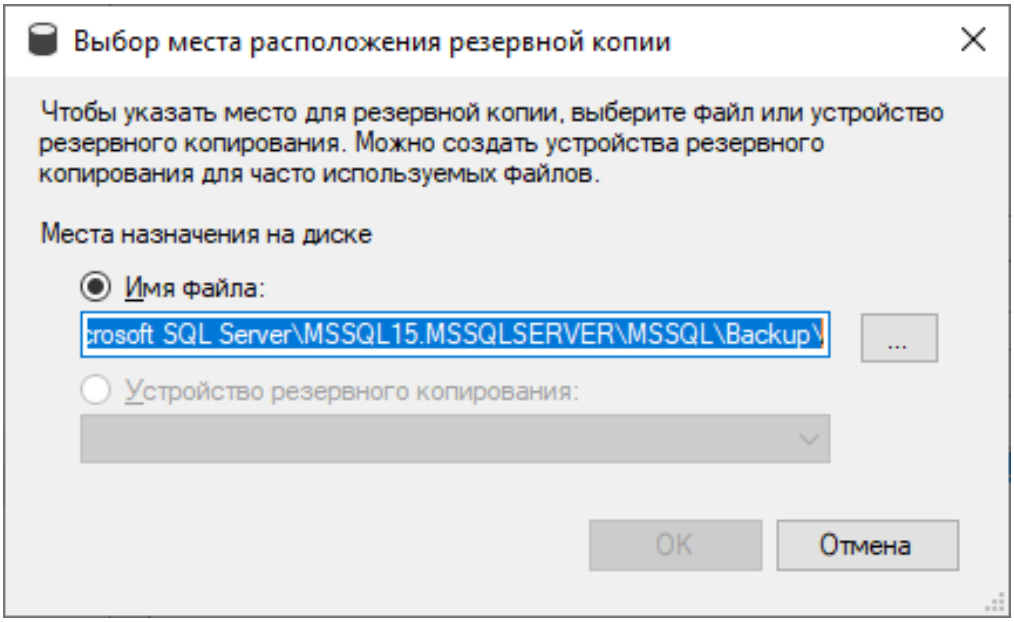

Рис. 11. Диалоговое окно «Выбор места расположения резервной копии»

4. Нажмите ОК в диалоговом окне «Выбор места расположения резервной копии». Оно закроется, а в списке «Назначение» появится строка, которую ввели. Выберите ее и нажмите кнопку ОК. Произойдет резервное копирование по указанному пути.

4. Задание

1. Установите SQL Server Developer Edition.
2. Установите SQL Server Management Studio (SSMS).
3. Создайте учетную запись пользователя MS SQL-Server.
4. Создайте новую базу данных.
5. Создайте нового пользователя базы данных.
6. Соединитесь с базой данных под учетной записью пользователя.
7. Создайте структуру базы в соответствии с порядком выполнения работы.
8. Выполните резервное копирование базы данных.

Практическая работа № 2

Команда SELECT

1. Цель работы

1. Изучение основ выборки данных.
2. Изучение конструкции DISTINCT | ALL.
3. Изучение сортировки данных.
4. Изучение конструкции TOP.
5. Изучение конструкции OFFSET и FETCH.

2. Теоретическая часть

Подавляющее большинство пользователей используют SQL для организации выборки данных. Для выборки данных из БД используется запрос SELECT. Он позволяет фильтровать выбранные данные и преобразовать их к нужному виду. Результатом выполнения запроса SELECT является другая таблица, к которой снова может быть применен запрос SELECT.

Полный синтаксис инструкции SELECT сложен, однако основные предложения можно вкратце описать следующим образом:

[WITH { [XMLNAMESPACES ,] [<common_table_expression>] }]
SELECT select_list [INTO new_table]
[FROM table_source]
[WHERE search_condition]
[GROUP BY group_by_expression]
[HAVING search_condition]
[ORDER BY order_expression [ASC | DESC]]

Обработка элементов запроса SELECT выполняется в следующей последовательности:

1. FROM – определяет имена используемых таблиц;
2. WHERE – фильтрует строки таблицы в соответствии с заданными условиями;
3. GROUP BY – группирует строки, имеющие одинаковые значения в указанном столбце;
4. HAVING – фильтрует группы строк в соответствии с указанным условием;
5. SELECT – форматирует выходные данные;
6. ORDER BY – сортирует результаты выполнения запроса.

Порядок предложений в запросе SELECT не может быть изменен. Предложения SELECT и FROM являются обязательными, присутствие остальных зависит от контекста.

В предложении SELECT указывается список столбцов, которые должны быть возвращены запросом. Можно указать исходные элементы или вычисляемые поля во время выполнения запроса.

Конструкция DISTINCT | ALL исключает / разрешает вывод повторяющихся строк. Конструкция ALL используется по умолчанию.

* означает вывод всех столбцов указанной таблицы. В случае, если выборка производится из нескольких таблиц, перед символом звездочки может указываться имя таблицы.

SQL-запрос может содержать вычисляемые столбцы, значения которых могут определятся на основе значений данных, хранящихся в БД конструкции. Вычисляемым столбцам следует давать название с помощью ключевого слова AS.

Вычисляемый столбец можно создать как: <Новое поле> = <выражение>

Если название столбца состоит из нескольких слов, разделенных пробелами, следует их записать в квадратных скобках: [].

Сортировка данных выполняется с помощью команды ORDER BY, которая добавляется в конец запроса, после чего перечисляется список столбцов. Для каждого столбца указывается тип сортировки ASC | DESC (ascending – по возрастанию | descending – по убыванию). ASC – по умолчанию, можно не указывать.

Конструкция TOP <N> позволяет выбрать определенное количество строк из таблицы. Дополнительный оператор PERCENT позволяет выбрать процентное количество строк из таблицы. Дополнительный оператор WITH TIES позволяет выбрать все строки с такими же свойствами.

Конструкция OFFSET <N> ROWS указывает число строк, которые необходимо пропустить, прежде чем будет начат возврат строк из выражения запроса.

Конструкция FETCH NEXT <N> ROWS ONLY указывает число строк, возвращаемых после обработки предложения OFFSET.

На языке T-SQL регистр не имеет значение (case insensitive).

3. Практическая часть

Дана таблица ***Академики:***

ФИО	**Дата_рождения**	**Специализация**	**Год_присвоения_звания**
Аничков Николай Николаевич	1885-11-03	медицина	1939
Бартольд Василий Владимирович	1869-11-15	историк	1913
Белопольский Аристарх Аполлонович	1854-07-13	астрофизик	1903
Бородин Иван Парфеньевич	1847-01-30	ботаник	1902
Вальден Павел Иванович	1863-07-26	химик-технолог	1910
Вернадский Владимир Иванович	1863-03-12	геохимик	1908
Виноградов Павел Гаврилович	1854-11-30	историк	1914
Ипатьев Владимир Николаевич	1867-11-21	химик	1916

ФИО	Дата_рождения	Специализация	Год_присвоения_звания
Истрин Василий Михайлович	1865-02-22	филолог	1907
Карпинский Александр Петрович	1847-01-07	геолог	1889
Коковцов Павел Константинович	1861-07-01	историк	1906
Курнаков Николай Семёнович	1860-12-06	химик	1913
Марр Николай Яковлевич	1865-01-06	лингвист	1912
Насонов Николай Викторович	1855-02-26	зоолог	1906
Ольденбург Сергей Фёдорович	1863-09-26	историк	1903
Павлов Иван Петрович	1849-09-26	физиолог	1907
Перетц Владимир Николаевич	1870-01-31	филолог	1914
Соболевский Алексей Иванович	1857-01-07	лингвист	1900
Стеклов Владимир Андреевич	1864-01-09	математик	1912

Пример 1: Вывести список академиков:

```
SELECT
    *
FROM
    Академики
```

Пример 2: Вывести ФИО и дату рождения всех академиков:

```
SELECT
    ФИО, Дата_рождения
FROM
    Академики
```

Пример 3: Создайте вычисляемое поле «Информация», содержащее информацию об академиках в таком виде: «Академик Петров Петр Петрович, специализация: математика»:

```
SELECT
    'Академик ' + ФИО + ', специализация: ' + Специализация AS Информация
FROM
    Академики
```

Пример 4: Вывести ФИО академиков и номер следующего года после присвоения звания:

```
SELECT
    ФИО
    ,[Через год] = Год_присвоения_звания + 1
FROM
    Академики
```

Пример 5: Выведите список специализаций, убрав дубликаты:

```
SELECT DISTINCT
    Специализация
FROM
    Академики
```

Пример 6: Вывести список академиков, отсортированный по возрастанию года присвоения звания:

```
SELECT
    *
FROM
    Академики
ORDER BY
    Год_присвоения_звания
```

Пример 7: Вывести список академиков, отсортированный в обратном алфавитном порядке по полю «Специализация» и в алфавитном порядке по полю «ФИО»:

```
SELECT
    *
FROM
    Академики
ORDER BY
    Специализация DESC
,ФИО ASC
```

Пример 8: Вывести первые две строки из списка академиков, отсортированного в алфавитном порядке по полю «ФИО»:

```
SELECT TOP 2
    *
FROM
    Академики
ORDER BY
    ФИО ASC
```

Пример 9: Вывести первые 30% строк из списка академиков, отсортированного по возрастанию года присвоения звания:

```
SELECT TOP 30 PERCENT
    *
FROM
    Академики
ORDER BY
    Год_присвоения_звания
```

Пример 10: Вывести из таблицы «Академики», отсортированной по возрастанию года присвоения звания, список академиков, у которых год присвоения звания – один из первых четырех в отсортированной таблице:

```
SELECT TOP 4 WITH TIES
        *
FROM
        Академики
ORDER BY
        Год_присвоения_звания
```

Пример 11: Вывести, начиная с третьего, список академиков, отсортированный в алфавитном порядке ФИО:

```
SELECT
        *
FROM
        Академики
ORDER BY
        ФИО
        OFFSET 2 ROWS
```

Пример 12: Вывести, начиная с третьего и до десятого, список академиков, отсортированный в алфавитном порядке ФИО:

```
SELECT
        *
FROM
        Академики
ORDER BY
        ФИО
        OFFSET 2 ROWS
        FETCH NEXT 8 ROWS ONLY
```

4. Задание

1. Вывести ФИО, специализацию и дату рождения всех академиков.
2. Создать вычисляемое поле «О присвоении звания», которое содержит информацию об академиках в виде: «Петров Петр Петрович получил звание в 1974».
3. Вывести ФИО академиков и вычисляемое поле «Через 5 лет после присвоения звания».
4. Вывести список годов присвоения званий, убрав дубликаты.
5. Вывести список академиков, отсортированный по убыванию даты рождения.

6. Вывести список академиков, отсортированный в обратном алфавитном порядке специализаций, по убыванию года присвоения звания, и в алфавитном порядке ФИО.

7. Вывести первую строку из списка академиков, отсортированного в обратном алфавитном порядке ФИО.

8. Вывести фамилию академика, который раньше всех получил звание.

9. Вывести первые 10% строк из списка академиков, отсортированного в алфавитном порядке ФИО.

10. Вывести из таблицы «Академики», отсортированной по возрастанию года присвоения звания, список академиков, у которых год присвоения звания – один из первых пяти в отсортированной таблице.

11. Вывести, начиная с десятого, список академиков, отсортированный по возрастанию даты рождения.

12. Вывести девятую и десятую строку из списка академиков, отсортированного в алфавитном порядке ФИО.

Практическая работа № 3

Фильтрация данных

1. Цель работы

1. Изучение основ фильтрации данных.
2. Изучение операций сравнения.
3. Изучение логических операторов.
4. Изучение BETWEEN.
5. Изучение LIKE.
6. Изучение NULL.
7. Изучение IN.

2. Теоретическая часть

Для фильтрации данных применяется оператор WHERE. Синтаксис: WHERE <условие>. В условиях поле таблицы сравнивается с константой или с выражением. Символьные константы пишутся в одинарных кавычках. Числовые константы и названия столбцов пишутся без кавычек.

Чтобы строка попала в результат, условие должно быть истинно. В условиях используются операции сравнения. В Transact-SQL применяются следующие операции сравнения:

= – равенство;

<> или != – неравенство;

< – меньше;

> – больше;

!< – не меньше;

!> – не больше;

<= – меньше или равно;

>= – больше или равно.

Можно использовать несколько условий для фильтрации данных. Для объединения их в одно выражение используются логические операторы. В Transact-SQL применяются следующие логические операторы:

AND – логическое умножение или конъюнкция (И). Бинарный оператор, объединяет два условия; если оба условия истинны, результат – истина, иначе – ложь.

OR – логическое сложение или дизъюнкция (ИЛИ). Бинарный оператор, объединяет два условия; если хотя бы одно из этих условий истинно, то общее условие оператора OR также будет истинно.

NOT – логическое отрицание или инверсия (НЕ). Унарный оператор, применяется к одному условию. Если выражение в этой операции ложно, то общее условие истинно.

Самый высокий приоритет у оператора NOT. Самый низкий – у OR. Если эти операторы встречаются в одном выражении, то сначала выполняется NOT, потом AND, а затем OR. При записи условий использование скобок – хороший тон программирования.

Оператор BETWEEN используется для сравнения с диапазоном от начального и до конечного значения. Начальное и конечное значения включены в промежуток.

Оператор LIKE используется для сравнения с шаблоном строки. Для определения шаблона применяются специальные символы:

% – любая последовательность символов, в том числе пустая;

_ – любой символ;

[ ] – символ, который указан в квадратных скобках;

[-] – символ из определенного диапазона;

[^] – символ, который не указан после символа ^.

В базах данных, для обозначения неизвестного значения, используется понятие NULL. Для проверки неизвестного значения нельзя использовать операторы сравнения. Допускается только IS NULL или IS NOT NULL.

Оператор IN используется для сравнения с набором значений. Список значений указывается в скобках.

3. Практическая часть

Дана таблица ***Страны:***

Название	Столица	Площадь	Население	Континент
Австрия	Вена	83858	8741753	Европа
Азербайджан	Баку	86600	9705600	Азия
Албания	Тирана	28748	2866026	Европа
Алжир	Алжир	2381740	39813722	Африка
Ангола	Луанда	1246700	25831000	Африка
Аргентина	Буэнос-Айрес	2766890	43847000	Южная Америка
Афганистан	Кабул	647500	29822848	Азия
Бангладеш	Дакка	144000	160221000	Азия
Бахрейн	Манама	701	1397000	Азия
Белиз	Бельмопан	22966	377968	Северная Америка
Белоруссия	Минск	207595	9498400	Европа
Бельгия	Брюссель	30528	11250585	Европа
Бенин	Порто-Ново	112620	11167000	Африка
Болгария	София	110910	7153784	Европа
Боливия	Сукре	1098580	10985059	Южная Америка
Ботсвана	Габороне	600370	2209208	Африка

Название	Столица	Площадь	Население	Континент
Бразилия	Бразилиа	8511965	206081432	Южная Америка
Буркина-Фасо	Уагадугу	274200	19034397	Африка
Бутан	Тхимпху	47000	784000	Азия
Великобритания	Лондон	244820	65341183	Европа
Венгрия	Будапешт	93030	9830485	Европа
Венесуэла	Каракас	912050	31028637	Южная Америка
Восточный Тимор	Дили	14874	1167242	Азия
Вьетнам	Ханой	329560	91713300	Азия

Пример 1: Вывести список стран, площадь которых больше 1 млн. кв. км:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Площадь > 1000000
```

Пример 2: Вывести список стран, население которых не больше 1 млн. чел.:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Население !> 1000000
```

Пример 3: Вывести список африканских стран:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Континент = 'Африка'
```

Пример 4: Вывести список всех стран, кроме европейских:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Континент != 'Европа'
```

Пример 5: Вывести список стран, население которых больше 1 млн. чел., а площадь меньше 100 тыс. кв. км:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        (Население > 1000000) AND (Площадь < 100000)
```

Пример 6: Вывести список стран, которые находятся в Европе и их население больше 10 млн. чел., или находятся в Азии, а население больше 50 млн. чел.:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        (Континент = 'Европа') AND (Население > 10000000)
        OR
        (Континент = 'Азия') AND (Население > 50000000)
```

Пример 7: Вывести список стран, население которых от 10 до 100 млн. чел., а площадь от 100 до 200 тыс. кв. км:

```
SELECT Название
```

```
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        (Население BETWEEN 10000000 AND 100000000)
        AND
        (Площадь >= 100000) AND (Площадь <= 200000)
```

Пример 8: Вывести отсортированный в алфавитном порядке список стран от Бенина до Ватикана:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Название BETWEEN 'Бенин' AND 'Ватикан'
ORDER BY
        Название
```

Пример 9: Вывести список стран, название которых начинается с буквы «С»:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Название LIKE 'С%'
```

Пример 10: Вывести список стран, в названии которых вторая буква – «а», а последняя – я»:

```
SELECT Название
    ,Столица
    ,Площадь
```

```
    ,Население
    ,Континент
FROM
        Страны
WHERE
  Название LIKE '_а%я'
```

Пример 11: Вывести список стран, в названии которых третья буква – «а», «о» или «у»:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Название LIKE '__[аоу]%'
```

Пример 12: Вывести список стран, название которых начинается с буквы от «А» до «Г»:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Название LIKE '[А-Г]%'
```

Пример 13: Вывести список стран, название которых не начинается с буквы от «А» до «Г» или с буквы «С»:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Название LIKE '[^А-Г,^С]%'
```

Пример 14: Вывести список стран, столицы которых не введены в базу:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Столица IS NULL
```

Пример 15: Вывести список европейских, азиатских и африканских стран:

```
SELECT Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Континент IN ('Европа', 'Азия', 'Африка')
```

4. Задание

1. Вывести названия и столицы пяти наибольших стран по площади.
2. Вывести список африканских стран, население которых не превышает 1 млн. чел.
3. Вывести список стран, население которых больше 5 млн. чел., а площадь меньше 100 тыс. кв. км, и они расположены не в Европе.
4. Вывести список стран Северной и Южной Америки, население которых больше 20 млн. чел., или стран Африки, у которых население больше 30 млн. чел.
5. Вывести список стран, население которых составляет от 10 до 100 млн. чел., а площадь не больше 500 тыс. кв. км.
6. Вывести список стран, названия которых не начинаются с буквы «К».
7. Вывести список стран, в названии которых третья буква – «а», а предпоследняя – «и».
8. Вывести список стран, в названии которых вторая буква – гласная.
9. Вывести список стран, названия которых начинаются с букв от «К» до «П».
10. Вывести список стран, названия которых начинаются с букв от «А» до «Г», но не с буквы «Б».
11. Вывести список стран, столицы которых есть в базе.
12. Вывести список стран Африки, Северной и Южной Америки.

Практическая работа № 4

Типы данных и встроенные функции

1. Цель работы

1. Изучить основные типы данных.
2. Изучить встроенные функции для работы со строками.
3. Изучить встроенные функции для работы с числами.
4. Изучить встроенные функции для работы с датами и временем.
5. Изучить встроенные функции преобразования данных.
6. Изучить CASE и IIF.

2. Теоретическая часть

2.1. Типы данных

На языке Transact-SQL используется множество различных *типов данных*. Всех их можно разделить на следующие группы:

Числовые типы данных: BIT (значение 0 или 1), TINYINT (от 0 до 255), SMALLINT (от -32768 до 32767), INT (от -2147483648 до 2147483647), BIGINT (от -9223372036854775808 до 9223372036854775807), DECIMAL (числа с фиксированной точностью), NUMERIC: (аналогичен типу DECIMAL), SMALLMONEY (дробные значения от -214748.3648 до 214748.3647), MONEY (дробные значения от -922337203685477.5808 до 922337203685477.5807), FLOAT (от -1.79E+308 до 1.79E+308), REAL (числа от –340E+38 до 3.40E+38);

Типы данных, представляющие дату и время: DATE (дата от 01/01/0001 до 31/12/9999), TIME (время в диапазоне от 00:00:00.0000000 до 23:59:59.9999999), DATETIME (дата и время от 01/01/1753 до 31/12/9999), DATETIME2 (дата и время от 01/01/0001 00:00:00.0000000 до 31/12/9999 23:59:59.9999999), SMALLDATETIME (дата и время от 01/01/1900 до 06/06/2079), DATETIMEOFFSET (дата и время от 01/01/0001 до 31/12/9999);

Строковые типы данных: CHAR (фиксированная строка длиной от 1 до 8000 символов), VARCHAR (переменная строка длиной от 1 до 8000 символов), NCHAR (Unicode – фиксированная строка длиной от 1 до 4000 символов), NVARCHAR (Unicode – переменная строка длиной от 1 до 4000 символов), TEXT и NTEXT (устаревшие, не рекомендуется использовать);

Бинарные типы данных: BINARY (фиксированные бинарные данные от 1 до 8000 байт), VARBINARY (переменные бинарные данные от 1 до 8000 байт), IMAGE (устаревшая, не рекомендуется использовать);

Другие типы данных: UNIQUEIDENTIFIER (уникальный идентификатор GUID), TIMESTAMP (номер версии строки в таблице), CURSOR (набор строк таблицы),

HIERARCHYID (позиция в иерархии), SQL_VARIANT (данные любого типа), XML (документы или фрагменты XML), TABLE (таблица), GEOGRAPHY (географические данные, такие как широта и долгота), GEOMETRY (координаты на плоскости).

2.2. Встроенные функции Transact-SQL

Функции SQL производят действия с данными и возвращают результат. *Встроенные функции* делятся на три основные группы:

– *скалярные функции* – обрабатывают одиночное значение и возвращают одно значение. Их можно использовать везде, где допускается применение выражений.

– *агрегатные функции* – используются для получения обобщающих значений. Они, в отличие от скалярных функций, оперируют значениями столбцов множества строк;

– *функции для списка значений*.

Скалярные функции бывают следующих категорий:

– *строковые функции* – выполняют определенные действия над строками и возвращают строковые или числовые значения;

– *числовые функции* – возвращают числовые значения на основании заданных в аргументе значений того же типа;

– *функции времени и даты* – выполняют различные действия над входными значениями времени и даты и возвращают строковое, числовое значение или значение в формате даты и времени;

– *функции преобразования типа*.

Список часто используемых *строковых функций*:

LEN(строка)	возвращает количество символов в заданной строке
TRIM(строка) TRIM([символ FROM] строка)	удаляет символ пробела или другие заданные символы в начале и в конце строки.
LTRIM(строка)	удаляет начальные пробелы из заданной строки
RTRIM(строка)	удаляет конечные пробелы из заданной строки
CHARINDEX(подстрока, строка) CHARINDEX(подстрока, строка, начальная позиция)	возвращает индекс, по которому находится первое вхождение подстроки в строке.
PATINDEX('%шаблон%', строка)	возвращает индекс, по которому находится первое вхождение определенного шаблона в строке
LEFT(строка, число)	возвращает с начала строки определенное количество символов
RIGHT(строка, число)	возвращает с конца строки определенное количество символов
SUBSTRING(строка, начальная позиция, длина)	возвращает подстроку заданной длиной, начиная с данной позиции
REPLACE(строка, подстрока, замена)	заменяет одну подстроку другой
REVERSE(строка)	переворачивает строку наоборот

CONCAT(строка1, строка2 [, строкаN])	объединяет заданные строки в одну
LOWER(строка)	переводит строку в нижний регистр
UPPER (строка)	переводит строку в верхний регистр
SPACE(число)	возвращает заданное количество пробелов
REPLICATE(строка, число)	повторяет значение строки указанное число раз
STUFF(строка, начальная позиция, количество, замена)	удаляет указанное количество символов первой строки в начальной позиции и вставляет на их место замену.

Список часто используемых *числовых функций*:

ABS(число)	возвращает абсолютное значение числа
CEILING(число)	возвращает наименьшее целое, большее или равное заданного числа.
FLOOR(число)	возвращает наибольшее целое число, меньшее или равное заданного числа
POWER(число, степень)	возвращает значение указанного выражения, возведенное в заданную степень
RAND([начальное значение])	возвращает псевдослучайное значение от 0 до 1
ROUND(число, точность)	возвращает число, округленное до указанной точности
SIGN(число)	возвращает положительное (+1), нулевое (0) или отрицательное (-1) значение, обозначающее знак заданного выражения
SQRT(число)	возвращает квадратный корень данного числа
SQUARE(число)	возвращает квадрат указанного числа
PI()	возвращает константное значение π
ACOS(число)	возвращает угол в радианах, косинус которого задан – арккосинус.
ASIN(число)	возвращает угол в радианах, синус которого задан – арксинус.
ATAN(число)	возвращает угол в радианах, тангенс которого задан – арктангенс.
COS(число)	возвращает косинус указанного угла в радианах.
SIN(число)	возвращает синус указанного угла в радианах.
TAN(число)	возвращает тангенс указанного угла в радианах.
COT(число)	возвращает котангенс указанного угла в радианах
DEGREES(число)	возвращает для значения угла в радианах соответствующее значение в градусах.
RADIANS(число)	возвращает для значения угла в градусах соответствующее значение в радианах
EXP(число)	возвращает экспонент заданного числа
LOG(число)	возвращает натуральный логарифм указанного числа
LOG(число, основа)	возвращает логарифм указанного числа
LOG10(число)	возвращает десятичный логарифм указанного числа

Список часто используемых *функций времени и даты*:

GETDATE()	возвращает текущую дату и время
CURRENT_TIMEZONE()	возвращает имя часового пояса
GETUTCDATE()	возвращает текущую дату и время по Гринвичу (UTC/GMT)
DAY(дата)	возвращает день месяца указанной даты
MONTH(дата)	возвращает номер месяца указанной даты
YEAR(дата)	возвращает год указанной даты
DATEPART(часть, дата)	возвращает целое число, представляющее указанную часть заданной даты
DATENAME(часть, дата)	возвращает строку символов, представляющую указанную часть заданной даты
DATEADD(часть, число, дата)	добавляет указанное целое число со знаком к части входного значения даты, а затем возвращает это измененное значение
DATEDIFF(часть, начальная дата, конечная дата)	возвращает разницу как целое число со знаком между частями заданных дат
EOMONTH(дата)	возвращает последний день месяца, заданной даты

Для функций времени и даты используются следующие аргументы как часть даты и времени:

Часть даты и времени	***Сокращения***
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs
nanosecond	ns
tzoffset	tz
iso_week	isowk, isoww

Список часто используемых *функций преобразования*:

CAST(выражение AS тип)	преобразуют выражение в заданный тип
CONVERT(тип, выражение [, стиль])	
ASCII(строка)	возвращает код ASCII первого символа указанного символьного выражения
UNICODE(строка)	возвращает код Юникод первого символа указанного символьного выражения
CHAR(число)	возвращает символ ASCII с указанным кодом
NCHAR(число)	возвращает символ Юникода с указанным кодом
STR(число)	возвращает символьные данные, преобразованные из числовых данных

Список часто используемых *функций проверки значений*:

ISDATE(выражение)	возвращает 1, если выражение имеет допустимое значение типа даты и времени, иначе возвращает значение 0
ISNUMERIC(выражение)	возвращает 1, если выражение имеет допустимое значение числовой тип данных, иначе возвращает 0
ISNULL(выражение, замена)	заменяет значение NULL указанным замещающим значением
COALESCE(выражение[,...n])	вычисляет аргументы по порядку и возвращает текущее значение первого выражения, изначально не вычисленного как NULL.

Особое место среди встроенных скалярных функций языка SQL занимают функции вывода, которые являются разновидностью CASE-выражений. Функция CASE проверяет значение некоторого выражения, и в зависимости от результата проверки может возвращать тот или иной результат.

Выражение CASE имеет два формата:

– простое выражение CASE для определения результата сравнивает выражение с набором простых выражений;

– поисковое выражение CASE для определения результата вычисляет набор логических выражений.

Оба формата поддерживают дополнительный аргумент ELSE.

Функция IIF(условие, выражение_если_истина, выражение_если_ложь) – возвращает одно из двух значений в зависимости от того, принимает логическое выражение значение true или false.

3. Практическая часть

Дана таблица ***Академики:***

ФИО	Дата_рождения	Специализация	Год_присвое-ния_звания
Аничков Николай Николаевич	1885-11-03	медицина	1939
Бартольд Василий Владимирович	1869-11-15	историк	1913
Белопольский Аристарх Аполлонович	1854-07-13	астрофизик	1903
Бородин Иван Парфеньевич	1847-01-30	ботаник	1902
Вальден Павел Иванович	1863-07-26	химик-технолог	1910
Вернадский Владимир Иванович	1863-03-12	геохимик	1908
Виноградов Павел Гаврилович	1854-11-30	историк	1914
Ипатьев Владимир Николаевич	1867-11-21	химик	1916
Истрин Василий Михайлович	1865-02-22	филолог	1907
Карпинский Александр Петрович	1847-01-07	геолог	1889
Коковцов Павел Константинович	1861-07-01	историк	1906
Курнаков Николай Семёнович	1860-12-06	химик	1913
Марр Николай Яковлевич	1865-01-06	лингвист	1912
Насонов Николай Викторович	1855-02-26	зоолог	1906
Ольденбург Сергей Фёдорович	1863-09-26	историк	1903
Павлов Иван Петрович	1849-09-26	физиолог	1907
Перетц Владимир Николаевич	1870-01-31	филолог	1914
Соболевский Алексей Иванович	1857-01-07	лингвист	1900
Стеклов Владимир Андреевич	1864-01-09	математик	1912

Пример 1: Вывести ФИО академиков и длину ФИО:

```
SELECT
    ФИО
    ,LEN(ФИО) AS Количество_символов
FROM
    Академики
```

Пример 2: Вывести список академиков, убрать лишние пробелы в ФИО:

```
SELECT
    TRIM(ФИО) AS ФИО
    ,Дата_рождения
    ,Специализация
    ,Год_присвоения_звания
FROM
    Академики
```

Пример 3: Найти позиции буквы «о» в ФИО каждого академика. Вывести ФИО и позицию:

```
SELECT
        ФИО
        ,CHARINDEX('о',ФИО) AS Позиция_о
FROM
        Академики
```

Пример 4: Вывести ФИО и первые три буквы специализации каждого академика:

```
SELECT
        ФИО
        ,LEFT(Специализация, 3) AS Спец_3
FROM
        Академики
```

Пример 5: Вывести ФИО и от второй до пятой буквы специализации каждого академика:

```
SELECT
        ФИО
        ,SUBSTRING(Специализация, 2, 4) AS Спец_2_5
FROM
        Академики
```

Пример 6: Вывести список академиков, заменить специализацию «лингвист» на «языковед»:

```
SELECT
        ФИО
        ,Дата_рождения
        ,REPLACE(Специализация, 'лингвист', 'языковед') AS Спец
        ,Год_присвоения_звания
FROM
        Академики
```

Пример 7: Вывести список академиков, специализацию на верхнем регистре:

```
SELECT
        ФИО
        ,Дата_рождения
        ,UPPER(Специализация) AS Спец
        ,Год_присвоения_звания
FROM
        Академики
```

Пример 8: Вывести ФИО академиков в правильном и обратном виде:

```
SELECT
        ФИО
        ,REVERSE(ФИО) AS ФИО_Обр
FROM
        Академики
        Название
```

Пример 9: Вывести каждую специализацию 4 раза в одной строке. Убрать дубликаты:

```
SELECT DISTINCT
        REPLICATE(Специализация, 4) AS Спец_4
FROM
        Академики
```

Пример 10: Вывести абсолютное значение тригонометрических функций на точке π:

```
SELECT
        ABS(COS(PI()))  AS Косинус_Пи
        ,ABS(SIN(PI()))  AS Синус_Пи
        ,ABS(TAN(PI()))  AS Тангенс_Пи
        ,ABS(COT(PI()))  AS КоТангенс_Пи
```

Пример 11: Вывести число 132.456, округленное с точностью от 3 до -3:

```
SELECT
         ROUND(123.456, 3)   AS Окр3
        ,ROUND(123.456, 2)   AS Окр2
        ,ROUND(123.456, 1)   AS Окр1
        ,ROUND(123.456, 0)   AS Окр0
        ,ROUND(123.456, -1)   AS Окр_1
        ,ROUND(123.456, -2)   AS Окр_2
        ,ROUND(123.456, -3)   AS Окр_3
```

Пример 12: Вывести наименьшее целое число, которое больше или равно 123.456, и наибольшее целое число, которое меньше или равно 123.456:

```
SELECT
         CEILING(123.456)  AS Больше
        ,FLOOR(123.456)    AS Меньше
```

Пример 13: Вывести квадратный корень, квадрат и куб числа 25:

```
SELECT
         SQRT(25)     AS Корень
        ,SQUARE(25)  AS Квадрат
        ,POWER(25, 3) AS Куб
```

Пример 14: Вывести текущую дату и время:

```
SELECT
    GETDATE() AS Сейчас
```

Пример 15: Вывести день, месяц, год, час, минуту, секунду, номер квартала, номер недели, день года, день недели для текущей даты и времени:

```
SELECT
    DAY(GETDATE()) AS День
    ,MONTH(GETDATE()) AS Месяц
    ,YEAR(GETDATE()) AS Год
    ,DATEPART(HOUR, GETDATE()) AS Час
    ,DATEPART(MINUTE, GETDATE()) AS Минута
    ,DATEPART(SECOND, GETDATE()) AS Секунда
    ,DATEPART(QUARTER, GETDATE()) AS Квартал
    ,DATEPART(WEEK, GETDATE()) AS Неделя
    ,DATEPART(DAYOFYEAR, GETDATE()) AS День_года
    ,DATEPART(WEEKDAY, GETDATE()) AS День_недели
```

Пример 16: Вывести дату 100 дней назад от текущей:

```
SELECT
    DATEADD(DAY, -100, GETDATE()) AS День_100_Назад
```

Пример 17: Академик Игорь Евгеньевич Тамм родился 8 июля 1895 года. И.Е. Тамм скончался 12 апреля 1971 года. Вывести количество прожитых дней:

```
SELECT
    DATEDIFF(DAY, '18950708', '19710412') AS Количество_прожитых_дней
```

Пример 18: Вывести ФИО и время года рождения каждого академика:

```
SELECT
    ФИО
    , CASE MONTH(Дата_рождения)
        WHEN 3 THEN 'Весна'
        WHEN 4 THEN 'Весна'
        WHEN 5 THEN 'Весна'
        WHEN 6 THEN 'Лето'
        WHEN 7 THEN 'Лето'
        WHEN 8 THEN 'Лето'
        WHEN 9 THEN 'Осень'
        WHEN 10 THEN 'Осень'
        WHEN 11 THEN 'Осень'
        ELSE 'Зима'
        END AS Времени_года
FROM Академики
```

Пример 19: Вывести ФИО, дату рождения и знак зодиака каждого академика:

```
SELECT
        ФИО
        , Дата_рождения
        , CASE
                WHEN (MONTH(Дата_рождения)=3 AND DAY(Дата_рождения) >= 21)
OR (MONTH(Дата_рождения)=4  AND DAY(Дата_рождения) <= 20) THEN 'Овен'
                WHEN (MONTH(Дата_рождения)=4 AND DAY(Дата_рождения) >= 21)
OR (MONTH(Дата_рождения)=5  AND DAY(Дата_рождения) <= 21) THEN 'Телец'
                WHEN (MONTH(Дата_рождения)=5 AND DAY(Дата_рождения) >= 22)
OR (MONTH(Дата_рождения)=6  AND DAY(Дата_рождения) <= 21) THEN 'Близнецы'
                WHEN (MONTH(Дата_рождения)=6 AND DAY(Дата_рождения) >= 22)
OR (MONTH(Дата_рождения)=7  AND DAY(Дата_рождения) <= 22) THEN 'Рак'
                WHEN (MONTH(Дата_рождения)=7 AND DAY(Дата_рождения) >= 23)
OR (MONTH(Дата_рождения)=8  AND DAY(Дата_рождения) <= 21) THEN 'Лев'
                WHEN (MONTH(Дата_рождения)=8 AND DAY(Дата_рождения) >= 22)
OR (MONTH(Дата_рождения)=9  AND DAY(Дата_рождения) <= 23) THEN 'Дева'
                WHEN (MONTH(Дата_рождения)=9 AND DAY(Дата_рождения) >= 24)
OR (MONTH(Дата_рождения)=10 AND DAY(Дата_рождения) <= 23) THEN 'Весы'
                WHEN (MONTH(Дата_рождения)=10 AND DAY(Дата_рождения) >= 24)
OR (MONTH(Дата_рождения)=11 AND DAY(Дата_рождения) <= 22) THEN 'Скорпион'
                WHEN (MONTH(Дата_рождения)=11 AND DAY(Дата_рождения) >= 23)
OR (MONTH(Дата_рождения)=12 AND DAY(Дата_рождения) <= 22) THEN 'Стрелец'
                WHEN (MONTH(Дата_рождения)=12 AND DAY(Дата_рождения) >= 23)
OR (MONTH(Дата_рождения)=1  AND DAY(Дата_рождения) <= 20) THEN 'Козерог'
                WHEN (MONTH(Дата_рождения)=1 AND DAY(Дата_рождения) >= 21)
OR (MONTH(Дата_рождения)=2  AND DAY(Дата_рождения) <= 19) THEN 'Водолей'
                WHEN (MONTH(Дата_рождения)=2 AND DAY(Дата_рождения) >= 20)
OR (MONTH(Дата_рождения)=3  AND DAY(Дата_рождения) <= 20) THEN 'Рыбы'
        END AS Знак_зодиака
FROM Академики
```

Пример 20: Вывести список академиков. Для каждого академика, в зависимости от возраста, при присвоении звания вывести «молодой» или «старый» в дополнительном столбце:

```
SELECT
        ФИО
        ,Дата_рождения
        ,Специализация
        ,Год_присвоения_звания
        ,IIF(Год_присвоения_звания - Year(Дата_рождения) <= 45, 'Молодой','Старый')
AS Возраст_при_присвоении
FROM Академики
```

4. Задание

1. Вывести список академиков, отсортированный по количеству символов в ФИО.
2. Вывести список академиков, убрать лишние пробелы в ФИО.
3. Найти позиции «ов» в ФИО каждого академика. Вывести ФИО и номер позиции.
4. Вывести ФИО и последние две буквы специализации для каждого академика.
5. Вывести список академиков, ФИО в формате Фамилия и Инициалы.
6. Вывести список специализаций в правильном и обратном виде. Убрать дубликаты.
7. Вывести свою фамилию в одной строке столько раз, сколько вам лет.
8. Вывести абсолютное значение функций $sin^2\left(\frac{\pi}{2}\right) - cos\left(\frac{3\pi}{2}\right)$ с точностью два знака после десятичной запятой.
9. Вывести количество дней до конца семестра.
10. Вывести количество месяцев от вашего рождения.
11. Вывести ФИО и високосность года рождения каждого академика.
12. Вывести список специализаций без повторений. Для каждой специализации вывести «длинный» или «короткий», в зависимости от количества символов.

Практическая работа № 5

Агрегатные функции

1. Цель работы

1. Изучить основы агрегации данных.
2. Изучить функцию MAX.
3. Изучить функцию MIN.
4. Изучить функцию SUM.
5. Изучить функцию AVG.
6. Изучить функцию COUNT.
7. Изучить группировки данных.
8. Изучить применение фильтрации в группировке данных.

2. Теоретическая часть

Агрегатные функции используются для получения обобщающих значений. Они, в отличие от скалярных функций, оперируют значениями столбцов множества строк. К агрегатным функциям относятся:

– SUM – вычисляет итог;

– MAX – возвращает наибольшее значение;

– MIN – возвращает наименьшее значение;

– AVG – вычисляет среднее значение;

– COUNT – вычисляет количество значений в столбце.

Аргументами функций выступают поля таблицы или результаты выражений над ними. Вложенность не допускается.

Из агрегатных функций можно составлять любые выражения.

Для функций SUM и AVG столбец должен содержать числовые значения.

Для функций COUNT() можно указать аргумент * для подсчета всех строк без исключения.

По умолчанию вышеперечисленные пять функций учитывают все строки выборки для вычисления результата. Но выборка может содержать повторяющиеся значения. Если необходимо выполнить вычисления только над уникальными значениями, исключив из набора значений повторяющиеся данные, то для этого применяется оператор DISTINCT (кроме COUNT (*)). По умолчанию вместо DISTINCT применяется оператор ALL, который выбирает все строки. Так как этот оператор неявно подразумевается при отсутствии DISTINCT, то его можно не указывать.

Агрегатные функции можно применить не только на всю таблицу, но также на группу значений. Для этого применяется команда GROUP BY, которая пишется после WHERE. После команды GROUP BY перечисляется название столбцов, по которым следует группировать данные. Предложение GROUP BY указывает, что результаты запроса следует разделить на группы, применить агрегатную функцию по отдельности к каждой группе и получить для каждой группы одну строку результатов.

В качестве элемента группировки должен выступать любой возвращаемый элемент, указанный в предложении SELECT, кроме значений агрегатных функций.

Если столбец, по которому производится группировка, содержит значение NULL, то строки со значением NULL составят отдельную группу.

Команда HAVING <условие> применяется для фильтрации строк, возвращаемых при использовании предложения GROUP BY. HAVING пишется после GROUP BY, имеет такой формат, как WHERE, но в качестве значения используется значение, возвращаемое агрегатными функциями.

3. Практическая часть

Дана таблица ***Страны:***

Название	Столица	Площадь	Население	Континент
Австрия	Вена	83858	8741753	Европа
Азербайджан	Баку	86600	9705600	Азия
Албания	Тирана	28748	2866026	Европа
Алжир	Алжир	2381740	39813722	Африка
Ангола	Луанда	1246700	25831000	Африка
Аргентина	Буэнос-Айрес	2766890	43847000	Южная Америка
Афганистан	Кабул	647500	29822848	Азия
Бангладеш	Дакка	144000	160221000	Азия
Бахрейн	Манама	701	1397000	Азия
Белиз	Бельмопан	22966	377968	Северная Америка
Белоруссия	Минск	207595	9498400	Европа
Бельгия	Брюссель	30528	11250585	Европа
Бенин	Порто-Ново	112620	11167000	Африка
Болгария	София	110910	7153784	Европа
Боливия	Сукре	1098580	10985059	Южная Америка
Ботсвана	Габороне	600370	2209208	Африка
Бразилия	Бразилиа	8511965	206081432	Южная Америка
Буркина-Фасо	Уагадугу	274200	19034397	Африка
Бутан	Тхимпху	47000	784000	Азия
Великобритания	Лондон	244820	65341183	Европа
Венгрия	Будапешт	93030	9830485	Европа

Название	Столица	Площадь	Население	Континент
Венесуэла	Каракас	912050	31028637	Южная Америка
Восточный Тимор	Дили	14874	1167242	Азия
Вьетнам	Ханой	329560	91713300	Азия

Пример 1: Вывести максимальную площадь стран:

```
SELECT
    MAX(Площадь) AS Макс_площадь
FROM
    Страны
```

Пример 2: Вывести наименьшее население стран в Африке:

```
SELECT
    MIN(Население) AS Мин_население
FROM
    Страны
WHERE
    Континент = 'Африка'
```

Пример 3: Вывести суммарное население стран Северной и Южной Америки:

```
SELECT
    SUM(Население) AS Суммарное_население
FROM
    Страны
WHERE
    Континент = 'Северная Америка' OR
    Континент = 'Южная Америка'
```

Пример 4: Вывести среднее население стран, кроме европейских. Результат округлить до двух знаков:

```
SELECT
    ROUND(AVG(CAST(Население AS FLOAT)), 2) AS Среднее_население
FROM
    Страны
WHERE
    Континент != 'Европа'
```

Пример 5: Вывести количество стран, название которых начинается с буквы «С»:

```
SELECT
    COUNT(*) AS Количество
FROM
    Страны
```

```
WHERE
    LEFT(Название, 1) = 'C'
```

Пример 6: Вывести количество континентов, где есть страны:

```
SELECT
    COUNT(DISTINCT Континент) AS Количество_Континентов
FROM
    Страны
```

Пример 7: Вывести разницу населения между странами с наибольшим и наименьшим количеством граждан:

```
SELECT
    MAX(Население) - MIN(Население) AS Разница
FROM
    Страны
```

Пример 8: Вывести количество стран на каждом континенте. Результат отсортировать по количеству стран по убыванию:

```
SELECT
    Континент
, COUNT(Название) AS Количество_Стран
FROM
    Страны
GROUP BY
    Континент
ORDER BY
    Количество_Стран DESC
```

Пример 9: Вывести количество стран по первым буквам в названии. Результат отсортировать в алфавитном порядке:

```
SELECT
    LEFT(Название, 1) AS Первая_буква
    , COUNT(Название) AS Количество_Стран
FROM
    Страны
GROUP BY
    LEFT(Название, 1)
ORDER BY
    Первая_буква
```

Пример 10: Вывести список континентов, где плотность населения больше, чем 100 чел. на кв. км:

```
SELECT
        Континент
        , AVG(CAST(Население AS FLOAT) / Площадь) AS Сред_Плотность
FROM
        Страны
GROUP BY
        Континент
HAVING
        AVG(CAST(Население AS FLOAT) / Площадь) > 100
```

Пример 11: Ожидается, что через 25 лет население Европы и Азии вырастет на 20%, Северной Америки и Африки на 50%, а остальных частей мира – на 70%. Вывести список континентов с прогнозируемым населением:

```
SELECT
        Континент
        , CASE
        WHEN Континент IN ('Европа', 'Азия') THEN FLOOR(SUM(Население) * 1.2)

        WHEN Континент IN ('Северная Америка', 'Африка') THEN FLOOR(SUM(Население) * 1.5)
        ELSE FLOOR(SUM(Население) * 1.7)
        END AS Суммарное_Население
FROM
        Страны
GROUP BY
        Континент
```

Пример 12: Вывести список континентов, где разница по населению между наибольшими и наименьшими странами не более в 1000 раз:

```
SELECT
        Континент
FROM
        Страны
GROUP BY
        Континент
HAVING MAX(Население) <= 1000 * MIN(Население)
```

Пример 13: Вывести количество стран, у которых нет столицы (не введена в базу):

```
SELECT
    COUNT(*) AS Количество
FROM
    Страны
WHERE
    Столица IS NULL
```

Пример 14: Вывести количество символов в самых длинных и коротких названиях стран и столиц:

```
SELECT
    MAX(LEN(Название)) AS Дл_Название
    , MAX(LEN(Столица)) AS Дл_Столица
    , MIN(LEN(Название)) AS Кр_Название
    , MIN(LEN(Столица)) AS Кр_Столица
FROM
    Страны
```

Пример 15: Вывести список континентов, у которых средняя плотность среди стран с площадью более 1 млн. кв. км больше, чем 30 чел. на кв. км. Результат отсортировать по плотности по убыванию:

```
SELECT
    Континент
    ,AVG(CAST(Население AS FLOAT) / Площадь) AS Плотность
FROM
    Страны
WHERE
    Площадь > 1000000
GROUP BY
    Континент
HAVING
    AVG(CAST(Население AS FLOAT) / Площадь) > 30
ORDER BY
    Плотность DESC
```

4. Задание

1. Вывести минимальную площадь стран.
2. Вывести наибольшую по населению страну в Северной и Южной Америке.
3. Вывести среднее население стран. Результат округлить до одного знака.
4. Вывести количество стран, у которых название заканчивается на «ан», кроме стран, у которых название заканчивается на «стан».
5. Вывести количество континентов, где есть страны, название которых начинается с буквы «Р».
6. Сколько раз страна с наибольшей площадью больше, чем страна с наименьшей площадью?
7. Вывести количество стран с населением больше, чем 100 млн. чел. на каждом континенте. Результат отсортировать по количеству стран по возрастанию.
8. Вывести количество стран по количеству букв в названии. Результат отсортировать по убыванию.
9. Ожидается, что через 20 лет население мира вырастет на 10%. Вывести список континентов с прогнозируемым населением:
10. Вывести список континентов, где разница по площади между наибольшими и наименьшими странами не более в 10000 раз:
11. Вывести среднюю длину названий Африканских стран.
12. Вывести список континентов, у которых средняя плотность среди стран с населением более 1 млн. чел. больше, чем 30 чел. на кв. км.

Практическая работа № 6

Соединения таблиц

1. Цель работы

1. Изучить неявные соединения таблиц.
2. Изучить явные соединения таблиц.
3. Изучить внутреннее соединение таблиц.
4. Изучить внешнее соединение таблиц.
5. Изучить соединения таблиц со своей копией.

2. Теоретическая часть

Данные часто хранятся в несколько связанных таблицах. Для выбора данных используются разные методы соединения таблиц.

Когда выбор данных осуществляется из нескольких таблиц, в конструкции SELECT для каждого поля указывается таблица в виде <таблица>.<поле>. Если название поля уникальное, то можно его указать без таблицы, иначе это обязательно, чтобы избежать коллизий. Чтобы не повторить длинные названия таблиц, можно использовать псевдоним для таблиц. Псевдоним указывается в конструкции как FROM.

При *неявном соединении таблиц* формат конструкций FROM и WHERE имеет следующий вид:

FROM <таблица1> [псевдоним1], <таблица2> [псевдоним2]…

[WHERE <условие_соединения> [AND <условие_поиска>]…]

Таким образом, если более одной таблицы присутствует в конструкции FROM, то их разделяют запятой.

Если условие соединения не указывать, тогда результат будет декартовым произведением, то есть для каждой строки одной из таблиц берутся все возможные сочетания строк из других таблиц.

Для *явного соединения таблиц* используется команда JOIN. У явного соединения есть следующие разновидности:

– Внутреннее соединение – осуществляется с помощью команды INNER JOIN. Из двух таблиц берутся только связанные строки. INNER JOIN имеет следующий формат записи:

<таблица1> INNER JOIN <таблица2> ON <таблица1>.<связующее_поле> = <таблица2>.<связующее_поле>.

При внутреннем соединении слово INNER можно пропустить.

– Внешнее соединение – осуществляется с помощью команды OUTER JOIN. OUTER JOIN имеет следующий формат записи:

<таблица1> LEFT | RIGHT | FULL OUTER JOIN <таблица2> ON <таблица1>.<связующее_поле> = <таблица2>.<связующее_поле>.

У внешнего соединения есть три разновидности:

– Левое внешнее соединение LEFT OUTER JOIN – из таблицы, название которой является левым операндом команды JOIN, выбираются все строки, из второй таблицы – только те записи, которые имеют связь с первой таблицей.

– Правое внешнее соединение RIGHT OUTER JOIN – из таблицы, название которой является правым операндом команды JOIN, выбираются все строки, из первой таблицы только те записи, которые имеют связь со второй таблицей.

– Полное внешнее соединение FULL OUTER JOIN – из обоих таблиц выбираются все строки.

При использовании внешних соединений, слово OUTER можно пропустить.

– Перекрестное соединение CROSS JOIN – декартово произведение двух таблиц. CROSS JOIN имеет следующий формат записи:

<таблица1> CROSS JOIN <таблица2>.

При выборе данных из трех и более таблиц, с помощью явного соединения, результат зависит от порядка соединения.

Используя соединение, можно связать таблицу с собой. При таком соединении псевдоним обязателен.

Набор данных, полученных из нескольких таблиц, не отличается от набора, полученного из одной таблицы. Ему тоже можно применить группировки и т.д.

3. Практическая часть

Даны следующие таблицы:

Таблица 1. *Факультет*

Аббревиатура	Название
Ен	Естественные науки
Гн	Гуманитарные науки
Ит	Информационные технологии
Фм	Физико-математический

Таблица 2. *Кафедра*

Шифр	Название	Факультет
вм	Высшая математика	ен
ис	Информационные системы	ит
мм	Математическое моделирование	фм
оф	Общая физика	ен
пи	Прикладная информатика	ит
эф	Экспериментальная физика	фм

Таблица 3. *Сотрудник*

Таб_номер	Шифр	Фамилия	Должность	Зарплата	Шеф
101	пи	Прохоров П.П.	зав. кафедрой	35 000,00 р.	101
102	пи	Семенов С.С.	преподаватель	25 000,00 р.	101
105	пи	Петров П.П.	преподаватель	25 000,00 р.	101
153	пи	Сидорова С.С.	инженер	15 000,00 р.	102
201	ис	Андреев А.А.	зав. кафедрой	35 000,00 р.	201
202	ис	Борисов Б.Б.	преподаватель	25 000,00 р.	201
241	ис	Глухов Г.Г.	инженер	20 000,00 р.	201
242	ис	Чернов Ч.Ч.	инженер	15 000,00 р.	202
301	мм	Басов Б.Б.	зав. кафедрой	35 000,00 р.	301
302	мм	Сергеева С.С.	преподаватель	25 000,00 р.	301
401	оф	Волков В.В.	зав. кафедрой	35 000,00 р.	401
402	оф	Зайцев З.З.	преподаватель	25 000,00 р.	401
403	оф	Смирнов С.С.	преподаватель	15 000,00 р.	401
435	оф	Лисин Л.Л.	инженер	20 000,00 р.	402
501	вм	Кузнецов К.К.	зав. кафедрой	35 000,00 р.	501
502	вм	Романцев Р.Р.	преподаватель	25 000,00 р.	501
503	вм	Соловьев С.С.	преподаватель	25 000,00 р.	501
601	эф	Зверев З.З.	зав. кафедрой	35 000,00 р.	601
602	эф	Сорокина С.С.	преподаватель	25 000,00 р.	601
614	эф	Григорьев Г.Г.	инженер	20 000,00 р.	602

Таблица 4. *Специальность*

Номер	Направление	Шифр
01.03.04	Прикладная математика	мм
09.03.02	Информационные системы и технологии	ис
09.03.03	Прикладная информатика	пи
14.03.02	Ядерные физика и технологии	эф
38.03.05	Бизнес-информатика	ис

Таблица 5. *Дисциплина*

Код	Объем	Название	Исполнитель
101	320	Математика	вм
102	160	Информатика	пи
103	160	Физика	оф
202	120	Базы данных	ис
204	160	Электроника	эф
205	80	Программирование	пи
209	80	Моделирование	мм

Таблица 6. *Заявка*

Номер	01.03.04			09.03.02						09.03.03					14.03.02				38.03.05			
Код	101	205	209	101	102	103	202	205	209	101	102	103	202	205	101	102	103	204	101	103	202	209

Таблица 7. *Зав_кафедрой*

Таб_номер	101	201	301	401	501	601
Стаж	15	18	20	10	18	8

Таблица 8. *Инженер*

Таб_номер	153	241	242	435	614
Специальность	электроник	электроник	программист	электроник	программист

Таблица 9. *Преподаватель*

Таб_номер	**Звание**	**Степень**
101	профессор	д. т.н.
102	доцент	к. ф.-м. н.
105	доцент	к. т.н.
201	профессор	д. ф.-м. н.
202	доцент	к. ф.-м. н.
301	профессор	д. т.н.
302	доцент	к. т.н.
401	профессор	д. т.н.
402	доцент	к. т.н.
403	ассистент	–
501	профессор	д. ф.-м. н.
502	профессор	д. ф.-м. н.
503	доцент	к. ф.-м. н.
601	профессор	д. ф.-м. н.

Таблица 10. *Студент*

Рег_номер	Номер	Фамилия
10101	09.03.03	Николаева Н. Н.
10102	09.03.03	Иванов И. И.
10103	09.03.03	Крюков К. К.
20101	09.03.02	Андреев А. А.
20102	09.03.02	Федоров Ф. Ф.
30101	14.03.02	Бондаренко Б. Б.
30102	14.03.02	Цветков К. К.
30103	14.03.02	Петров П. П.
50101	01.03.04	Сергеев С. С.
50102	01.03.04	Кудрявцев К. К.
80101	38.03.05	Макаров М. М.
80102	38.03.05	Яковлев Я. Я.

Таблица 11. *Экзамен*

Дата	Код	Рег_номер	Таб_номер	Аудитория	Оценка
05.06.2015	102	10101	102	т505	4
05.06.2015	102	10102	102	т505	4
05.06.2015	202	20101	202	т506	4
05.06.2015	202	20102	202	т506	3
07.06.2015	102	30101	105	ф419	3
07.06.2015	102	30102	101	т506	4
07.06.2015	102	80101	102	м425	5
09.06.2015	205	80102	402	м424	4
09.06.2015	209	20101	302	ф333	3
10.06.2015	101	10101	501	т506	4
10.06.2015	101	10102	501	т506	4
10.06.2015	204	30102	601	ф349	5
10.06.2015	209	80101	301	э105	5
10.06.2015	209	80102	301	э105	4
12.06.2015	101	80101	502	с324	4
15.06.2015	101	30101	503	ф417	4
15.06.2015	101	50101	501	ф201	5
15.06.2015	101	50102	501	ф201	3
15.06.2015	103	10101	403	ф414	4
17.06.2015	102	10101	102	т505	5

Пример 1: Выбрать факультет и кафедры, используя неявное соединение. Результат отсортировать по алфавиту:

```
SELECT
    Ф.Название AS Факультет
    , К.Название AS Кафедра
FROM
    Факультет Ф, Кафедра К
WHERE
    Ф.Аббревиатура = К.Факультет
ORDER BY
    Факультет, Кафедра
```

Пример 2: Выбрать факультет и кафедры, используя явное соединение. Результат отсортировать по алфавиту:

```
SELECT
    Ф.Название AS Факультет
    , К.Название AS Кафедра
FROM
    Факультет Ф
        INNER JOIN Кафедра К ON Ф.Аббревиатура = К.Факультет
ORDER BY
    Факультет, Кафедра
```

Пример 3: Выбрать все факультеты и их кафедры, если существуют. Результат отсортировать по алфавиту:

```
SELECT
    Ф.Название AS Факультет
    , К.Название AS Кафедра
FROM
    Факультет Ф
        LEFT OUTER JOIN Кафедра К ON Ф.Аббревиатура = К.Факультет
ORDER BY
    Факультет, Кафедра
```

Пример 4: Вывести из таблиц «Кафедра», «Специальность» и «Студент» данные о студентах:

```
SELECT
    С.Фамилия
    , П.Направление
    , К.Название AS Кафедра
FROM
```

```
        Студент С
            INNER JOIN Специальность П ON С.Номер = П.Номер
            INNER JOIN Кафедра К ON П.Шифр = К.Шифр
```

Пример 5: Вывести для каждого сотрудника фамилию, должность, зарплату и фамилию его непосредственного руководителя:

```
SELECT
        С.Фамилия
        , С.Должность
        , С.Зарплата
        , П.Фамилия AS Руководитель
FROM
        Сотрудник С
            INNER JOIN Сотрудник П ON С.Шеф = П.Таб_номер
```

Пример 6: Вывести список студентов, сдавших хотя бы один экзамен. По правилам соединения, студенты, не сдававшие экзамены, в выборке представлены не будут:

```
SELECT
        С.Фамилия
FROM
        Студент С
            INNER JOIN Экзамен Э ON С.Рег_номер = Э.Рег_номер
GROUP BY
        С.Фамилия
```

Пример 7: Вывести из таблиц «Студент» и «Экзамен» учетные номера и фамилии студентов, а также количество сданных экзаменов и средний балл для каждого студента:

```
SELECT
        С.Фамилия
        , COUNT(Э.Оценка) AS [Количество экзаменов]
        , AVG(Э.Оценка) AS [Средний балл]
FROM
        Студент С
            INNER JOIN Экзамен Э ON С.Рег_номер = Э.Рег_номер
GROUP BY
        С.Фамилия
```

Пример 8: Вывести список заведующих кафедрами и их зарплаты, и стаж работы:

```
SELECT
        С.Фамилия
        , С.Зарплата
        , З.Стаж
FROM
```

```
        Сотрудник С
            INNER JOIN Зав_кафедрой З ON С.Таб_номер = З.Таб_номер
```

Пример 9: Вывести список кандидатов и докторов физико-математических наук:

```
SELECT
        С.Фамилия
        , П.Степень
FROM
        Сотрудник С
            INNER JOIN Преподаватель П ON С.Таб_номер = П.Таб_номер
WHERE
        П.Степень IN ('к.ф.-м.н.', 'д.ф.-м.н.')
```

Пример 10: Вывести название дисциплины, фамилию, должность и степень преподавателя, дату и место проведения экзаменов в хронологическом порядке:

```
SELECT DISTINCT
        Д.Название AS Дисциплина
        , С.Фамилия
        , С.Должность
        , П.Степень
        , Э.Дата
        , Э.Аудитория
FROM
        Экзамен Э
            INNER JOIN Дисциплина Д ON Э.Код = Д.Код
            INNER JOIN Сотрудник С ON Э.Таб_номер = С.Таб_номер
            INNER JOIN Преподаватель П ON Э.Таб_номер = П.Таб_номер
ORDER BY
        Э.Дата
```

Пример 11: Вывести фамилию преподавателей и количество их экзаменов:

```
SELECT
         С.Фамилия
        , COUNT(Э.Дата) AS [Количество экзаменов]
FROM
        Экзамен Э
            INNER JOIN Сотрудник С ON Э.Таб_номер = С.Таб_номер
GROUP BY
        С.Фамилия
```

Пример 12: Вывести список студентов, не сдавших ни одного экзамена:

```
SELECT
        С.Фамилия
FROM
        Студент С
                LEFT OUTER JOIN Экзамен Э ON С.Рег_номер = Э.Рег_номер
WHERE
        Э.Рег_номер IS NULL
```

4. Задание

1. Вывести из таблиц «Кафедра», «Специальность» и «Студент» данные о студентах, которые обучаются на данном факультете (например, «ит»).
2. Вывести из таблиц «Кафедра», «Специальность» и «Сотрудник» данные о выпускающих кафедрах (факультет, шифр, название, фамилию заведующего). Выпускающей считается та кафедра, на которую есть ссылки в таблице «Специальность».
3. Вывести в запросе для каждого сотрудника номер и фамилию его непосредственного руководителя. Для заведующих кафедрами поле руководителя оставить пустым.
4. Вывести список студентов, сдавших минимум два экзамена.
5. Вывести список инженеров с зарплатой, меньшей 20000 руб.
6. Вывести список студентов, сдавших экзамены в заданной аудитории.
7. Вывести из таблиц «Студент» и «Экзамен» учетные номера и фамилии студентов, а также количество сданных экзаменов и средний балл для каждого студента только для тех студентов, у которых средний балл не меньше заданного (например, 4).
8. Вывести список заведующих кафедрами и их зарплаты, и степень.
9. Вывести список профессоров.
10. Вывести название дисциплины, фамилию, должность и степень преподавателя, дату и место проведения экзаменов в хронологическом порядке в заданном интервале даты.
11. Вывести фамилию преподавателей, принявших более трех экзаменов.
12. Вывести список студентов, не сдавших ни одного экзамена в указанной дате.

Практическая работа № 7

Объединение результатов нескольких запросов

1. Цель работы

1. Изучить основы объединения результатов запросов.
2. Изучить UNION.
3. Изучить EXCEPT.
4. Изучить INTERSECT.

2. Теоретическая часть

Для объединения результатов двух или более запросов в одну таблицу используется команда UNION. Команда UNION объединяет вывод двух или более запросов в единый набор строк и столбцов и имеет вид:

Первый запрос

UNION [ALL]

Второй запрос

…

Для объединения результатов нескольких запросов с помощью UNION, они должны соответствовать следующим требованиям:

− содержать одинаковое количество столбцов;

− типы данных столбцов должны совпадать во всех запросах;

− в промежуточных запросах нельзя использовать сортировку ORDER BY.

Чтобы отсортировать результат объединения, в конце запроса добавляется ORDER BY. Названия столбцов в запросах могут отличаться. Поэтому в команде ORDER BY указывается название столбцов с первого запроса.

Если объединяемые наборы содержат в строках идентичные значения, то при объединении повторяющиеся строки удаляются.

Если необходимо при объединении сохранить повторяющиеся строки, то для этого используется параметр ALL.

Все запросы выполняются независимо друг от друга, а уже их вывод объединяется.

В объединяемых запросах можно использовать одну и ту же таблицу.

Чтобы найти разность двух выборок, то есть те строки, которые есть в первой выборке, но которых нет во второй, используется команда EXCEPT. Команда EXCEPT имеет следующий вид:

Первый запрос

EXCEPT

Второй запрос.

Возвращаются все различные значения, указанные слева от оператора EXCEPT. Эти значения возвращаются, если они отсутствуют в результатах выполнения правого запроса.

Требования к использованию команды EXCEPT такие же, как к команде UNION.

Если сравниваются значения столбцов с целью определения различных строк, два значения NULL считаются равными.

Команда INTERSECT позволяет найти общие строки в результатах двух запросов, то есть данный оператор выполняет операцию пересечения множеств. Команда INTERSECT имеет следующий вид:

Первый запрос

INTERSECT

Второй запрос.

Требования к использованию команды INTERSECT такие же, как к командам UNION и EXCEPT.

3. Практическая часть

Дана таблица ***Страны:***

Название	Столица	Площадь	Население	Континент
Австрия	Вена	83858	8741753	Европа
Азербайджан	Баку	86600	9705600	Азия
Албания	Тирана	28748	2866026	Европа
Алжир	Алжир	2381740	39813722	Африка
Ангола	Луанда	1246700	25831000	Африка
Аргентина	Буэнос-Айрес	2766890	43847000	Южная Америка
Афганистан	Кабул	647500	29822848	Азия
Бангладеш	Дакка	144000	160221000	Азия
Бахрейн	Манама	701	1397000	Азия
Белиз	Бельмопан	22966	377968	Северная Америка
Белоруссия	Минск	207595	9498400	Европа
Бельгия	Брюссель	30528	11250585	Европа
Бенин	Порто-Ново	112620	11167000	Африка
Болгария	София	110910	7153784	Европа
Боливия	Сукре	1098580	10985059	Южная Америка
Ботсвана	Габороне	600370	2209208	Африка
Бразилия	Бразилиа	8511965	206081432	Южная Америка
Буркина-Фасо	Уагадугу	274200	19034397	Африка
Бутан	Тхимпху	47000	784000	Азия
Великобритания	Лондон	244820	65341183	Европа
Венгрия	Будапешт	93030	9830485	Европа

Название	Столица	Площадь	Население	Континент
Венесуэла	Каракас	912050	31028637	Южная Америка
Восточный Тимор	Дили	14874	1167242	Азия
Вьетнам	Ханой	329560	91713300	Азия

Пример 1: Вывести объединенный результат выполнения запросов, которые выбирают страны с площадью больше 1 млн. кв. км и с населением больше 100 млн. чел.:

```
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Площадь > 1000000
UNION
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Население > 100000000
```

Пример 2: Вывести объединенный результат выполнения запросов, которые выбирают страны с площадью больше 1 млн. кв. км и с населением больше 100 млн. чел., при этом оставляет дубликаты:

```
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
```

```
        Площадь > 1000000
UNION ALL
SELECT
          Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Население > 100000000
```

Пример 3: Вывести объединенный результат выполнения запросов, которые выбирают европейские страны с плотностью более 300 чел. на кв. км, азиатские страны с плотностью более 200 чел. на кв. км. и африканские страны с плотностью более 150 чел. на кв. км. Результат отсортировать по континентам:

```
SELECT
          Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Континент = 'Европа' AND
        CAST(Население AS FLOAT) / Площадь > 400
UNION
SELECT
          Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Континент = 'Азия' AND
        CAST(Население AS FLOAT) / Площадь > 300
```

```
UNION
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Континент = 'Африка' AND
        CAST(Население AS FLOAT) / Площадь > 200
ORDER BY
        Континент
```

Пример 4: Вывести список стран с площадью больше 1 млн. кв. км, исключить страны с населением больше 10 млн. чел.:

```
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Площадь > 1000000
EXCEPT
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Население > 10000000
```

Пример 5: Вывести список стран с площадью больше 1 млн. кв. км и с населением больше 100 млн. чел.:

```
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Площадь > 1000000
INTERSECT
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Население > 100000000
```

4. Задание

1. Вывести объединенный результат выполнения запросов, которые выбирают страны с площадью меньше 500 кв. км и с площадью больше 5 млн. кв. км:
2. Вывести список стран с площадью больше 1 млн. кв. км, исключить страны с населением меньше 100 млн. чел.:
3. Вывести список стран с площадью меньше 500 кв. км и с населением меньше 100 тыс. чел.

Практическая работа № 8

Подзапросы

1. Цель работы

1. Изучить виды вложенных запросов.
2. Изучить некоррелирующие подзапросы.
3. Изучить коррелирующие подзапросы.
4. Изучить применение конструкции IN к подзапросам.
5. Изучить конструкцию ALL.
6. Изучить конструкцию ANY/SOME.
7. Изучить конструкцию EXISTS.

2. Теоретическая часть

В зависимости от контекста запрос SELECT может вернуть результат в одном из трех видов:

– таблица – запрос возвращает набор строк и столбцов;

– список значений – запрос возвращает значения только одного столбца, но, возможно, в нескольких строках;

– скалярное значение – запрос возвращает значение одного столбца в одной строке.

Результат запроса можно использовать в других запросах. Место использования запроса зависит от вида возвращаемого значения.

В предложении SELECT может использоваться только скалярный подзапрос, который возвращает одно значение.

Подзапросы пишутся в скобках.

Подзапросы, которые используются в предложении FROM, должны иметь название для всех столбцов.

Подзапросы, которые используются в предложении FROM, должны иметь псевдоним.

Подзапросы, которые используются в предложении WHERE для сравнения со столбцом, должны возвращать значения соответствующего типа.

Подзапросы, которые используются в предложении WHERE для сравнения со столбцом, должны быть вторым операндом оператора сравнения.

Каждый вложенный запрос, в свою очередь, может содержать один или более вложенный запрос. В инструкцию можно вложить любое количество запросов (в практике, до 32). Подзапросы выполняются, начиная с самого глубокого.

Подзапросы бывают коррелирующими и некоррелирующими. В некоррелирующих подзапросах команды выполняются один раз, то есть результат подзапроса не зависит от

строк, выбранных в основном запросе. Такой подзапрос выполняется один раз для всего внешнего запроса.

Также существуют коррелирующие подзапросы, результаты которых зависят от строк, выбранных в основном запросе. Коррелирующие подзапросы имеют связь с внешним запросом и выполняется столько раз, сколько строк в основном запросе.

Выполнение коррелирующих подзапросов сильно влияет на эффективность выполнения, поэтому их необходимо использовать только в крайних случаях.

Команду IN можно применить к результатам подзапросов, возвращающих список значений.

В предложении WHERE значение столбца можно сравнить со списками значений, возвращаемых подзапросом. Для этого используются операторы ALL и ANY|SOME.

При использовании оператора ALL условие в операции сравнения должно быть верно для всех значений, возвращаемых подзапросом.

При использовании оператора ANY|SOME условие в операции сравнения должно быть верно для всех значений, возвращаемых подзапросом.

Оператор EXISTS проверяет, возвращает ли подзапрос какое-либо значение. Здесь важны не данные, а их существование.

3. Практическая часть

Дана таблица ***Страны:***

Название	Столица	Площадь	Население	Континент
Австрия	Вена	83858	8741753	Европа
Азербайджан	Баку	86600	9705600	Азия
Албания	Тирана	28748	2866026	Европа
Алжир	Алжир	2381740	39813722	Африка
Ангола	Луанда	1246700	25831000	Африка
Аргентина	Буэнос-Айрес	2766890	43847000	Южная Америка
Афганистан	Кабул	647500	29822848	Азия
Бангладеш	Дакка	144000	160221000	Азия
Бахрейн	Манама	701	1397000	Азия
Белиз	Бельмопан	22966	377968	Северная Америка
Белоруссия	Минск	207595	9498400	Европа
Бельгия	Брюссель	30528	11250585	Европа
Бенин	Порто-Ново	112620	11167000	Африка
Болгария	София	110910	7153784	Европа
Боливия	Сукре	1098580	10985059	Южная Америка
Ботсвана	Габороне	600370	2209208	Африка
Бразилия	Бразилиа	8511965	206081432	Южная Америка
Буркина-Фасо	Уагадугу	274200	19034397	Африка
Бутан	Тхимпху	47000	784000	Азия

Название	Столица	Площадь	Население	Континент
Великобритания	Лондон	244820	65341183	Европа
Венгрия	Будапешт	93030	9830485	Европа
Венесуэла	Каракас	912050	31028637	Южная Америка
Восточный Тимор	Дили	14874	1167242	Азия
Вьетнам	Ханой	329560	91713300	Азия

Пример 1: Вывести список стран и процентное соотношение их населения к суммарному населению мира:

```
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
        ,ROUND(CAST(Население AS FLOAT) * 100 /
        (
            SELECT
                SUM(Население)
            FROM
                Страны
        ), 3) AS Процент
FROM
    Страны
ORDER BY
    Процент DESC
```

Пример 2: Вывести список стран мира, население которых больше, чем среднее население всех стран мира:

```
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
    Страны
WHERE
    Население > (
        SELECT
            AVG(Население)
        FROM
            Страны)
```

Пример 3: С помощью подзапроса вывести список африканских стран, население которых больше 50 млн. чел.:

```
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        (
        SELECT
            Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
        FROM
            Страны
        WHERE
            Континент = 'Африка') A
WHERE
        Население > 50000000
```

Пример 4: Вывести список стран и процентное соотношение их населения к суммарному населению к той части мира, где они находятся:

```
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
        ,ROUND(CAST(Население AS FLOAT) * 100 /
        (
            SELECT
                SUM(Население)
            FROM
                Страны Б
            WHERE
                А.Континент = Б.Континент
        ), 3) AS Процент
```

```
FROM
        Страны А
ORDER BY
        Процент DESC
```

Пример 5: Вывести список стран мира, население которых больше, чем среднее население стран в той части света, где они находятся:

```
SELECT
          Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны А
WHERE
        Население > (
                SELECT
                        AVG(Население)
                FROM
                        Страны Б
                WHERE
                        Б.Континент = А.Континент
                                )
```

Пример 6: Вывести список стран мира, которые находятся в тех частях света, среднее население которых больше, чем общемировое:

```
SELECT
        Название
   ,Столица
   ,Площадь
   ,Население
   ,Континент
FROM
        Страны
WHERE
        Континент IN (
                SELECT
                        Континент
                FROM
                        Страны
```

```
        GROUP BY
            Континент
        HAVING
            AVG(Население) > (
                SELECT
                    AVG(Население)
                FROM
                    Страны
                )
    )                                   )
```

Пример 7: Вывести список азиатских стран, население которых больше, чем в любой европейской стране:

```
SELECT
    Название
  ,Столица
  ,Площадь
  ,Население
  ,Континент
FROM
    Страны
WHERE
    Континент = 'Азия'
    AND
    Население > ALL (
        SELECT
            Население
        FROM
            Страны
        WHERE
            Континент = 'Европа'
    )
```

Пример 8: Вывести список европейских стран, население которых больше, чем население хотя бы одной южноамериканской страны:

```
SELECT
    Название
  ,Столица
  ,Площадь
  ,Население
```

```
    ,Континент
FROM
        Страны
WHERE
        Континент = 'Европа'
        AND
        Население > ANY (
                SELECT
                        Население
                FROM
                        Страны
                WHERE
                        Континент = 'Южная Америка'
        )
```

Пример 9: Если в Африке есть хотя бы одна страна, население которой больше 100 млн. чел., вывести список всех африканских стран:

```
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Континент = 'Африка'
        AND
        EXISTS (
                SELECT
                        *
                FROM
                        Страны
                WHERE
                        Континент = 'Африка'
                        AND
                        Население > 100000000
        )
```

Пример 10: Вывести список стран в той части света, где находится страна «Науру»:

```
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Континент = (
                SELECT
                        Континент
                FROM
                        Страны
                WHERE
                        Название = 'Науру'
        )
```

Пример 11: Вывести список стран, население которых не превышает населении страны «Гондурас»:

```
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Население !> (
                SELECT
                        Население
                FROM
                        Страны
                WHERE
                        Название = 'Гондурас'
        )
```

Пример 12: Вывести название страны с наибольшим населением среди стран с наименьшим населением на каждом континенте:

```
SELECT
        Название
    ,Столица
    ,Площадь
    ,Население
    ,Континент
FROM
        Страны
WHERE
        Население = (
                SELECT
                        MAX(Мин_Нас)
                FROM
                        (
                                SELECT
                                        MIN(Население) AS Мин_Нас
                                FROM
                                        Страны
                                GROUP BY
                                        Континент
                        ) A
        )
```

4. Задание

1. Вывести список стран и процентное соотношение площади каждой из них к общей площади всех стран мира.
2. Вывести список стран мира, плотность населения которых больше, чем средняя плотность населения всех стран мира.
3. С помощью подзапроса вывести список европейских стран, население которых меньше 5 млн. чел.
4. Вывести список стран и процентное соотношение их площади к суммарной площади той части мира, где они находятся.
5. Вывести список стран мира, площадь которых больше, чем средняя площадь стран той части света, где они находятся.
6. Вывести список стран мира, которые находятся в тех частях света, средняя плотность населения которых превышает общемировую.

7. Вывести список южноамериканских стран, в которых живет больше людей, чем в любой африканской стране.

8. Вывести список африканских стран, в которых живет больше людей, чем хотя бы в одной южноамериканской стране.

9. Если в Африке есть хотя бы одна страна, площадь которой больше 2 млн. кв. км, вывести список всех африканских стран.

10. Вывести список стран той части света, где находится страна «Фиджи».

11. Вывести список стран, население которых не превышает население страны «Фиджи».

12. Вывести название страны с наибольшим населением среди стран с наименьшей площадью на каждом континенте.

Практическая работа № 9

Основы DDL

1. Цель работы

1. Изучить создание базы данных.
2. Изучить удаление базы данных.
3. Изучить создание таблиц.
4. Изучить удаление таблиц.

2. Теоретическая часть

В стандарте ANSI нет команды для создания базы данных. Но в языке Transact-SQL существует команда CREATE DATABASE. Процедура создания базы данных обычно закрепляется только за администратором базы данных, и часто выполняется с использованием графического интерфейса.

Синтаксис команды создания базы данных имеет следующий вид: CREATE DATABASE <название_базы_данных>. В SQL Server можно создать до 32768 баз данных.

После создания базы данных, ее можно установить в качестве текущей с помощью команды USE: USE <название_базы_данных>.

Для удаления базы данных применяется команда DROP DATABASE, которая имеет следующий синтаксис: DROP DATABASE <название_базы_данных>. Перед удалением рекомендуется создать резервную копию базы данных.

Основные объекты базы данных – таблицы. Они содержат все данные в базе данных. В таблицах данные организованы в виде строк и столбцов. Каждая строка представляет собой уникальную запись, а каждый столбец – поле записи.

В базе данных можно создать до 2147483648 таблиц. Стандартная таблица может содержать до 1024 столбцов. Число строк и размер таблицы ограничиваются только пространством для хранения на сервере.

При создании таблицы можно установить отдельные свойства для каждого столбца. Данные в таблице могут быть сжаты либо по строкам, либо по страницам. Сжатие данных может позволить отображать больше строк на странице.

Название базы данных и таблиц может состоять максимум из 128 символов, а имена локальных временных таблиц – из 116 символов.

Таблицы можно создать с помощью конструктора или с помощью команд Transact-SQL. Для создания таблиц существует команда CREATE TABLE. Ее упрощённый синтаксис имеет следующий вид:

CREATE TABLE <название_таблицы>

(<название_столбца1> <тип_данных>,

…

<названиеСтолбцаN> <тип_данных>)

После типа данных можно добавить ограничения к значениям столбца.

Ограничение NULL | NOT NULL определяет, допустимы ли для столбца значения NULL.

Ограничение UNIQUE обеспечивает уникальность значения для указанного столбца. В таблице может быть несколько ограничений UNIQUE.

Ограничение PRIMARY KEY обеспечивает целостность и уникальность значения для указанного столбца с помощью уникального индекса. Можно создать только одно ограничение PRIMARY KEY для таблицы.

Ограничение CHECK обеспечивает целостность домена путем ограничения возможных значений, которые можно ввести в столбец. Условие может включать несколько логических выражений, соединенных операторами AND и OR.

Ограничение DEFAULT позволяет указать значение по умолчанию, если значение не задано. Оно может содержать значения констант, функции, или значение NULL. При этом, нельзя ссылаться на другой столбец таблицы, а также на другие таблицы, представления или хранимые процедуры. Его нельзя создавать для столбцов с типом данных timestamp или столбцов со свойством IDENTITY.

Свойство IDENTITY указывает, что новый столбец является столбцом идентификаторов. Для этого столбца формируется уникальное последовательное значение. Обычно используется вместе с ограничением PRIMARY KEY для поддержания уникальности идентификаторов строк в таблице. Свойство IDENTITY может назначаться столбцам типа tinyint, smallint, int, bigint, decimal(p,0) или numeric(p,0). Для таблицы можно создать только один столбец идентификаторов. Ограниченные значения по умолчанию и ограничения DEFAULT не могут использоваться в столбце идентификаторов. Необходимо указать как начальное значение, так и приращение, или же не указывать ничего. Если ничего не указано, применяется значение по умолчанию (1,1).

Для удаления таблиц используется команда DROP TABLE, которая имеет следующий синтаксис: DROP TABLE <название_таблицы>.

3. Практическая часть

Таблица ***Страны:***

Страна	**Столица**	**Часть света**	**Население тыс. чел.**	**Площадь тыс. кв. км**	**Тип управления**
Австрия	Вена	Европа	7513	84	4
Великобритания	Лондон	Европа	55928	244	1
Греция	Афины	Европа	9280	132	4

Страна	Столица	Часть света	Население тыс. чел.	Площадь тыс. кв. км	Тип управления
Афганистан	Кабул	Азия	20340	647	3
Монголия	Улан-Батор	Азия	1555	1565	4
Япония	Токио	Азия	114276	372	1
Франция	Париж	Европа	53183	551	3
Швеция	Стокгольм	Европа	8268	450	1
Египет	Каир	Африка	38740	1001	3
Сомали	Могадишо	Африка	3350	638	
США	Вашингтон	Америка	217700	9363	3
Мексика	Мехико	Америка	62500	1973	4
Мальта	Валлетта	Европа	330	0,3	4
Монако	Монако	Европа	25	0,2	1

Таблица ***Управление***:

ID	Вид
1	Конституционная монархия
2	Абсолютная монархия
3	Президентская республика
4	Парламентская республика
5	Военная хунта

Пример 1: Создать таблицу «Управление»:

```
CREATE TABLE Управление
(
        ID      INT ,
        Вид     VARCHAR(20)
)
```

Пример 2: Удалить таблицу «Управление»:

```
DROP TABLE Управление
```

Пример 3: Создать таблицу «Управление», значения столбца «ID» сделать уникальными, а столбец «Вид» запретить оставлять незаполненным:

```
CREATE TABLE Управление
(
        ID      INT UNIQUE,
        Вид     VARCHAR(20) NOT NULL
)
```

Пример 4: Создать таблицу «Управление», в столбец «ID» разрешить вводить значения меньше 200, а для столбца «Вид» установить значение по умолчанию «Президентская республика»:

```
CREATE TABLE Управление
(
        ID      INT CHECK (ID < 200),
        Вид     VARCHAR(20) DEFAULT 'Президентская республика'
)
```

Пример 5: Создать таблицу «Управление», столбец «ID» определить, как основной ключ, и настроить автоматический идентификатор с начальным значением 5 и с шагом 3:

```
CREATE TABLE Управление
(
        ID      INT PRIMARY KEY IDENTITY(5,3),
        Вид     VARCHAR(20)
)
```

4. Задание

1. Создать таблицу «Управление_ВашаФамилия». Определить основной ключ, идентификатор, значение по умолчанию
2. Создать таблицу «Страны_ВашаФамилия». Определить основной ключ, разрешение / запрет на NULL, условие на вводимое значение.
3. Создать таблицу «Цветы_ВашаФамилия». Определить основной ключ, значения столбца «ID» сделать уникальными, для столбца «Класс» установить значение по умолчанию «Двудольные».
4. Создать таблицу «Животные_ВашаФамилия». Определить основной ключ, значения столбца «ID» сделать уникальными, для столбца «Отряд» установить значение по умолчанию «Хищные».

Практическая работа № 10

Основы DML

1. Цель работы

1. Изучить команду INSERT.
2. Изучить команду UPDATE.
3. Изучить команду DELETE.
4. Изучить команду TRUNCATE TABLE.
5. Изучить конструкцию SELECT … INTO.

2. Теоретическая часть

Команда INSERT осуществляет добавление данных в определенную таблицу. После команды INSERT можно добавить необязательное ключевое слово INTO. Упрощенный синтаксис команды имеет следующий вид:

INSERT INTO <таблица>
[(<список столбцов>)]
VALUES
(<список значений>)

Если добавляется две и более строки, тогда используется следующий синтаксис:

INSERT INTO <таблица>
[(<список столбцов>)]
VALUES
(<список значений>),
…
(<список значений>)

Количество и тип значений должны совпадать со списком столбцов. Последовательность столбцов может не совпадать с таблицей. Список столбцов должен быть заключен в круглые скобки, а его элементы должны разделяться запятыми.

Если столбец имеет свойства идентификатор, его нельзя указать в списке. Для таких столбцов сервер автоматически вычисляет новое значение.

Если столбец имеет свойство DEFAULT, при отсутствии его, в таблицу вставляется значение по умолчанию.

Если столбец имеет свойство NULL, при отсутствии его, в таблицу вставляется значение NULL.

Если столбец имеет свойство NOT NULL, его обязательно надо включить в список.

В списке значений для каждого столбца из указанных в списке столбцов должно быть одно значение. Список значений должен быть заключен в скобки.

Если значения в списке идут в том же порядке, как в таблице, и для каждого столбца таблицы определено значение, то список столбцов можно не указывать.

Если одновременно добавляется несколько строк значений, каждый список значений заключается в круглые скобки и разделяется запятыми.

Если значение для столбца неизвестно, и столбец имеет свойство NULL, для него в списке значений можно указать NULL (без кавычек).

Если требуется перенести строку из одной таблицы в другую таблицу можно использовать следующий синтаксис:

```
INSERT INTO <таблица>
[(<список столбцов>) ]
SELECT
        [(<список столбцов>) ]
FROM
исходная_таблица
WHERE
<условие>
```

Типы данных в исходной и целевой таблицах должны совпадать.

Если таблицы имеют одинаковую структуру, можно после команды INSERT пропустить список столбцов, а после команды SELECT указать все столбцы, с помощью астериска «*».

Для изменения строк в таблице применяется команда UPDATE. Она имеет следующий формальный синтаксис:

```
UPDATE <таблица>
SET столбец1 = значение1, столбец2 = значение2, …, столбецN = значениеN
[WHERE <условие>]
```

Использование условий необязательно, но тогда обновляются все строки таблицы. Рекомендуется сначала выполнять выборку строк с помощью SELECT, только потом использовать команду UPDATE.

Для удаления одной или нескольких строк из таблицы применяется команда DELETE. Она имеет следующий формальный синтаксис:

```
DELETE [FROM] <таблица>
 [WHERE <условие>]
```

Ключевое слово FROM необязательно.

Использование условий необязательно, но тогда удаляются все строки таблицы. Рекомендуется сначала выполнять выборку строк с помощью SELECT, только потом использовать команду DELETE.

Для удаления всех строк из таблицы можно использовать команду TRUNCATE TABLE. Она имеет следующий формальный синтаксис:

```
TRUNCATE TABLE <таблица>
```

Инструкция TRUNCATE TABLE похожа на инструкцию DELETE без предложения WHERE, однако TRUNCATE TABLE выполняется быстрее и требует меньших ресурсов системы и журналов транзакций.

Для создания новой таблицы и ее заполнения можно использовать конструкцию SELECT…INTO. Она имеет следующий формальный синтаксис:

SELECT
<список столбцов>
INTO
<новая таблица>
FROM
<исходная таблица>

Столбцы в новой таблице создаются в порядке, соответствующем списку выбора и получают такие же имена, значения, типы данных и свойства допустимости значений NULL, которые указаны в соответствующем выражении в списке выбора.

3. Практическая часть

Таблица ***Ученики:***

ID	Фамилия	Предмет	Школа	Баллы
1	Иванова	Математика	Лицей	98,5
2	Петров	Физика	Лицей	99
3	Сидоров	Математика	Лицей	88
4	Полухина	Физика	Гимназия	78
5	Матвеева	Химия	Лицей	92
6	Касимов	Химия	Гимназия	68
7	Нурулин	Математика	Гимназия	81
8	Авдеев	Физика	Лицей	87
9	Никитина	Химия	Лицей	94
10	Барышева	Химия	Лицей	88

Код для создания данной таблицы:

```
CREATE TABLE Ученики
(
        ID          INT PRIMARY KEY IDENTITY(1,1),
        Фамилия     VARCHAR(50) NOT  NULL,
        Предмет     VARCHAR(50) NOT NULL,
        Школа       VARCHAR(50) NOT NULL,
        Баллы       FLOAT CHECK ((Баллы >= 0) AND (Баллы <= 100)) NULL
)
```

Пример 1: В таблицу «Ученики» внести новую запись для ученика гимназии Маркина, который по физике набрал 96 баллов:

```
INSERT INTO Ученики
        (Фамилия, Предмет, Школа, Баллы)
VALUES
        ('Маркин', 'Физика', 'Гимназия', 96)
```

Пример 2: В таблицу «Ученики» внести две строки, для ученицы лицея Никишиной, которая по химии набрала 77 баллов, и для ученика школы № 18 Андреева, оценка которого по математике неизвестна:

```
INSERT INTO Ученики
        (Фамилия, Предмет, Школа, Баллы)
VALUES
        ('Никишина', 'Химия', 'Лицей', 77),
        ('Андреев', 'Математика', 'Школа №18', NULL)
```

Пример 3: В таблице «Ученики» изменить данные Андреева, оценку исправить на 87:

```
UPDATE
        Ученики
SET
        Баллы = 87
WHERE
        Фамилия = 'Андреев'
```

Пример 4: В таблице «Ученики» изменить данные Никишиной, школу исправить на «Школа №31», а предмет на математику:

```
UPDATE
        Ученики
SET
        Школа = 'Школа №31', Предмет = 'Математика'
WHERE
        Фамилия = 'Никишина'
```

Пример 5: В таблице «Ученики» изменить данные всех учеников по математике, оценку уменьшить на 5 баллов:

```
UPDATE
        Ученики
SET
        Баллы = Баллы - 5
WHERE
        Предмет = 'Математика'
```

Пример 6: В таблице «Ученики» удалить данные всех учеников из школы №18:

```
DELETE FROM
        Ученики
WHERE
        Школа = 'Школа №18'
```

Пример 7: Создать таблицу «Лицеисты» и скопировать туда всех лицеистов:

```
SELECT
        ID
        ,Фамилия
        ,Предмет
        ,Школа
        ,Баллы
INTO
        Лицеисты
FROM
        Ученики
WHERE
        Школа = 'Лицей'
```

Пример 8: Очистить таблицу «Лицеисты»:

```
TRUNCATE TABLE Лицеисты
```

4. Задание

1. В таблицу «Ученики» внести новую запись для ученика школы № 18 Трошкова, оценка которого по химии неизвестна.
2. В таблицу «Ученики» внести три строки.
3. В таблице «Ученики» изменить данные Трошкова, школу исправить на № 21, предмет на математику, а оценку на 56.
4. В таблице «Ученики» изменить данные всех учеников по химии, оценку увеличить на 10%, если она ниже 60 баллов.
5. В таблице «Ученики» удалить данные всех учеников из школы №21.
6. Создать таблицу «Гимназисты» и скопировать туда данные всех гимназистов, кроме тех, которые набрали меньше 60 баллов.
7. Очистить таблицу «Гимназисты».

Практическая работа № 11

Программирование на SQL

1. Цель работы

1. Изучение переменных в T-SQL.
2. Изучение условных выражений.
3. Изучение циклов.

2. Теоретическая часть

Несмотря на то, что T-SQL – декларативный язык, у него есть расширение, позволяющее обрабатывать ошибки, создавать и выполнять хранимые процедуры и пользовательские функции, триггеры и сценарии с использованием локальных переменных, операторов присваивания, ветвлений и циклов.

Объявление переменной осуществляется с помощью оператора DECLARE. Упрощенный синтаксис команды имеет следующий вид:

DECLARE <@название> AS <тип>

Имена переменных в Transact-SQL начинаются с символа @.

Объявить сразу несколько переменных одним оператором DECLARE можно так:

DECLARE <@название1> AS <тип1>, …, <@названиеN> AS <типN>

Ключевое слово AS необязательно.

При объявлении переменной можно ее инициализировать:

DECLARE <@название> AS <тип> = <значение>

Объявленным переменным можно присвоить различные значения с помощью оператора присваивания SET. Переменным должны присваиваться значения того типа данных, с каким они были объявлены. Упрощенный синтаксис команды имеет следующий вид:

SET <@название> = <значение>

Переменным можно присваивать скалярный результат выполнения запросов:

SET <@название> = (SELECT <значение> FROM <таблица>)

Неинициализированные переменные имеют значение NULL, их нельзя использовать в выражениях.

Переменным можно присваивать значения с помощью команды SELECT:

SELECT <@переменная1> = <столбец1>, …, <@переменнаяN> = <столбецN> FROM <таблица>)

Значения переменных можно вывести с помощью команды PRINT. Синтаксис команды имеет следующий вид:

PRINT <сообщение>

Сообщение может быть символьной константой, переменной символьного типа, переменной, неявно преобразуемой в последовательность символов, или выражения, возвращающего символьный результат.

Значения переменных можно вывести с помощью команды SELECT. Синтаксис команды имеет следующий вид:

SELECT <@переменная1> [AS псевдоним1], …, <@переменнаяN> [AS псевдонимN]

Для выполнения команды в зависимости от условия используется управляющая команда IF ... ELSE … . Инструкция, следующая за ключевым словом IF и его условием, выполняется только в том случае, если логическое выражение возвращает TRUE. Необязательное ключевое слово ELSE представляет другую инструкцию, которая выполняется, если условие IF не удовлетворяется и логическое выражение возвращает FALSE. Упрощенный синтаксис команды имеет следующий вид:

IF <условие>
[BEGIN]
<команды>
[END]
[ELSE
[BEGIN]
<команды>
[END]
]

Условие должно возвращать только TRUE (ИСТИНА) или FALSE (ЛОЖЬ).

Если в блоке более чем одна команда, использование [BEGIN] … [END] обязательно.

Для выполнения повторяющихся операций применяется цикл WHILE. Упрощенный синтаксис команды имеет следующий вид:

WHILE <условие>
[BEGIN]
<команды| BREAK | CONTINUE >
[END]

Команда BREAK приводит к выходу из цикла и вызывает инструкции, следующие за ключевым словом END, обозначающим конец цикла.

Команда CONTINUE пропускает все команды после себя до конца цикла и переводит цикл на следующий шаг.

3. Практическая часть

Таблица ***Ученики:***

ID	Фамилия	Предмет	Школа	Баллы
1	Иванова	Математика	Лицей	98,5
2	Петров	Физика	Лицей	99
3	Сидоров	Математика	Лицей	88
4	Полухина	Физика	Гимназия	78
5	Матвеева	Химия	Лицей	92
6	Касимов	Химия	Гимназия	68
7	Нурулин	Математика	Гимназия	81
8	Авдеев	Физика	Лицей	87
9	Никитина	Химия	Лицей	94
10	Барышева	Химия	Лицей	88

Пример 1: Даны числа a и b. Найти и вывести их сумму:

```
DECLARE @a INT, @b INT, @c INT
SET @a = 5
SET @b = 10
SET @c = @a + @b
PRINT @c
```

Пример 2: В таблице «Ученики» найти разницу между наибольшими баллами среди лицеистов и гимназистов:

```
DECLARE @licey FLOAT, @gimn FLOAT, @diff FLOAT
SET @licey = (
                SELECT
                        MAX(Баллы)
                FROM
                        Ученики
                WHERE
                        Школа = 'Лицей'
                )
SET @gimn  = (
                SELECT
                        MAX(Баллы)
                FROM
                        Ученики
                WHERE
                        Школа = 'Гимназия'
                )
SET @diff = ABS(@licey - @gimn)
PRINT @diff
```

Пример 3: В таблице «Ученики» найти разницу между наибольшими и наименьшими баллами:

```
DECLARE @maxp FLOAT, @minp FLOAT, @diff FLOAT
SELECT
        @maxp = MAX(Баллы),
        @minp = MIN(Баллы)
FROM
        Ученики
SET @diff = @maxp - @minp
PRINT @diff
```

Пример 4: Дано случайное целое число меньше 1000. Вывести его квадрат:

```
DECLARE @a INT = RAND() * 1000, @b INT
SET @b = SQUARE(@a)
PRINT @b
```

Пример 5: Даны случайные целые числа a и b. Найти наибольшие из них:

```
DECLARE @a INT = RAND() * 100, @b INT = RAND() * 100
IF @a > @b
        PRINT '@a = ' + CAST(@a AS VARCHAR(3))
ELSE
        PRINT '@b = ' + CAST(@b AS VARCHAR(3))
```

Пример 6: Дано случайное целое число a. Проверить, делится ли данное число на 3:

```
DECLARE @a INT = RAND() * 100
IF @a % 3 = 0
        PRINT CAST(@a AS VARCHAR(3)) + ' делится на 3'
ELSE
        PRINT CAST(@a AS VARCHAR(3)) + ' не делится на 3'
```

Пример 7: Дано случайное целое число N (N < 1000). Если оно является степенью числа 5, то вывести «Да», если не является – вывести «Нет»:

```
DECLARE @a INT = RAND() * 1000
WHILE @a % 3 = 0
        SET @a = @a / 3
IF @a = 1
        PRINT 'Да'
ELSE
        PRINT 'Нет'
```

Пример 8: Даны случайные целые числа a и b. Найти наибольший общий делитель (НОД):

```
DECLARE @a INT = RAND() * 1000, @b INT = RAND() * 1000
PRINT '@a = ' + CAST(@a AS VARCHAR(4))
PRINT '@b = ' + CAST(@b AS VARCHAR(4))

WHILE @a != @b
BEGIN
	IF @a > @b
		SET @a = @a - @b
	ELSE
		SET @b = @b - @a
END
PRINT 'НОД = ' + CAST(@a AS VARCHAR(4))
```

Пример 9: Даны два целых числа A и B (A < B). Найти сумму всех целых чисел от A до B включительно:

```
DECLARE @a INT = 5, @b INT = 10, @s INT = 0
WHILE @a <= @b
BEGIN
	SET @s = @s + @a
	SET @a = @a + 1
END
PRINT 'Сумма = ' + CAST(@s AS VARCHAR(5))
```

Пример 10: Дано случайное целое число *N (N < 100)*. Найти квадрат данного числа, используя для его вычисления следующую формулу:

$$N^2 = 1 + 3 + 5 + \cdots + (2 \cdot N - 1)$$

После добавления к сумме каждого слагаемого выводить текущее значение суммы (в результате будут выведены квадраты всех целых чисел от 1 до *N*):

```
DECLARE @N INT = RAND() * 10, @M INT = 1, @S INT = 0
WHILE @M <= 2 * @N - 1
BEGIN
	SET @S = @S + @M
	PRINT @S
	SET @M = @M + 2
END
```

Пример 11: Даны случайные целые числа A и B (A < B). Вывести все целые числа от A до B включительно; при этом число A должно выводиться 1 раз, число A + 1 должно выводиться 2 раза и т.д.:

```
DECLARE @A INT = RAND() * 5, @C INT = 1
DECLARE @B INT = @A + RAND() * 5
PRINT '@A = ' + CAST(@A AS CHAR(1)) + ', @B = ' + CAST(@B AS CHAR(1))
WHILE @A <= @B
BEGIN
        PRINT REPLICATE(@A, @C)
        SET @A = @A + 1
        SET @C = @C + 1
END
```

Пример 12: Напечатать те из двузначных чисел, которые делятся на 4, но не делятся на 6:

```
DECLARE @A INT = 10
WHILE @A < 100
BEGIN
        IF (@A % 4 = 0) AND (@A % 6 != 0)
                PRINT @A
        SET @A = @A + 1
END
```

Пример 13: Даны два целых числа D (день) и M (месяц), определяющие правильную дату невисокосного года. Вывести значения D и M для даты, следующей за указанной:

```
DECLARE @D INT = 31, @M INT = 12
SET @D = CASE
                        WHEN @M IN (1, 3, 5, 7, 8, 10, 12) AND @D = 31 THEN 1
                        WHEN @M IN (4, 6, 9, 11) AND @D = 30 THEN 1
                        WHEN @M = 2 AND @D = 29 THEN 1
                        ELSE @D + 1
                END
SET @M = CASE
                        WHEN @D = 1 AND @M = 12 THEN 1
                        WHEN @D = 1 AND @M < 12 THEN @M + 1
                        ELSE @M
                END
PRINT CAST(@D AS VARCHAR(2)) + '/' + CAST(@M AS VARCHAR(2))
```

Пример 14: Вывести слово «Нижневартовск» на экран столько раз, сколько в нем букв:

```
DECLARE @L INT, @N CHAR(13) = 'Нижневартовск'
SET @L = LEN(@N)
```

```
WHILE @L > 0
BEGIN
        PRINT @N
        SET @L = @L - 1
END
```

Пример 15: Напишите код для вывода на экран с помощью цикла:

```
НижневартовскксвотравенжиН
Нижневартовс свотравенжиН
Нижневартов   вотравенжиН
Нижневарто    отравенжиН
Нижневарт     травенжиН
Нижневар      равенжиН
Нижнева       авенжиН
Нижнев        венжиН
Нижне         енжиН
Нижн          нжиН
Ниж           жиН
Ни            иН
Н             Н
Ни            иН
Ниж           жиН
Нижн          нжиН
Нижне         енжиН
Нижнев        венжиН
Нижнева       авенжиН
Нижневар      равенжиН
Нижневарт     травенжиН
Нижневарто    отравенжиН
Нижневартов   вотравенжиН
Нижневартовс свотравенжиН
НижневартовскксвотравенжиН
```

```
DECLARE @L INT, @M INT, @N CHAR(13)
SET @N = 'Нижневартовск'
SET @L = LEN(@N)
SET @M = @L
WHILE @L > 0
BEGIN
      PRINT LEFT(@N, @L) + SPACE(2 * (@M - @L)) + RIGHT(REVERSE(@N), @L)
      SET @L = @L - 1
```

```
END
SET @L = 2
WHILE @L <= @M
BEGIN
    PRINT LEFT(@N, @L) + SPACE(2 * (@M - @L)) + RIGHT(REVERSE(@N), @L)
    SET @L = @L + 1
END
```

4. Задание

1. Даны числа A и B. Найти и вывести их произведение.
2. В таблице «Ученики» найти разницу между средними баллами лицеистов и гимназистов.
3. В таблице «Ученики» проверить на четность количество строк.
4. Дано четырехзначное число. Вывести сумму его цифр.
5. Даны случайные целые числа a, b и c. Найти наименьшее из них.
6. Дано случайное целое число a. Проверить, делится ли данное число на 11.
7. Дано случайное целое число N (N < 1000). Если оно является степенью числа 3, то вывести «Да», если не является – вывести «Нет».
8. Даны случайные целые числа a и b. Найти наименьший общий кратный (НОК).
9. Даны два целых числа A и B (A<B). Найти сумму квадратов всех целых чисел от A до B включительно.
10. Найти первое натуральное число, которое при делении на 2, 3, 4, 5, и 6 дает остаток 1, но делится на 7.
11. Вывести свою фамилию на экран столько раз, сколько в нем букв.
12. Напишите код для вывода на экран с помощью цикла:

```
                    Н
                  иНи
               жиНиж
               нжиНижн
               енжиНижне
               венжиНижнев
               авенжиНижнева
             равенжиНижневар
            травенжиНижневарт
           отравенжиНижневарто
          вотравенжиНижневартов
         свотравенжиНижневартовс
        ксвотравенжиНижневартовск
```

Практическая работа № 12

Пользовательские функции

1. Цель работы

1. Изучение скалярных функций.
2. Изучение функции INLINE.
3. Изучение функции MULTI-STATEMENT.
4. Изучение удаления пользовательских функций.

2. Теоретическая часть

На языке Transact-SQL очень много встроенных функций. Несмотря на этот факт, иногда требуются функции, которых нет в стандартной библиотеке. Transact-SQL предоставляет возможность создать собственную функцию для решения задач. Пользовательская скалярная функция возвращает в качестве ответа единственное значение.

Пользовательскую функцию можно использовать следующими способами:

– В инструкциях Transact-SQL, например, SELECT.

– В приложениях, вызывающих функцию.

– В определении другой пользовательской функции.

– Для определения столбца таблицы.

– Для определения ограничения CHECK на столбец.

Пользовательские функции не могут выполнять действия, изменяющие состояние базы данных.

Пользовательские функции не могут возвращать несколько результирующих наборов.

Пользовательские функции не могут использовать динамический SQL и временные таблицы. Табличные переменные разрешены к использованию.

Пользовательские функции могут быть вложенными, то есть из одной функции может быть вызвана другая. Вложенность функций не может превышать 32 уровней.

Упрощенный синтаксис создания пользовательской скалярной функции имеет следующий вид:

```
CREATE FUNCTION <название>
(
[{<@параметр> [AS] <тип> [= <значение по умолчанию>]}]
)
RETURNS <тип возврата>
[AS]
BEGIN
<команды>
```

RETURN <значение>
END

Значение, переменная или выражение после ключевого слова RETURN имеет такой же тип, который указан после ключевого слова RETURNS.

Созданная функция может быть вызвана, как и обычная встроенная функция, но при этом должны вызываться с помощью имени владельца. Простой вызов имеет следующий вид:

SELECT <владелец>.<функция>(<параметры>)

Для пользовательских функций допускается не более 2100 параметров. При выполнении функции значение каждого из объявленных параметров должно быть указано пользователем, если для них не определены значения по умолчанию.

Имя параметра, как и имя переменных, использует знак @ как первый символ.

Для INLINE функций ключевого слова RETURNS указывается тип TABLE без указания списка столбцов. Тело такой функции представляет собой единственный оператор SELECT, который начинается сразу после ключевого слово RETURN. Упрощенный синтаксис создания пользовательской функции INLINE имеет следующий вид:

```
CREATE FUNCTION <название>
(
[{<@параметр> [AS] <тип> [= <значение по умолчанию>]}]
)
RETURNS TABLE
AS
RETURN
(
SELECT
<список столбцов>
FROM
<таблица>
WHERE
<условие>
)
```

В MULTI-STATEMENT функциях после ключевого слова RETURNS указывается тип TABLE с определением столбцов и их типов данных. Упрощенный синтаксис создания пользовательской MULTI-STATEMENT функции имеет следующий вид:

```
CREATE FUNCTION <название>
(
[{<@параметр> [AS] <тип> [= <значение по умолчанию>]}]
)
RETURNS <@таблица> TABLE (<определение таблицы>)
AS
```

BEGIN

<команды>

RETURN

END

Для MULTI-STATEMENT функций оператор RETURN не имеет аргумента. Значение возвращаемой переменной функции возвращается как значение функции.

Для удаления пользовательских функций используется команда DROP FUNCTION. Упрощенный синтаксис имеет следующий вид:

DROP FUNCTION [IF EXISTS] <название функции>

Ключевые слова IF EXISTS удаляют функцию только в том случае, если она уже существует.

3. Практическая часть

Дана таблица ***Страны:***

Название	Столица	Площадь	Население	Континент
Австрия	Вена	83858	8741753	Европа
Азербайджан	Баку	86600	9705600	Азия
Албания	Тирана	28748	2866026	Европа
Алжир	Алжир	2381740	39813722	Африка
Ангола	Луанда	1246700	25831000	Африка
Аргентина	Буэнос-Айрес	2766890	43847000	Южная Америка
Афганистан	Кабул	647500	29822848	Азия
Бангладеш	Дакка	144000	160221000	Азия
Бахрейн	Манама	701	1397000	Азия
Белиз	Бельмопан	22966	377968	Северная Америка
Белоруссия	Минск	207595	9498400	Европа
Бельгия	Брюссель	30528	11250585	Европа
Бенин	Порто-Ново	112620	11167000	Африка
Болгария	София	110910	7153784	Европа
Боливия	Сукре	1098580	10985059	Южная Америка
Ботсвана	Габороне	600370	2209208	Африка
Бразилия	Бразилиа	8511965	206081432	Южная Америка
Буркина-Фасо	Уагадугу	274200	19034397	Африка
Бутан	Тхимпху	47000	784000	Азия
Великобритания	Лондон	244820	65341183	Европа
Венгрия	Будапешт	93030	9830485	Европа
Венесуэла	Каракас	912050	31028637	Южная Америка
Восточный Тимор	Дили	14874	1167242	Азия
Вьетнам	Ханой	329560	91713300	Азия

Пример 1: Напишите функцию для вывода столицы данной страны, и вызовите ее:

```
CREATE FUNCTION Пример1
(
	@Страна AS VARCHAR(50)
)
RETURNS VARCHAR(50)
AS
BEGIN
	DECLARE @S AS VARCHAR(50)
	SELECT
		@S = Столица
	FROM
		Страны
	WHERE
		Название = @Страна
	RETURN @S
END

SELECT dbo.Пример1('Австрия')
```

Пример 2: Напишите функцию для перевода площади в тыс. кв. км., и вызовите ее:

```
CREATE FUNCTION Пример2
(
	@Площадь AS FLOAT
)
RETURNS FLOAT
AS
BEGIN
	DECLARE @P AS FLOAT
	SET @P = ROUND(@Площадь / 1000, 2)
	RETURN @P
END

SELECT
	Название,
	Столица,
	Континент,
	Население,
	dbo.Пример2(Площадь) AS [Площадь тыс.кв.км]
FROM
	Страны
```

Пример 3: Напишите функцию для вычисления плотности населения, и вызовите ее:

```
CREATE FUNCTION Пример3
(
	@Население AS INT,
	@Площадь AS FLOAT
)
RETURNS FLOAT
AS
BEGIN
	DECLARE @P AS FLOAT
	SET @P = ROUND(CAST(@Население AS FLOAT) / @Площадь, 2)
	RETURN @P
END

SELECT
	Название,
	Столица,
	Континент,
	Население,
	Площадь,
	dbo.Пример3(Население, Площадь) AS Плотность
FROM
	Страны
ORDER BY
	Плотность DESC
```

Пример 4: Напишите функцию для поиска страны второй по площади, и вызовите ее:

```
CREATE FUNCTION Пример4()
RETURNS VARCHAR(50)
AS
BEGIN
	DECLARE @P AS VARCHAR(50)
	DECLARE @M1 AS FLOAT
	DECLARE @M2 AS FLOAT

	SELECT
		@M1 = MAX(Площадь)
	FROM
		Страны

	SELECT
		@M2 = MAX(Площадь)
```

```
        FROM
            Страны
        WHERE
            Площадь < @M1

        SELECT
            @P = Название
        FROM
            Страны
        WHERE
            Площадь = @M2

        RETURN @P
END

SELECT
    dbo.Пример4() AS [Второй по площади страна]
```

Пример 5: Напишите функцию для поиска страны с минимальной площадью в заданной части света, и вызовите ее. Если часть света не указана, выбрать Европу:

```
CREATE FUNCTION Пример5
(
    @Конт AS VARCHAR(50) = 'Европа'
)
RETURNS VARCHAR(50)
AS
BEGIN
    DECLARE @P AS VARCHAR(50)
    DECLARE @M AS FLOAT

    SELECT
        @M = MIN(Площадь)
    FROM
        Страны
    WHERE
        Континент = @Конт

    SELECT
        @P = Название
    FROM
        Страны
    WHERE
```

```
            Континент = @Конт
            AND
            Площадь = @M

    RETURN @P
END

SELECT
    dbo.Пример5('Азия') AS [Наименьшая по площади страна в Азии]

SELECT
    dbo.Пример5(DEFAULT) AS [Наименьшая по площади страна в Европе]
```

Пример 6: Напишите функцию для замены букв в заданном слове от второй до предпоследней на точку, и примените ее для названия страны:

```
CREATE FUNCTION Пример6
(
    @A AS VARCHAR(50)
)
RETURNS VARCHAR(50)
AS
BEGIN
    RETURN LEFT(@A, 1) + REPLICATE('.', LEN(@A) - 2) + RIGHT(@A, 1)
END

SELECT
    dbo.Пример6(Название) AS [Скрытое название]
    ,Столица
    ,Континент
    ,Площадь
    ,Население
FROM
    Страны
```

Пример 7: Напишите функцию, которая возвращает количество стран, содержащих в названии заданную букву:

```
CREATE FUNCTION Пример7
(
    @C AS CHAR(1)
)
RETURNS INT
AS
```

```
BEGIN
	DECLARE @K AS INT

	SELECT
		@K = COUNT(*)
	FROM
		Страны
	WHERE
		CHARINDEX(@C, Название) > 0

	RETURN @K
END
```

Пример 8: Напишите функцию для вывода списка стран с населением больше заданного числа, и вызовите ее:

```
CREATE FUNCTION Пример8
(
	@N AS INT
)
RETURNS TABLE
AS
RETURN (
		SELECT
			Название
			,Столица
			,Площадь
			,Население
			,Континент
		FROM
			Страны
		WHERE
			Население > @N
			)
SELECT
	*
FROM
	dbo.Пример8(100000000)
```

Пример 9: Напишите функцию для вывода списка стран с площадью в интервале заданных значений, и вызовите ее:

```
CREATE FUNCTION Пример9
```

```
(
        @A AS FLOAT,
        @B AS FLOAT
)
RETURNS TABLE
AS
RETURN (
                SELECT
                        Название
                        ,Столица
                        ,Площадь
                        ,Население
                        ,Континент
                FROM
                        Страны
                WHERE
                        Площадь BETWEEN @A AND @B
                        )
SELECT
        *
FROM
        dbo.Пример9(1000, 10000)
```

Пример 10: Напишите функцию для возврата таблицы с названием страны и плотностью населения, и вызовите ее:

```
CREATE FUNCTION Пример10()
RETURNS @Ст_Плот TABLE
                        (
                                Название VARCHAR(50),
                                Плотность FLOAT
                        )
AS
BEGIN
        INSERT
                @Ст_Плот
        SELECT
                Название
                , CAST(Население AS FLOAT) / Площадь AS Плотность
        FROM
                Страны
        RETURN
END
```

```
SELECT
        Название
        ,Плотность
FROM
        dbo.Пример10()
```

Пример 11: Удалите функцию из примера 10:

```
DROP FUNCTION Пример10
```

4. Задание

1. Напишите функцию для вывода названия страны с заданной столицей, и вызовите ее.
2. Напишите функцию для перевода населения в млн. чел. и вызовите ее.
3. Напишите функцию для вычисления плотности населения заданной части света и вызовите ее.
4. Напишите функцию для поиска страны, третьей по населению и вызовите ее.
5. Напишите функцию для поиска страны с максимальным населением в заданной части света и вызовите ее. Если часть света не указана, выбрать Азию.
6. Напишите функцию для замены букв в заданном слове от третьей до предпоследней на “тест” и примените ее для столицы страны.
7. Напишите функцию, которая возвращает количество стран, не содержащих в названии заданную букву.
8. Напишите функцию для возврата списка стран с площадью меньше заданного числа и вызовите ее.
9. Напишите функцию для возврата списка стран с населением в интервале заданных значений и вызовите ее.
10. Напишите функцию для возврата таблицы с названием континента и суммарным населением и вызовите ее.
11. Напишите функцию IsPalindrom(P) целого типа, возвращающую 1, если целый параметр P (P > 0) является палиндромом, и 0 в противном случае.
12. Напишите функцию Quarter(x, y) целого типа, определяющую номер координатной четверти, содержащей точку с ненулевыми вещественными координатами (x, y).
13. Напишите функцию IsPrime(N) целого типа, возвращающую 1, если целый параметр N (N > 1) является простым числом, и 0 в противном случае.
14. Напишите код для удаления созданных вами функций.

Практическая работа № 13

Хранимые процедуры

1. Цель работы

1. Изучение создания хранимых процедур.
2. Изучение передачи входных параметров.
3. Изучение передачи выходных параметров.
4. Изучение вызовов хранимых процедур.
5. Изучение удаления хранимых процедур.

2. Теоретическая часть

При программировании в SQL Server введенный код сначала компилируется, потом запускается. Процесс компиляции может занимать определенное время. На языке Transact-SQL также есть возможность написанный блок кода сохранить и заранее скомпилировать. Особенно, если код многократно используется в операции базы данных, отличным решением будет произвести его инкапсуляцию в процедуры. Для этой цели используются хранимые процедуры, которые представляют собой набор инструкций, выполняющихся как единое целое. Процедуры аналогичны конструкциям в других языках программирования и выполняют следующие задачи:

– обрабатывают входные параметры и возвращают значения в виде выходных параметров;

– содержат инструкции, которые выполняют операции в базе данных, в отличии от пользовательских функций;

– возвращают сведения об успешном или неуспешном завершении.

В клиент-серверной и распределенных системах хранимые процедуры позволяют существенно сократить сетевой трафик, поскольку по сети отправляется только вызов на выполнение процедуры.

С точки зрения безопасности, хранимые процедуры выполняют очень большую роль, так как устраняют необходимость предоставлять разрешения на уровне объектов и упрощают формирование уровней безопасности. С помощью хранимых процедур можно предотвратить атаки типа «инъекция SQL».

Хранимая процедура создается с помощью команды CREATE PROCEDURE или CREATE PROC, которая имеет следующий упрощенный вид:

```
CREATE {PROC | PROCEDURE} <название>
[<@параметр> <тип> [= <значение по умолчанию>] [OUT | OUTPUT]]
AS
```

[BEGIN]
<команды>
[END]

При создании процедуры после команды CREATE указывается тип создаваемого объекта с помощью ключевого слова PROCEDURE или его сокращенного варианта PROC.

Названия процедур должны соответствовать требованиям, предъявляемым к идентификаторам, и должны быть уникальными в базе данных. При этом не следует пользоваться префиксом «sp_». Этим префиксом в SQL Server обозначаются системные процедуры.

В хранимую процедуру можно передать до 2100 параметров. При выполнении процедуры значение каждого из объявленных параметров должно быть указано пользователем, если для параметра не определено значение по умолчанию.

Ключевое слово OUT (можно использовать и OUTPUT) показывает, что параметр процедуры является выходным.

Для выполнения хранимой процедуры используется ключевое слово EXECUTE (или EXEC). Процедуру также можно вызывать и выполнять без ключевого слова, если она является первой инструкцией. Синтаксис команды EXECUTE имеет следующий вид:

EXECUTE [<@статус возврата>=] <название процедуры> [<@параметр>=] <значение>| <@переменная> [OUTPUT] | [DEFAULT]

В отличии от вызова функций, при вызове хранимых процедур с указанием названия параметра ([<@параметр>=] <значение>), последовательность параметров можно не соблюдать.

Для выходных параметров при вызове указывается ключевое слово OUTPUT.

Если для параметра указано значение по умолчанию, можно его использовать с помощью ключевого слова DEFAULT.

Для удаления хранимых процедур используется команда DROP PROCEDURE. Упрощенный синтаксис имеет следующий вид:

DROP PROC | PROCEDURE [IF EXISTS] <название хранимой процедуры>

Ключевые слова IF EXISTS удаляют хранимую процедуру только в том случае, если она уже существует.

3. Практическая часть

Дана таблица ***Страны:***

Название	Столица	Площадь	Население	Континент
Австрия	Вена	83858	8741753	Европа
Азербайджан	Баку	86600	9705600	Азия
Албания	Тирана	28748	2866026	Европа
Алжир	Алжир	2381740	39813722	Африка

Название	Столица	Площадь	Население	Континент
Ангола	Луанда	1246700	25831000	Африка
Аргентина	Буэнос-Айрес	2766890	43847000	Южная Америка
Афганистан	Кабул	647500	29822848	Азия
Бангладеш	Дакка	144000	160221000	Азия
Бахрейн	Манама	701	1397000	Азия
Белиз	Бельмопан	22966	377968	Северная Америка
Белоруссия	Минск	207595	9498400	Европа
Бельгия	Брюссель	30528	11250585	Европа
Бенин	Порто-Ново	112620	11167000	Африка
Болгария	София	110910	7153784	Европа
Боливия	Сукре	1098580	10985059	Южная Америка
Ботсвана	Габороне	600370	2209208	Африка
Бразилия	Бразилиа	8511965	206081432	Южная Америка
Буркина-Фасо	Уагадугу	274200	19034397	Африка
Бутан	Тхимпху	47000	784000	Азия
Великобритания	Лондон	244820	65341183	Европа
Венгрия	Будапешт	93030	9830485	Европа
Венесуэла	Каракас	912050	31028637	Южная Америка
Восточный Тимор	Дили	14874	1167242	Азия
Вьетнам	Ханой	329560	91713300	Азия

Пример 1: Напишите хранимую процедуру для вывода информации о сервере, о базе данных и о текущем пользователе, и вызовите ее:

```
CREATE PROC Пример1
AS
BEGIN
        SELECT
                @@Servername AS Сервер
                ,@@Version AS [Версия СУБД]
                ,Db_Name() AS [База данных]
                ,User AS [Пользователь базы данных]
                ,System_User AS [Системный пользователь]
END
EXECUTE Пример1
```

Пример 2: Напишите хранимую процедуру, которая выводит названия и столицы всех стран:

```
CREATE PROC Пример2
AS
BEGIN
```

```
        SELECT
                Название
                , Столица
        FROM
                Страны
END
```

Пример 3: Напишите хранимую процедуру, которая выводит список стран заданной части света, и вызовите ее:

```
CREATE PROC Пример3
        @Конт AS VARCHAR(50)
AS
BEGIN
        SELECT
                Название
                ,Столица
                ,Площадь
                ,Население
        FROM
                Страны
        WHERE
                Континент = @Конт
END
EXECUTE Пример3 'Азия'
```

Пример 4: Напишите хранимую процедуру, которая выводит список стран, площадь которых находится в заданном интервале, и вызовите ее:

```
CREATE PROC Пример4
        @A AS FLOAT,
        @B AS FLOAT
AS
BEGIN
        SELECT
                Название
                ,Столица
                ,Площадь
                ,Население
                ,Континент
        FROM
                Страны
```

```
        WHERE
            Площадь BETWEEN @A AND @B
END
EXECUTE Пример4 1000, 10000
```

Пример 5: Напишите хранимую процедуру, которая возвращает количество стран, содержащих в названии заданную букву, и вызовите ее:

```
CREATE PROC Пример5
    @Буква AS CHAR(1),
    @Количество AS INT OUTPUT
AS
BEGIN
    SELECT
        @Количество = COUNT(*)
    FROM
        Страны
    WHERE
        CHARINDEX(@Буква, Название) > 0
END

DECLARE @К AS INT
DECLARE @Б AS CHAR(1)
SET @Б = 'у'
EXECUTE Пример5 @Б, @К OUTPUT
SELECT
    @К AS [Количество стран]
```

Пример 6: Напишите хранимую процедуру для вывода трех стран с наименьшей площадью в заданной части света, и вызовите ее. Если часть света не указана, выбрать Европу:

```
CREATE PROC Пример6
    @Конт AS VARCHAR(50) = 'Европа'
AS
BEGIN
    SELECT TOP 3
        Название
        ,Столица
        ,Площадь
        ,Население
        ,Континент
    FROM
```

```
        Страны
    WHERE
        Континент = @Конт
    ORDER BY
        Площадь
END
EXECUTE Пример6 DEFAULT
```

Пример 7: Напишите хранимую процедуру, которая создает таблицу «Страны_У», и заполняет ее странами, названия которых начинаются на букву «У»:

```
CREATE PROC Пример7
AS
BEGIN
    SELECT
        Название
        ,Столица
        ,Площадь
        ,Население
        ,Континент
    INTO
        Страны_У
    FROM
        Страны
    WHERE
        LEFT(Название, 1) = 'У'
END

EXECUTE Пример7
```

Пример 8: Напишите хранимую процедуру, которая удаляет таблицу «Страны_У» и возвращает количество строк:

```
CREATE PROC Пример8
AS
BEGIN
    DECLARE @K AS INT

    SELECT
        @K = COUNT(*)
    FROM
        Страны_У
```

```
        DROP TABLE Страны_У
            RETURN @K
END

DECLARE @C AS INT
EXECUTE @C = Пример8
SELECT @C AS [Количество строк в удаленной таблице]
```

Пример 9: Напишите код, который удаляет хранимую процедуру «Пример8»:

```
DROP PROC Пример8
```

4. Задание

1. Напишите хранимую процедуру для вывода информации о сервере, о базе данных, о текущем пользователе, о текущем времени, и вызовите ее.
2. Напишите хранимую процедуру, которая выводит данные всех стран.
3. Напишите хранимую процедуру, которая выводит список стран, кроме заданной части света, и вызовите ее.
4. Напишите хранимую процедуру, которая выводит список стран, население которых находится в заданном интервале, и вызовите ее.
5. Напишите хранимую процедуру, которая возвращает количество стран, у которых в названии отсутствует заданная буква, и вызовите ее.
6. Напишите хранимую процедуру для вывода пяти стран с наибольшим населением в заданной части света, и вызовите ее. Если часть света не указана, выбрать Африку.
7. Напишите хранимую процедуру, которая создает таблицу «Страны_<первая буква вашей фамилии>», и заполняет ее странами, названия которых начинаются с первой буквой вашей фамилии.
8. Напишите хранимую процедуру, которая удаляет таблицу, которую вы создали в предыдущем задании и возвращает количество удаленных строк.
9. Напишите хранимую процедуру, принимающую число и возвращающую количество цифр в нем через параметр OUTPUT.
10. Напишите хранимую процедуру AddRightDigit, добавляющую к целому положительному числу K справа цифру D (D – входной параметр целого типа, лежащий в диапазоне [0..9], K – параметр целого типа, являющийся одновременно входным и выходным).
11. Напишите хранимую процедуру InvDigit, меняющую порядок следования цифр целого положительного числа K на обратный (K – параметр целого типа, являющийся одновременно входным и выходным).
12. Напишите хранимую процедуру Swap, меняющую содержимое переменных X и Y (X и Y – вещественные параметры, являющиеся одновременно входными и выходными).

13. Напишите хранимую процедуру SortInc, меняющую содержимое переменных A, B, C, таким образом, чтобы их значения оказались упорядоченными по возрастанию (A, B, C – вещественные параметры, являющиеся одновременно входными и выходными).

14. Напишите хранимую процедуру DigitCountSum, находящую количество C цифр целого положительного числа K, а также их сумму S (K – входной, C, S – выходные параметры целого типа).

15. Напишите код, который удаляет все хранимые процедуры, вами созданные.

Практическая работа № 14

Триггеры

1. Цель работы

1. Изучить создание триггеров.
2. Изучить триггеры после событий.
3. Изучить триггеры вместо событий.
4. Изучить виртуальные таблицы в триггерах.
5. Изучить приостановление триггеров.
6. Изучить удаление триггеров.

2. Теоретическая часть

Триггер – это вид хранимой процедуры, который вызывается автоматически при определенных событиях. Часто триггеры применяются для автоматической поддержки целостности и защиты БД.

В MS SQL Server существует *три вида триггеров*, которые отличаются по функциям и по синтаксису создания и изменения:

 Триггеры DML вызываются при выполнении команд INSERT, UPDATE или DELETE. Можно создать триггер, реагирующий на две или на все три команды.

 Триггеры DDL реагируют на события изменения структуры БД: создание, изменение или удаление отдельных объектов БД.

 Триггеры входа в систему запускаются при соединении пользователя с экземпляром сервера. Их можно применять для дополнительной проверки полномочий пользователей.

Триггеры DML можно вызвать после событий (FOR | AFTER), или вместо него (INSTEAD OF).

Триггер AFTER выполняется после успешного завершения вызвавшего его события. Можно определить несколько AFTER-триггеров для каждой операции.

Триггер INSTEAD OF вызывается вместо выполнения команд. Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF-триггер.

Упрощенный синтаксис создания триггера имеет следующий вид:

```
CREATE TRIGGER <название триггера> ON <название таблицы>
<FOR | AFTER | INSTEAD OF> <INSERT | UPDATE | DELETE>
AS
[BEGIN]
<команды>
[END]
```

Ключевое слово FOR или AFTER указывает, что триггер DML срабатывает только после успешного запуска всех операций в инструкции SQL, по которой срабатывает триггер.

Ключевое слово INSTEAD OF указывает, что триггер DML выполняется вместо инструкции SQL, по которой он срабатывает, то есть переопределяет действия запускающих инструкций.

В определении триггера ключевые слова INSERT | UPDATE | DELETE определяют инструкции изменения данных, при применении которых к таблице или представлению срабатывает триггер DML. Указание хотя бы одного варианта обязательно. В определении триггера разрешены любые сочетания вариантов в любом порядке.

Триггеры не вызываются рекурсивно.

Хотя инструкция TRUNCATE TABLE по сути аналогичная инструкции DELETE, она не активирует триггер.

Если триггер выполняется для события добавления данных (команды INSERT), в теле триггера доступна виртуальная таблица INSERTED, которая содержит список добавленных данных.

Если триггер выполняется для события удаления данных (команды DELETE), в теле триггера доступна виртуальная таблица DELETED, которая содержит список удаленных данных.

Если триггер выполняется для события изменения данных (команды UPDATE), в теле триггера доступны две виртуальные таблицы INSERTED и DELETED, которые содержат список новых и старых данных, соответственно.

Если при определенных обстоятельствах выполнение триггера нежелательно, то можно его отключить. Для этого используется команда DISABLE TRIGGER, его синтаксис:

DISABLE TRIGGER <название триггера> ON <название таблицы>

А когда триггер снова понадобится, его можно включить с помощью команды ENABLE TRIGGER, его синтаксис:

ENABLE TRIGGER <название триггера> ON <название таблицы>

Для удаления триггера используется команда DROP TRIGGER, его синтаксис:

DROP TRIGGER <название триггера>

3. Практическая часть

Дана таблица ***Ученики:***

ID	Фамилия	Предмет	Школа	Баллы
1	Иванова	Математика	Лицей	98,5
2	Петров	Физика	Лицей	99
3	Сидоров	Математика	Лицей	88
4	Полухина	Физика	Гимназия	78

ID	Фамилия	Предмет	Школа	Баллы
5	Матвеева	Химия	Лицей	92
6	Касимов	Химия	Гимназия	68
7	Нурулин	Математика	Гимназия	81
8	Авдеев	Физика	Лицей	87
9	Никитина	Химия	Лицей	94

Пример 1: Напишите триггер на добавление записи в таблицу «Ученики». Данный триггер, в случае успешного добавления данных, выводит «Запись добавлена»:

```
CREATE TRIGGER Пример1 ON Ученики
FOR INSERT
AS
BEGIN
        PRINT 'Запись добавлена'
END
```

Пример 2: Напишите триггер на удаление записи из таблицы «Ученики». Данный триггер, в случае успешного удаления данных, выводит «Запись удалена»:

```
CREATE TRIGGER Пример2 ON Ученики
AFTER DELETE
AS
BEGIN
        PRINT 'Запись удалена'
END
```

Пример 3: Напишите триггер на добавление, изменение и удаление данных для таблицы «Ученики». Данный триггер выводит «Таблица изменена»:

```
CREATE TRIGGER Пример3 ON Ученики
FOR INSERT, UPDATE, DELETE
AS
BEGIN
        PRINT 'Таблица изменена'
END
```

Пример 4: Напишите триггер на удаление записи из таблицы «Ученики». Данный триггер, при попытке удаления данных, выводит «Нельзя удалить данные»:

```
CREATE TRIGGER Пример4 ON Ученики
INSTEAD OF DELETE
AS
BEGIN
        PRINT 'Нельзя удалить данные'
END
```

Пример 5: Создать таблицу «Ученики_Архив», которая будет содержать все данные об удаленных учениках и даты их удаления. Написать триггер, который будет фиксировать в таблице «Ученики_Архив» данные ученика, удаленного из таблицы «Ученики»:

```
CREATE TABLE Ученики_Архив
        (
        ID INT NOT NULL,
        Фамилия VARCHAR(50) NULL,
        Предмет VARCHAR(50) NULL,
        Школа VARCHAR(50) NULL,
        Баллы FLOAT NULL,
        Удалено DATETIME NOT NULL
        )

CREATE TRIGGER Пример5 ON Ученики
FOR DELETE
AS
BEGIN
        INSERT
                Ученики_Архив
        SELECT
                ID,
                Фамилия,
                Предмет,
                Школа,
                Баллы,
                GETDATE() AS Удалено
        FROM
                DELETED
END
```

Пример 6: Напишите команды для приостановления и запуска триггера из примера 5:

```
DISABLE TRIGGER Пример5 ON Ученики

ENABLE TRIGGER Пример5 ON Ученики
```

Пример 7: Напишите команды для удаления триггера из примера 5:

```
DROP TRIGGER Пример5
```

4. Задание

1. Напишите триггер на изменение записи в таблице «Ученики». Данный триггер, в случае изменения данных, должен вывести «Запись изменена».
2. Напишите триггер на добавление и удаление записи из таблицы «Ученики». Данный триггер, в случае успешного добавления или удаления данных, должен вывести «Количество строк изменено».
3. Напишите триггер на добавление, изменение и удаление данных в таблице «Ученики». Данный триггер должен вывести «{Текущий пользователь} изменил таблицу. Время: {текущее время}».
4. Напишите триггер на изменение записи в таблице «Ученики». Данный триггер, при попытке изменения данных, должен вывести «Нельзя редактировать данные».
5. Создать таблицу «Ученики_{Ваша_фамилия}», которая будет содержать фамилии удаленных учеников и даты их удаления. Написать триггер, который будет фиксировать в таблице «Ученики_{Ваша_фамилия}» данные учеников при удалении из таблицы «Ученики», в том случае, если у них остались однофамильцы в таблице «Ученики».
6. Напишите команды для приостановления и запуска триггера из предыдущей задачи.
7. Напишите команды для удаления всех созданных вами триггеров.

Практическая работа № 15

Представления и табличные объекты

1. Цель работы

1. Изучить создание и удаление представлений.
2. Изучить табличные переменные.
3. Изучить временные таблицы.
4. Изучить производные таблицы.

2. Теоретическая часть

Представление – это виртуальная таблица, содержимое которой определяется запросом. Как и таблица, представление состоит из ряда именованных столбцов и строк данных.

Представления, как таблицы, могут иметь до 1024 столбцов.

Запрос для создания представления может обращаться не более чем к 256 таблицам.

Можно создавать представления на основе других представлений, при этом уровень вложенности не может быть больше 32-х.

Представление можно использовать как обычную таблицу.

Упрощенный синтаксис создания преставления имеет следующий вид:

CREATE VIEW <название> <список столбцов>
AS <запрос SELECT>

Запрос SELECT, используемый в определении представления, не может включать предложение ORDER BY, если только в списке выбора инструкции SELECT нет также предложения TOP.

Для удаления представления используется команда DROP VIEW, его синтаксис:

DROP VIEW <название>

В Transact-SQL есть специальный тип данных для хранения результирующего набора для обработки в будущем. Его используют в основном для временного хранения набора строк, возвращаемых как результирующий набор функций с табличным значением. Функции и переменные могут быть объявлены как табличные переменные. Табличные переменные могут использоваться в функциях, хранимых процедурах и пакетах. Для объявления табличных переменных используется следующий синтаксис:

DECLARE <@название переменной> TABLE (<объявление столбцов>)

Табличная переменная ведет себя как локальная переменная, она имеет точно определенную область применения.

Табличная переменная может быть применена в любом месте, где используется таблица или табличное выражение в инструкциях SELECT, INSERT, UPDATE и DELETE. Но табличную переменную нельзя использовать в инструкции SELECT … INTO …

Табличные переменные автоматически очищаются в конце функции, хранимой процедуры или пакета, в котором они были определены.

Операция присвоения между табличными переменными не поддерживается.

В MS SQL Server для хранения промежуточных данных можно использовать временные таблицы. Они по поведению не отличаются от базовых таблиц. Создание, удаление и обращение к ним аналогично к базовым.

Первый символ в названии временной таблицы должен быть знак решетки #. Для локальных временных таблиц используется один знак #. Локальные временные таблицы доступны в течение текущей сессии и удаляются, когда пользователь отсоединяется от сервера.

Для глобальных временных таблиц используются два знака ##. Глобальные временные таблицы доступны всем открытым сессиям базы данных и удаляются, когда все пользователи, ссылающиеся на таблицы, отсоединяются от сервера.

Временные таблицы хранятся в системной базе данных TEMPDB.

Для принудительного удаления временных таблиц используется команда DROP TABLE.

В MS SQL Server можно создать временно именованный результирующий набор, называемый обобщенным табличным выражением. Он формируется при выполнении простого запроса.

За обобщенным табличным выражением должны следовать одиночные инструкции SELECT, INSERT, UPDATE или DELETE, ссылающиеся на некоторые или на все столбцы.

Обобщенные табличные выражения хранятся в оперативной памяти и существуют только во время первого выполнения запроса, который представляет эту таблицу.

3. Практическая часть

Пример 1: Создайте представление, содержащее список стран, население которых меньше 1 млн. чел., а площадь больше 100 тыс. кв. км, и используйте его:

```
CREATE VIEW Пример1
AS
SELECT
        Название
        ,Столица
        ,Площадь
        ,Население
        ,Континент
FROM
        Страны
```

```
WHERE
        Население < 1000000
        AND
        Площадь > 100000

SELECT
        Название
        ,Столица
        ,Площадь
        ,Население
        ,Континент
FROM
        Пример1
```

Пример 2: Создайте представление, содержащее список континентов, суммарную площадь и суммарное население стран, которые находятся на каждом континенте и используйте его:

```
CREATE VIEW Пример2
(
        Континент
        ,Площадь
        ,Население
)
AS
SELECT
        Континент
        ,SUM(Площадь)
        ,SUM(Население)
FROM
        Страны
GROUP BY
        Континент

SELECT
        Континент
        ,Площадь
        ,Население
FROM
        Пример2
```

Пример 3: Создайте представление, содержащее фамилии преподавателей, должность, каждого преподавателя, звание, степень, место работы, зарплату и используйте его:

```
CREATE VIEW Пример3
(
	Фамилия
	,Должность
	,Звание
	,Степень
	,Кафедра
	,Зарплата
)
AS
SELECT
	Фамилия
	,Должность
	,Звание
	,Степень
	,Название
	,Зарплата
FROM
	Сотрудник С
		INNER JOIN Преподаватель П ON С.Таб_номер = П.Таб_номер
		INNER JOIN Кафедра К ON С.Шифр = К.Шифр

SELECT
	Фамилия
	,Должность
	,Звание
	,Степень
	,Кафедра
	,Зарплата
FROM
	Пример3
```

Пример 4: Создайте табличную переменную, содержащую три столбца («Номер недели», «Дата начала», «Дата конца»). Заполните ее для текущего года и используйте:

```
DECLARE @Пример4 TABLE
(
	[Номер недели] INT,
	[Дата начала] DATE,
	[Дата конца] DATE
)
```

```
DECLARE @T AS DATE, @N INT = 1
SET @T = CAST(YEAR(GETDATE()) AS CHAR(4)) + '0101'
WHILE DATEPART(WEEKDAY, @T) > 1
	SET @T = DATEADD(DAY, -1, @T)
PRINT DATEPART(WEEK, @T)
WHILE YEAR(@T) < YEAR(DATEADD(YEAR, 1, GETDATE()))
BEGIN
	INSERT
		@Пример4
	VALUES
		(@N, @T, DATEADD(DAY, 6, @T))

	SET @T = DATEADD(DAY, 7, @T)
	SET @N = @N + 1
END

SELECT
	[Номер недели]
	,[Дата начала]
	,[Дата конца]
FROM
	@Пример4
```

Пример 5: Создайте табличную переменную, содержащую список стран, площадь которых в 1000 раз меньше, чем средняя площадь стран в мире и используйте:

```
DECLARE @Пример5 TABLE
(
	Название VARCHAR(50),
	Столица VARCHAR(50),
	Площадь FLOAT,
	Население BIGINT,
	Континент VARCHAR(50)
)
INSERT INTO
	@Пример5
SELECT
	Название
	,Столица
	,Площадь
	,Население
	,Континент
```

```
FROM
        Страны
WHERE
        Площадь * 1000 < (
                        SELECT
                                AVG(Площадь)
                        FROM
                                Страны
                        )
SELECT
        Название
        ,Столица
        ,Площадь
        ,Население
        ,Континент
FROM
        @Пример5
```

Пример 6: Создайте локальную временную таблицу, имеющую три столбца («Название месяца», «Количество экзаменов», «Количество студентов»), заполните и используйте ее:

```
SELECT
        DATENAME(MONTH, Дата) AS [Название месяца]
        , COUNT(DISTINCT Код) AS [Количество экзаменов]
        , COUNT(DISTINCT Рег_номер) AS [Количество студентов]
INTO
        #Пример6
FROM
        Экзамен
GROUP BY
        DATENAME(MONTH, Дата)

SELECT * FROM #Пример6
```

Пример 7: Создайте глобальную временную таблицу, содержащую название стран и плотность их населения, заполните и используйте ее:

```
CREATE TABLE ##Пример7
(
        Название VARCHAR(50),
        Плотность FLOAT
)
INSERT INTO
        ##Пример7
                (Название, Плотность)
```

```
SELECT
    Название, ROUND(Население / Площадь, 0) AS Плотность
FROM
    Страны
SELECT * FROM ##Пример7
DROP TABLE #Пример6
```

Пример 8: *С* помощью обобщенных табличных выражений, напишите запрос для вывода списка сотрудников, чьи зарплаты меньше, чем средняя зарплата по кафедре, их зарплаты и название кафедры:

```
WITH СЗК AS
(
SELECT
    К.Название AS Кафедра
    ,К.Шифр
    ,AVG(Зарплата) AS [Средняя зарплата по кафедре]
FROM
    Сотрудник С
        INNER JOIN Кафедра К ON С.Шифр = К.Шифр
GROUP BY
    К.Название, К.Шифр
)
SELECT
    С.Фамилия
    , С.Зарплата
    , З.Кафедра
    , З.[Средняя зарплата по кафедре]
FROM
    Сотрудник С
    INNER JOIN СЗК З ON С.Шифр = З.Шифр
WHERE
    С.Зарплата < З.[Средняя зарплата по кафедре]
```

4. Задание

1. Создайте представление, содержащее список африканских стран, население которых больше 10 млн. чел., а площадь больше 500 тыс. кв. км, и используйте его.
2. Создайте представление, содержащее список континентов, среднюю площадь стран, которые находятся на нем, среднюю плотность населения, и используйте его.
3. Создайте представление, содержащее фамилии преподавателей, их должность, звание, степень, место работы, количество их экзаменов, и используйте его.
4. Создайте табличную переменную, содержащую три столбца («Номер месяца», «Название месяца», «Количество дней»), заполните ее для текущего года, и используйте ее.
5. Создайте табличную переменную, содержащую список стран, площадь которых в 100 раз меньше, чем средняя площадь стран на континенте, где они находятся, и используйте ее.
6. Создайте локальную временную таблицу, имеющую три столбца («Номер недели», «Количество экзаменов», «Количество студентов»), заполните и используйте ее.
7. Создайте глобальную временную таблицу, содержащую название континентов, наибольшую и наименьшую площадь стран на них, заполните и используйте ее.
8. С помощью обобщенных табличных выражений напишите запрос для вывода списка сотрудников, чьи зарплаты меньше, чем средняя зарплата по факультету, их зарплаты и название факультета.
9. Напишите команды для удаления всех созданных вами представлений.

Практическая работа № 16

Курсоры

1. Цель работы

1. Изучить использование курсоров.
2. Изучить прокрутку курсоров.
3. Изучить использование циклов в курсорах.
4. Изучить использование переменных в курсорах.

2. Теоретическая часть

Инструкции Transact-SQL выполняют операции над множествами, но интерактивным приложениям иногда требуется обрабатывать результаты построчно. Для выполнения команд над отдельной строкой предусмотрены *курсоры*. Открытие курсора в результирующем наборе делает возможной его построчную обработку. Можно присвоить курсор переменной или параметру с типом данных CURSOR.

Курсоры позволяют усовершенствовать обработку результатов:

- позиционируясь на отдельные строки результирующего набора;
- получая одну или несколько строк от текущей позиции в результирующем наборе;
- поддерживая изменение данных в строках в текущей позиции результирующего набора;
- поддерживая разные уровни видимости изменений, сделанных другими пользователями для данных, представленных в результирующем наборе;
- предоставляя инструкциям Transact-SQL в скриптах, хранимых процедурах и триггерах доступ к данным результирующего набора.

Обычно курсоры используются для выбора из базы данных некоторого подмножества хранимой в ней информации. В каждый момент времени прикладной программой может быть проверена одна строка курсора. Курсоры часто применяются в операторах SQL, встроенных в написанные на языках процедурного типа прикладные программы. Некоторые из них неявно создаются сервером базы данных, в то время как другие определяются программистами.

В соответствии со стандартом SQL при работе с курсорами можно выделить следующие основные действия:

- создание или объявление курсора;
- открытие курсора, т.е. наполнение его данными, которые сохраняются в многоуровневой памяти;
- выборка из курсора и изменение с его помощью строк данных;

 закрытие курсора, после чего он становится недоступным для пользовательских программ;

 освобождение курсора, т.е. удаление курсора как объекта, поскольку его закрытие необязательно освобождает ассоциированную с ним память.

После освобождения курсора ассоциированная с ним память также освобождается. При этом становится возможным повторное использование его имени.

В некоторых случаях применение курсора неизбежно. Однако по возможности этого следует избегать и работать со стандартными командами обработки данных: SELECT, UPDATE, INSERT, DELETE. Помимо того, что курсоры не позволяют проводить операции изменения над всем объемом данных, скорость выполнения операций обработки данных посредством курсора заметно ниже, чем у стандартных средств Transact-SQL.

SQL Server поддерживает *четыре типа курсоров*:

Однонаправленный курсор – указывается как FORWARD_ONLY и не поддерживает прокрутку. Он также называется курсором FIREHOSE и поддерживает только получение строк последовательно, от начала до конца курсора. Строки нельзя получить из базы данных, пока они не будут выбраны. Результаты всех инструкций INSERT, UPDATE и DELETE, влияющих на строки результирующего набора (выполненных текущим пользователем или зафиксированных другими пользователями), отображаются как строки, выбранные из курсора.

Статический курсор – полный результирующий набор статического курсора создается в базе данных TEMPDB при открытии курсора. Статический курсор всегда отображает результирующий набор точно в том виде, в котором он был при открытии курсора. Статическими курсорами обнаруживаются лишь некоторые изменения или не обнаруживаются вовсе, но при этом в процессе прокрутки такие курсоры потребляют сравнительно мало ресурсов. Статические курсоры всегда доступны только для чтения.

KEYSET курсор – членство и порядок строк в курсоре, управляемом набором ключей, являются фиксированными при открытии курсора. Такие курсоры управляются с помощью набора уникальных идентификаторов – ключей. Ключи создаются из набора столбцов, который уникально идентифицирует строки результирующего набора.

Динамический курсор – это противоположность статических курсоров. Динамические курсоры отражают все изменения строк в результирующем наборе при прокрутке курсора. Значения типа данных, порядок и членство строк в результирующем наборе могут меняться для каждой выборки. Все инструкции UPDATE, INSERT и DELETE, выполняемые пользователями, видимы посредством курсора.

Упрощенный шаблон использования курсора имеет следующий синтаксис:

```
DECLARE <имя_курсора> CURSOR [FORWARD_ONLY | SCROLL]
FOR <SELECT_оператор>
OPEN <имя_курсора>
FETCH <NEXT|PRIOR|FIRST|LAST|ABSOLUTE <число>| RELATIVE <число>>
FROM <имя_курсора> INTO <@переменная>
CLOSE <имя_курсора>
DEALLOCATE <имя_курсора>
```

Инструкция DEALLOCATE удаляет связь между курсором и переменной, и освобождает структуры данных, составляющие курсор.

После создания курсора, чтобы его использовать, надо открыть курсор с помощью команды OPEN. А команда CLOSE закрывает открытый курсор, высвобождая текущий результирующий набор и снимая блокировки курсоров для строк.

Для получения определенной строки используется команда FETCH.

Ключевое слово NEXT возвращает следующую строку и перемещает указатель текущей строки на возвращенную строку. Если инструкция FETCH NEXT выполняет первую выборку в отношении курсора, она возвращает первую строку в результирующем наборе. NEXT является параметром по умолчанию выборки из курсора.

Ключевое слово PRIOR возвращает предыдущую строку и перемещает указатель текущей строки на возвращенную строку. Если инструкция FETCH PRIOR выполняет первую выборку из курсора, не возвращается никакая строка и положение курсора остается перед первой строкой.

Ключевое слово FIRST возвращает первую строку в курсоре и делает ее текущей.

Ключевое слово LAST возвращает последнюю строку в курсоре, делая ее текущей.

Ключевое слово ABSOLUTE с аргументом – положительным целым числом, возвращает строку с указанным номером от начала курсора, и делает ее текущей строкой. Если число отрицательное, возвращает строку с указанным номером от конца курсора, делая ее текущей строкой.

Ключевое слово RELATIVE с аргументом – положительным целым числом возвращает строку с указанным номером после текущей строки и делает ее текущей строкой. Если число отрицательное, возвращает строку с указанным номером до текущей строки и делает ее текущей строкой. Если число равно 0, возвращает текущую строку.

Ключевое слово INTO позволяет поместить данные из столбцов выборки в локальные переменные. Каждая переменная из списка, слева направо, связывается с соответствующим столбцом в результирующем наборе курсора. Типы данных переменных должны соответствовать типам данных соответствующего столбца результирующего набора. Количество переменных и столбцов тоже должны совпадать.

Для обработки результирующего набора построчно можно использовать инструкцию FETCH NEXT в цикле WHILE. Как условие в цикле используется функция @@FETCH_STATUS, которая возвращает состояние последней инструкции FETCH, вызванной в любом курсоре, открытом в рамках этого подключения.

Функция @@FETCH_STATUS возвращает одно из четырех значений:

0 Инструкция FETCH была выполнена успешно.

-1 Выполнение инструкции FETCH завершилось неудачно или строка оказалась вне пределов результирующего набора.

-2 Выбранная строка отсутствует.

-9 Курсор не выполняет операцию выборки.

Упрощенный синтаксис использования функции @@FETCH_STATUS имеет следующий вид:

```
FETCH NEXT FROM <имя_курсора>
WHILE @@FETCH_STATUS = 0
BEGIN
FETCH NEXT FROM <имя_курсора>
END
```

3. Практическая часть

Таблица ***Ученики:***

ID	Фамилия	Предмет	Школа	Баллы
1	Иванова	Математика	Лицей	98,5
2	Петров	Физика	Лицей	99
3	Сидоров	Математика	Лицей	88
4	Полухина	Физика	Гимназия	78
5	Матвеева	Химия	Лицей	92
6	Касимов	Химия	Гимназия	68
7	Нурулин	Математика	Гимназия	81
8	Авдеев	Физика	Лицей	87
9	Никитина	Химия	Лицей	94
10	Барышева	Химия	Лицей	88

Пример 1: Создайте курсор, содержащий отсортированные по алфавиту фамилии учеников и названия их предметов, откройте его, выведите первую строку, закройте и освободите курсор:

```
DECLARE MyCursor CURSOR
FOR
        SELECT
                Фамилия
                ,Предмет
        FROM
                Ученики
        ORDER BY
                Фамилия

OPEN MyCursor

FETCH MyCursor
```

```
CLOSE MyCursor
DEALLOCATE MyCursor
```

Пример 2: Создайте курсор с прокруткой, содержащий список учеников, откройте его, выведите пятую, предыдущую, с конца четвертую, шесть позиций назад находящуюся, четыре позиций вперед находящуюся, следующую, первую, последнюю строку, закройте и освободите курсор:

```
DECLARE MyCursor CURSOR SCROLL
FOR
        SELECT
                ID
                ,Фамилия
                ,Предмет
                ,Школа
                ,Баллы
        FROM
                Ученики

OPEN MyCursor

FETCH ABSOLUTE 5 FROM MyCursor
FETCH PRIOR FROM  MyCursor
FETCH ABSOLUTE -4 FROM MyCursor
FETCH RELATIVE -6 FROM MyCursor
FETCH RELATIVE 4 FROM MyCursor
FETCH NEXT FROM MyCursor
FETCH FIRST FROM MyCursor
FETCH LAST FROM MyCursor

CLOSE MyCursor
DEALLOCATE MyCursor
```

Пример 3: С помощью курсора, вычислите среднее арифметическое значение балла у учеников с наибольшим и наименьшим баллом:

```
DECLARE MyCursor CURSOR SCROLL
FOR
        SELECT
                Баллы
        FROM
                Ученики
        ORDER BY
                Баллы
```

```
DECLARE @S FLOAT = 0, @B FLOAT
OPEN MyCursor

FETCH FIRST FROM MyCursor INTO @B
SET @S = @S + @B
FETCH LAST FROM MyCursor INTO @B
SET @S = @S + @B

SET @S = @S / 2
PRINT @S

CLOSE MyCursor
DEALLOCATE MyCursor
```

Пример 4: С помощью курсора, сгенерируйте строку вида «Ученики <список фамилий и названий школ, разделенных запятыми> участвовали в олимпиаде»:

```
DECLARE MyCursor CURSOR SCROLL
FOR
	SELECT
		Фамилия
		,Школа
	FROM
		Ученики

DECLARE @S VARCHAR(2000), @F VARCHAR(50), @W VARCHAR(50)
OPEN MyCursor

SET @S = 'Ученики'
FETCH NEXT FROM MyCursor INTO @F, @W
WHILE @@FETCH_STATUS = 0
BEGIN
	SET @S = @S + ', ' + @F + ' из школы "' + @W + '"'
	FETCH NEXT FROM MyCursor INTO @F, @W
END

SET @S = @S + ' участвовали на олимпиаде.'
PRINT @S

CLOSE MyCursor
DEALLOCATE MyCursor
```

Пример 5: Создайте курсор, содержащий список учеников, с его помощью выведите учеников с четной позицией:

```
DECLARE MyCursor CURSOR SCROLL
FOR
        SELECT
                ID
                ,Фамилия
                ,Предмет
                ,Школа
                ,Баллы
        FROM
                Ученики

OPEN MyCursor

FETCH ABSOLUTE 2 FROM MyCursor
WHILE @@FETCH_STATUS = 0
BEGIN
        FETCH RELATIVE 2 FROM MyCursor
END

CLOSE MyCursor
DEALLOCATE MyCursor
```

Пример 6: Создайте курсор, содержащий отсортированный по убыванию баллов список учеников, откройте его, для каждого ученика выведите фамилию, предмет, школу, баллы и разницу баллов с предыдущим учеником:

```
DECLARE MyCursor CURSOR SCROLL
FOR
        SELECT
                Фамилия
                ,Предмет
                ,Школа
                ,Баллы
        FROM
                Ученики

DECLARE @F VARCHAR(50)
DECLARE @P VARCHAR(50)
DECLARE @S VARCHAR(50)
DECLARE @B FLOAT
DECLARE @OB FLOAT = 0
```

```
OPEN MyCursor

FETCH NEXT FROM MyCursor INTO @F, @P, @S, @B
WHILE @@FETCH_STATUS = 0
BEGIN
        SELECT
                @F AS Фамилия
                ,@P AS Предмет
                ,@S AS Школа
                ,@B AS Баллы
                ,ABS(@B - @OB) AS Разница
        SET @OB = @B
        FETCH NEXT FROM MyCursor INTO @F, @P, @S, @B
END

CLOSE MyCursor
DEALLOCATE MyCursor
```

4. Задание

1. Создайте курсор, содержащий отсортированные по баллам фамилии и баллы учеников, откройте его, выведите первую строку, закройте и освободите курсор.
2. Создайте курсор с прокруткой, содержащий список учеников, откройте его, выведите пятую, предыдущую, с конца четвертую, следующую, первую строку, закройте и освободите курсор.
3. Создайте курсор с прокруткой, содержащий список учеников, откройте его, выведите последнюю, шесть позиций назад находящуюся, четыре позиций вперед находящуюся строку, закройте и освободите курсор.
4. С помощью курсора, вычислите сумму баллов у учеников с наибольшим и наименьшим баллом.
5. С помощью курсора, сгенерируйте строку вида «Ученики <список фамилий и названий предметов, разделенных запятыми> участвовали в олимпиаде».
6. Создайте курсор, содержащий список учеников, с его помощью выведите учеников с нечетной позицией.
7. Создайте курсор, содержащий отсортированный по убыванию баллов список учеников, откройте его, для каждого ученика выведите фамилию, предмет, школу, баллы и процентное соотношение баллов с предыдущим учеником.

Практическая работа № 17

Оконные функции

1. Цель работы

1. Изучить оконные функции.
2. Изучить аналитические функции.
3. Изучить ранжирующие функции.
4. Изучить функции смещения.

2. Теоретическая часть

Оконные функции используются для вычисления в заданной секции. Оконные функции не приводят к группированию строк в одну строку вывода, строки сохраняют свои отдельные идентификаторы, а агрегированное значение добавляется к каждой строке.

Для определения секций используется инструкция OVER. Упрощенный синтаксис имеет следующий вид:

<функция> <столбец для вычислений> OVER ([PARTITION BY <столбец для группировки>] [ORDER BY <столбец для сортировки>])

Предложение PARTITION BY разделяет результирующий набор запроса на секции. Оконная функция применяется к каждой секции отдельно, и вычисление начинается заново для каждой секции.

Предложение ORDER BY определяет сортировку строк в каждой секции результирующего набора.

Оконные функции разделяются на *агрегирующие, ранжирующие* и *функции смещения*.

Агрегирующие функции MAX(), MIN(), SUM, AVG(), COUNT() можно применить к секциям результирующего набора запроса. Синтаксис функций имеет следующий вид:

MAX|MIN|SUM|AVG|COUNT(<столбец>) OVER([PARTITION BY <столбцы>] [ORDER BY <столбцы>])

В агрегирующих функциях предложение PARTITION BY можно не указывать, тогда функция обрабатывает все строки результирующего набора запроса как одну группу. Предложение ORDER BY определяет последовательность, в которой строкам назначаются уникальные номера. Его также можно не указать.

Ранжирующие функции возвращают ранжирующее значение для каждой строки в секции и являются недетерминированными.

В ранжирующих функциях предложение PARTITION BY можно не указывать, тогда функция обрабатывает все строки результирующего набора запроса как одну группу. Предложение ORDER BY определяет последовательность, в которой строкам назначаются уникальные номера. Оно должно указываться обязательно.

Transact-SQL содержит следующие ранжирующие функции:

ROW_NUMBER – нумерует выходные данные результирующего набора, то есть, возвращает последовательный номер строки, начиная с 1, в секции результирующего набора. Синтаксис функции имеет следующий вид:

ROW_NUMBER( ) OVER([PARTITION BY <столбцы>] ORDER BY <столбцы>)

RANK – возвращает ранг каждой строки в секции результирующего набора. Ранг строки вычисляется как единица плюс количество рангов, находящихся до этой строки. В отличии от функции ROW_NUMBER, RANK назначает одинаковое значение строкам, претендующим на один ранг. Синтаксис функции имеет следующий вид:

RANK( ) OVER([PARTITION BY <столбцы>] ORDER BY <столбцы>)

DENSE_RANK – возвращает ранг каждой строки в секции результирующего набора без промежутков в значениях ранжирования. Ранг определенной строки равен количеству различных значений рангов, предшествующих строке, увеличенному на единицу. Синтаксис функции имеет следующий вид:

DENSE_RANK( ) OVER([PARTITION BY <столбцы>] ORDER BY <столбцы>)

NTILE – распределяет строки упорядоченной секции в заданное количество групп. Группы нумеруются, начиная с единицы. Для каждой строки функция NTILE возвращает номер группы, которой принадлежит строка. Синтаксис функции имеет следующий вид:

NTILE(<количество групп>) OVER([PARTITION BY <столбцы>] ORDER BY <столбцы>)

Функции смещения возвращают значение из другой строки секции результирующего набора запроса. Предложение ORDER BY должно указываться обязательно.

Transact-SQL содержит следующие функции смещения:

Функции LAG() и LEAD() – обращаются к данным из предыдущей или последующей строки того же результирующего набора. Синтаксис функций имеет следующий вид:

LAG|LEAD(<столбец> [,<смещение>] [,<значение по умолчанию>]) OVER([PARTITION BY <столбцы>] ORDER BY <столбцы>)

Параметр «смещение» указывает количество строк до строки перед или после текущей строки, из которой необходимо получить значение. Если значение аргумента не указано, то по умолчанию принимается 1.

«Значение по умолчанию» должно иметь такой же тип данных, как первый параметр, используется, когда «смещение» находится за пределами секции. Если не задано, то возвращается NULL.

Функции FIRST_VALUE() и LAST_VALUE() – возвращают первое или последнее значение из упорядоченного набора значений. Синтаксис функций имеет следующий вид:

FIRST_VALUE|LAST_VALUE(<столбец>) OVER([PARTITION BY <столбцы>] ORDER BY <столбцы>)

3. Практическая часть

Таблица ***Ученики:***

ID	Фамилия	Предмет	Школа	Баллы
1	Иванова	Математика	Лицей	98,5
2	Петров	Физика	Лицей	99
3	Сидоров	Математика	Лицей	88
4	Полухина	Физика	Гимназия	78
5	Матвеева	Химия	Лицей	92
6	Касимов	Химия	Гимназия	68
7	Нурулин	Математика	Гимназия	81
8	Авдеев	Физика	Лицей	87
9	Никитина	Химия	Лицей	94
10	Барышева	Химия	Лицей	88

Пример 1: Вывести список учеников и максимальный балл в каждой строке:

```
SELECT
        Фамилия
        ,Предмет
        ,Школа
        ,Баллы
        , MAX(Баллы) OVER() AS Макс_Балл
    FROM
        Ученики
```

Пример 2: Вывести список учеников и разницу между баллами ученика и минимальным баллом в каждой строке:

```
SELECT
        Фамилия
        ,Предмет
        ,Школа
        ,Баллы
        , Баллы - MIN(Баллы) OVER() AS Разница
    FROM
        Ученики
```

Пример 3: Вывести список учеников и процентное соотношение к суммарному баллу в каждой строке:

```
SELECT
        Фамилия
        ,Предмет
        ,Школа
        ,Баллы
        ,Баллы * 100 / SUM(Баллы) OVER() AS Процент
    FROM
        Ученики
```

Пример 4: Вывести список учеников и средний балл в школе в каждой строке:

```
SELECT
        Фамилия
        ,Предмет
        ,Школа
        ,Баллы
        ,AVG(Баллы) OVER(PARTITION BY Школа) AS Сред_Шк
    FROM
        Ученики
```

Пример 5: Вывести список учеников и количество учеников в школе в каждой строке, отсортировать по школам в оконной функции:

```
SELECT
        Фамилия
        ,Предмет
        ,Школа
        ,Баллы
        ,COUNT(*) OVER(PARTITION BY Школа ORDER BY Школа) AS
Кол_Шк
    FROM
        Ученики
```

Пример 6: Вывести список учеников и номер строки при сортировке по баллам по убыванию:

```
SELECT
    ROW_NUMBER() OVER(ORDER BY Баллы DESC) AS Номер_строки
    ,Фамилия
    ,Предмет
```

```
    ,Школа
    ,Баллы
FROM
    Ученики
```

Пример 7: Вывести список учеников и номер строки внутри школы при сортировке по баллам по убыванию:

```
SELECT
    ROW_NUMBER() OVER(PARTITION BY Школа ORDER BY Баллы DESC) AS Номер_строки
    ,Фамилия
    ,Предмет
    ,Школа
    ,Баллы
FROM
    Ученики
```

Пример 8: Вывести список учеников и ранг по баллам в каждой школе:

```
SELECT
    RANK() OVER(PARTITION BY Школа ORDER BY Баллы DESC) AS Ранг_Шк
    ,Фамилия
    ,Предмет
    ,Школа
    ,Баллы
FROM
    Ученики
```

Пример 9: Вывести список учеников и сжатый ранг по баллам в каждой школе. Результат отсортировать по фамилии в алфавитном порядке:

```
SELECT
    DENSE_RANK() OVER(PARTITION BY Школа ORDER BY Баллы DESC) AS Сж_Ранг_Шк
    ,Фамилия
    ,Предмет
    ,Школа
    ,Баллы
FROM
    Ученики
ORDER BY
    Фамилия
```

Пример 10: Вывести список учеников, распределенных по трем группам по фамилии:

```
SELECT
    NTILE(3) OVER(ORDER BY Фамилия) AS Гр_Фам
    ,Фамилия
    ,Предмет
    ,Школа
    ,Баллы
FROM
    Ученики
```

Пример 11: Вывести список учеников, распределенных по двум группам по баллам внутри школы:

```
SELECT
    NTILE(2) OVER(PARTITION BY Школа ORDER BY Баллы DESC) AS Гр_Балл
    ,Фамилия
    ,Предмет
    ,Школа
    ,Баллы
FROM
    Ученики
```

Пример 12: Вывести список учеников и разницу с баллами предыдущего ученика, при сортировке по возрастанию баллов:

```
SELECT
    Фамилия
    ,Предмет
    ,Школа
    ,Баллы
    ,Баллы - LAG(Баллы) OVER(ORDER BY Баллы) AS Разница
FROM
    Ученики
```

Пример 13: Вывести список учеников и разницу с баллами ученика через две позиции при сортировке по убыванию баллов, значение по умолчанию использовать 0:

```
SELECT
    Фамилия
    ,Предмет
    ,Школа
    ,Баллы
```

```
        ,Баллы - LEAD(Баллы, 2, 0) OVER(ORDER BY Баллы DESC) AS Разница
    FROM
        Ученики
```

Пример 14: Вывести список учеников и разницу с баллами первого ученика при сортировке по убыванию баллов:

```
    SELECT
        Фамилия
        ,Предмет
        ,Школа
        ,Баллы
        ,FIRST_VALUE(Баллы) OVER(ORDER BY Баллы DESC) - Баллы AS Разница
    FROM
        Ученики
```

Пример 15: Вывести список учеников и разницу с баллами последнего ученика в школе при сортировке по убыванию баллов:

```
    SELECT
        Фамилия
        ,Предмет
        ,Школа
        ,Баллы
        ,LAST_VALUE(Баллы) OVER(ORDER BY Баллы RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) - Баллы AS Разница
    FROM
        Ученики
```

4. Задание

1. Вывести список учеников и разницу между баллами ученика и максимальным баллом в каждой строке.
2. Вывести список учеников и процентное соотношение к среднему баллу в каждой строке.
3. Вывести список учеников и минимальный балл в школе в каждой строке.
4. Вывести список учеников и суммарный балл в школе в каждой строке, отсортировать по школам в оконной функции.
5. Вывести список учеников и номер строки при сортировке по фамилиям в обратном алфавитном порядке.
6. Вывести список учеников, номер строки внутри школы и количество учеников в школе при сортировке по баллам по убыванию.

7. Вывести список учеников и ранг по баллам.

8. Вывести список учеников и сжатый ранг по баллам. Результат отсортировать по фамилии в алфавитном порядке.

9. Вывести список учеников, распределенных по пяти группам по фамилии.

10. Вывести список учеников, распределенных по трем группам по баллам внутри школы.

11. Вывести список учеников и разницу с баллами ученика, находящегося выше на три позиции при сортировке по возрастанию баллов.

12. Вывести список учеников и разницу с баллами следующего ученика при сортировке по убыванию баллов, значение по умолчанию использовать 0.

Практическая работа № 18

Инструкции OLAP

1. Цель работы

1. Изучить использование ROLLUP.
2. Изучить использование CUBE.
3. Изучить использование GROUPING SETS.
4. Изучить использование PIVOT.
5. Изучить использование UNPIVOT.

2. Теоретическая часть

Operator ROLLUP создаёт группу для каждого сочетания выражений столбцов. Кроме того, выполняет сведение результатов в промежуточные и общие итоги. Для этого запрос перемещается справа налево, уменьшая количество выражений столбцов, по которым он создает группы и агрегаты. Синтаксис команды имеет следующий вид:

GROUP BY ROLLUP(<список столбцов>)

или

GROUP BY <список столбцов> WITH ROLLUP

Порядок столбцов влияет на выходные данные ROLLUP и может отразиться на количестве строк в результирующем наборе.

GROUP BY ROLLUP(col1, col2) создает группы для каждой комбинации выражений столбцов в следующих списках:

col1, col2

col1, NULL

NULL, NULL – это общий итог.

Оператор CUBE создает группы для всех возможных сочетаний столбцов. Синтаксис команды имеет следующий вид:

GROUP BY CUBE(<список столбцов>)

или

GROUP BY <список столбцов> WITH CUBE

GROUP BY CUBE(col1, col2) создает группы для каждой комбинации выражений столбцов в следующих списках:

col1, col2

col1, NULL

NULL, col2

NULL, NULL – это общий итог.

Оператор GROUPING SETS позволяет объединять несколько предложений GROUP BY в одно предложение GROUP BY. Результаты эквивалентны тем, что формируются с применением конструкции UNION ALL к указанным группам. Синтаксис команды имеет следующий вид:

GROUP BY GROUPING SETS(<список столбцов>)

Если параметр GROUPING SETS имеет два или более элементов, результатом будет объединение элементов.

SQL не консолидирует повторяющиеся группы, созданные для списка GROUPING SETS.

GROUP BY GROUPING SETS(col1, col2) создает группы для каждой комбинации выражений столбцов в следующих списках:

col1, NULL

NULL, col2

Для предложения GROUP BY, использующего ROLLUP, CUBE или GROUPING SETS, допускается максимум 32 выражения.

Функция GROUPING указывает, является ли указанное выражение столбца в списке GROUP BY статистическим или нет. В результирующем наборе возврат будет 1 (статистическое выражение) или ноль (нестатистическое выражение). Функция GROUPING может использоваться только в предложениях SELECT, HAVING и ORDER BY, если указано предложение GROUP BY.

Функция GROUPING_ID вычисляет уровень группирования. Функция GROUPING_ID может использоваться только в предложениях SELECT, HAVING и ORDER BY, если указано предложение GROUP BY.

Оператор PIVOT поворачивает возвращающее табличное значение выражение, преобразуя уникальные значения одного столбца выражения в несколько выходных столбцов. В случае необходимости PIVOT также объединяет оставшиеся повторяющиеся значения столбца и отображает их в выходных данных.

Синтаксис PIVOT является более простым и понятным, чем синтаксис, который может выполнить то же действие с помощью последовательности инструкций SELECT...CASE. Синтаксис имеет следующий вид:

SELECT < столбцы для группировки>, <пивотируемые столбцы >

FROM

(<запрос возвращающий данных>)

AS <псевдоним>

PIVOT

(<агрегирующая функция>(<столбец>)

FOR [<столбец, значения которого будут заголовками>]

IN (<список пивотируемых столбцов>)

) AS <псевдоним для пивот-таблицы>

Оператор UNPIVOT поворачивает возвращающее табличное значение выражения, преобразуя нескольких выходных столбцов в один столбец. Синтаксис имеет следующий вид:

SELECT <список столбцов>
FROM
<таблица>
UNPIVOT
(<столбец значения строк> FOR [<столбец, значения заголовок>]
IN (<список анпивотируемых столбцов>)
) AS <псевдоним для анпивот-таблицы>

3. Практическая часть

Таблица ***Ученики:***

ID	Фамилия	Предмет	Школа	Баллы
1	Иванова	Математика	Лицей	98,5
2	Петров	Физика	Лицей	99
3	Сидоров	Математика	Лицей	88
4	Полухина	Физика	Гимназия	78
5	Матвеева	Химия	Лицей	92
6	Касимов	Химия	Гимназия	68
7	Нурулин	Математика	Гимназия	81
8	Авдеев	Физика	Лицей	87
9	Никитина	Химия	Лицей	94
10	Барышева	Химия	Лицей	88

Пример 1: Напишите запрос, который выводит количество учеников по предметам по каждой школе, и промежуточные итоги:

```
SELECT
    Предмет
    ,Школа
    ,COUNT(Фамилия) AS Количество
FROM
    Ученики
GROUP BY
    Предмет, Школа WITH ROLLUP
```

Пример 2: Напишите запрос, который выводит количество учеников по предметам и по школам, и промежуточные итоги:

```
SELECT
```

```
        Предмет
        ,Школа
        ,COUNT(Фамилия) AS Количество
FROM
        Ученики
GROUP BY
        Предмет, Школа WITH CUBE
```

Пример 3: Напишите запрос, который выводит количество учеников по предметам и по школам:

```
SELECT
        Предмет
        ,Школа
        ,COUNT(Фамилия) AS Количество
FROM
        Ученики
GROUP BY
        GROUPING SETS(Предмет, Школа)
```

Пример 4: Напишите запрос, который выводит количество учеников по предметам по каждой школе и промежуточные итоги. NULL значения заменить на соответствующий текст:

```
SELECT
        COALESCE(Предмет, 'ИТОГО') AS Предмет
        ,COALESCE(Школа, 'Итого') AS Школа
        ,COUNT(Фамилия) AS Количество
FROM
        Ученики
GROUP BY
        ROLLUP(Предмет, Школа)
```

Пример 5: Напишите запрос, который выводит количество учеников по предметам и по школам, и промежуточные итоги. В итоговых строках NULL значения заменить на соответствующий текст в зависимости от группировки:

```
SELECT
        IIF(GROUPING(Предмет)=1, 'ИТОГО', Предмет) AS Предмет
        ,IIF(GROUPING(Школа)=1, 'Итого', Школа) AS Школа
        ,COUNT(Фамилия) AS Количество
FROM
        Ученики
GROUP BY
        CUBE(Предмет, Школа)
```

Пример 6: Напишите запрос, который выводит количество учеников по предметам и по школам. В итоговых строках NULL значения заменить на соответствующий текст в зависимости от уровней группировки:

```
SELECT
	CASE GROUPING_ID(Предмет, Школа)
		WHEN 1 THEN 'Итого по предметам'
		WHEN 3 THEN 'Итого'
		ELSE ''
	END AS Итого
	,ISNULL(Предмет, '') AS Предмет
	,ISNULL(Школа, '') AS Школа
	,COUNT(Фамилия) AS Количество
FROM
	Ученики
GROUP BY
	ROLLUP(Предмет, Школа)
```

Пример 7: Напишите запрос, который выводит количество учеников по предметам по столбцам:

```
SELECT
	'Количество' AS [Количество учеников по предметам]
	,Математика
	,Физика
	,Химия
FROM
	(
		SELECT
			Предмет
			,Фамилия
		FROM
			Ученики
	) AS SOURCE_TABLE
	PIVOT
	(
		COUNT(Фамилия)
		FOR Предмет IN (Математика, Физика, Химия)
	) AS PIVOT_TABLE
```

Пример 8: Напишите запрос для вывода количества учеников для каждой школы по каждому предмету (школы должны быть указаны в строках, предметы в столбцах):

```
SELECT
        Школа
        ,Математика
        ,Физика
        ,Химия
FROM
        (
                SELECT
                        Школа
                        ,Предмет
                        ,Фамилия
                FROM
                        Ученики
        ) AS SOURCE_TABLE
        PIVOT
        (
                COUNT(Фамилия)
                FOR Предмет IN (Математика, Физика, Химия)
        ) AS PIVOT_TABLE
```

Пример 9: Напишите запрос, который выводит фамилию учеников и предметы вместе со школами в один столбец:

```
SELECT
        Фамилия,
        [Предмет или школа]
FROM Ученики
UNPIVOT (
   [Предмет или школа] FOR Значение IN (Предмет, Школа)
) unpvt
```

4. Задание

1. Напишите запрос, который выводит максимальный балл учеников по школам, по каждому предмету по каждой школе и промежуточные итоги.

2. Напишите запрос, который выводит минимальный балл учеников по школам и по предметам, и промежуточные итоги.

3. Напишите запрос, который выводит средний балл учеников по школам и по предметам.

4. Напишите запрос, который выводит количество учеников по каждой школе по предметам и промежуточные итоги. NULL значения заменить на соответствующий текст.

5. Напишите запрос, который выводит суммарный балл учеников по школам и по предметам, и промежуточные итоги. В итоговых строках NULL значения заменить на соответствующий текст в зависимости от группировки.

6. Напишите запрос, который выводит максимальный балл учеников по школам и по предметам. В итоговых строках NULL значения заменить на соответствующий текст в зависимости от уровней группировки.

7. Напишите запрос, который выводит средний балл учеников по школам в столбцы.

8. Напишите запрос, который выводит средний балл учеников по школам в столбцы и по предметам в строки.

9. Напишите запрос, который выводит названия предметов, фамилии учеников и школы в один столбец.

Практическая работа № 19

Динамический SQL, XML и JSON

1. Цель работы

1. Изучить использование динамического SQL.
2. Изучить вывод данных в формате XML.
3. Изучить вывод данных в формате JSON.

2. Теоретическая часть

В интерактивных приложениях часто состав SQL-запросов заранее не известен и может изменяться в зависимости от условий.

Для решения таких ситуаций используется динамический SQL. Запросы формируются в виде текстовых строк непосредственно во время работы программы. Выполнение динамического SQL медленнее, чем статического.

Для выполнения динамического SQL используется команда EXEC | EXECUTE. Упрощенный синтаксис имеет следующий вид:

EXEC | EXECUTE (<строка динамического SQL>)

Результат выполнения запроса можно получать в формате XML. Для этого в запросе необходимо указать предложение FOR XML.

В предложении FOR XML можно указать один из следующих режимов:

RAW – в этом режиме создается одиночный элемент <row> для каждой строки набора строк, возвращенного инструкцией SELECT. XML-иерархию можно создать с помощью написания вложенных запросов FOR XML.

AUTO – в этом режиме вложенность XML создается эвристически, в зависимости от метода определения инструкции SELECT.

EXPLICIT – этот режим предоставляет больше возможностей по управлению формой XML-структуры. В XML-структуре могут быть использованы смешанные атрибуты и элементы. Однако написание запросов в режиме EXPLICIT может быть очень обременительным.

PATH – этот режим совместно с вложенным запросом FOR XML обеспечивает гибкость режима EXPLICIT более простым образом.

Эти режимы на самом деле работают только при выполнении запросов, для которых они установлены. Они не влияют на результаты любых последующих запросов.

Результат выполнения запроса можно получать в формате JSON. Для этого в запросе необходимо указать предложение FOR JSON. Предложение FOR JSON упрощает клиентские приложения, делегируя форматирование выходных данных JSON из приложения в SQL Server.

В предложении FOR XML можно указать один из следующих режимов:

PATH – этот режим используется для того чтобы сохранить полный контроль над форматом выходных данных JSON. Можно создавать объекты-оболочки и вкладывать сложные свойства друг в друга.

AUTO – этот режим используется для того чтобы отформатировать выходные данные автоматически на основе структуры инструкции SELECT.

3. Практическая часть

Пример 1: Создайте и запустите динамический SQL-запрос, выбирающий все данные из заданной таблицы:

```
DECLARE @DSQL VARCHAR(100)
DECLARE @TN VARCHAR(50)

SET @TN = 'Ученики'
SET @DSQL = 'SELECT * FROM ' + @TN

EXECUTE (@DSQL)
```

Пример 2: Создайте и запустите динамический SQL-запрос, выбирающий всех учеников из таблицы «Ученики», вторая буква фамилии которых расположена в заданном диапазоне:

```
DECLARE @L1 CHAR(1)
DECLARE @L2 CHAR(1)

SET @L1 = 'а'
SET @L2 = 'д'

EXECUTE ('SELECT * FROM Ученики WHERE Фамилия LIKE ''_[' + @L1 + '-' + @L2 + ']%''')
```

Пример 3: Создайте временную таблицу #Temp и добавить в нее название пяти таблиц из базы данных «Учебная». Создайте курсор, который, построчно читая таблицу #Temp, выбирает из каждой таблицы все данные:

```
CREATE TABLE #TEMP
(
	TABLENAME VARCHAR(50)
)

INSERT INTO
	#TEMP
```

```
VALUES
        ('Дисциплина'),
        ('Заявка'),
        ('Инженер'),
        ('Кафедра'),
        ('Сотрудник')

DECLARE TC CURSOR
        FOR SELECT TABLENAME FROM #TEMP

OPEN TC
DECLARE @TN VARCHAR(50)

FETCH FROM TC INTO @TN
WHILE @@FETCH_STATUS = 0
BEGIN
        EXECUTE ('SELECT * FROM ' + @TN)
        FETCH FROM TC INTO @TN
END

CLOSE TC
DEALLOCATE TC

DROP TABLE #TEMP
```

Пример 4: Выберите данные из таблицы «Ученики» и экспортируйте в RAW формате XML:

```
SELECT
        ID
        ,Фамилия
        ,Предмет
        ,Школа
        ,Баллы
FROM
        Ученики
FOR XML RAW
```

Пример 5: Выберите данные из таблицы «Страны» и экспортируйте в PATH формате XML:

```
SELECT
        Название
        ,Столица
```

```
        ,Площадь
        ,Население
        ,Континент
    FROM
        Страны
    FOR XML PATH
```

Пример 6: Выберите данные из таблицы «Ученики» и экспортируйте в PATH формате JSON:

```
    SELECT
        ID
        ,Фамилия
        ,Предмет
        ,Школа
        ,Баллы
    FROM
        Ученики
    FOR JSON PATH
```

4. Задание

1. Создайте и запустите динамический SQL-запрос, выбирающий первые N строк из заданной таблицы.
2. Создайте и запустите динамический SQL-запрос, выбирающий все страны из таблицы «Страны», последняя буква названия которых расположена в заданном диапазоне.
3. Создайте временную таблицу #Temp и добавьте к ней название столбцов таблицы «Страны». Создайте курсор, который, построчно читая таблицу #Temp, выбирает каждый раз соответствующий столбец из таблицы «Страны».
4. Создайте процедуру, которая принимает как параметр список столбцов, название таблицы и выбирает заданные столбцы из заданной таблицы.
5. Создайте процедуру, принимающую как параметр список столбцов, название таблицы, название проверяемого столбца, знак сравнения, значение проверки и выбирающую заданные столбцы из заданной таблицы в заданных условиях.
6. Выберите список европейских стран из таблицы «Страны» и экспортируйте в RAW формате XML.
7. Выберите список стран с населением больше 100 млн. чел. из таблицы «Страны» и экспортируйте в PATH формате XML.
8. Выберите список учеников из школы «Лицей» из таблицы «Ученики» и экспортируйте в PATH формате JSON

Практическая работа № 20

Пространственные данные

1. Цель работы

1. Изучить пространственные данные.
2. Изучить тип данных GEOMETRY.
3. Изучить тип данных GEOGRAPHY.

2. Теоретическая часть

Пространственные данные представляют сведения о физическом расположении и форме геометрических объектов. Этими объектами могут быть точки расположения или более сложные объекты, такие как страны, дороги или озера.

SQL Server поддерживает *два пространственных типа данных*: GEOMETRY и GEOGRAPHY.

Пространственный тип данных GEOMETRY представляет данные в евклидовой (плоской) системе координат.

Пространственный тип данных GEOGRAPHY представляет данные в системе координат для сферической Земли.

Типы данных GEOMETRY и GEOGRAPHY поддерживают шестнадцать объектов пространственных данных или типов экземпляров. Однако только одиннадцать из этих типов экземпляров являются материализуемыми. Такие экземпляры можно создавать в базе данных и работать с ними.

Часто используемые объекты:

POINT(X Y) – для геометрических типов данных представляет собой единое место, где x и y представляют координаты создаваемого экземпляра. Для географических типов данных представляет единичное расположение, где X представляет широту, а Y – долготу. Значения широты и долготы измеряются в градусах. Значения широты находятся в интервале [-90, 90]. Значения долготы находятся в интервале [-180, 180].

LINESTRING(X_1 Y_1, X_2 Y_2[, X_n Y_n]) – является одномерным объектом, представляющим последовательность точек и соединяющих их линейных сегментов.

CIRCULARSTRING(X_1 Y_1, X_2 Y_2, X_3 Y_3 [, X_n Y_n]) – коллекция, состоящая из нуля или большего количества непрерывных круговых сегментов дуги. *Сегмент дуги* – это сегмент кривой, определяемый тремя точками на двумерной плоскости; первая точка не может совпадать с третьей. Если все три точки сегмента дуги лежат на одной прямой, сегмент дуги считается линейным сегментом.

POLYGON(X_1 Y_1, X_2 Y_2, X_3 Y_3 [, X_n Y_n], X_1 Y_1) – представляет собой двухмерную поверхность, хранимую в виде последовательности точек, содержащую не менее трех уникальных точек, первая и последняя точки должны совпадать.

3. Практическая часть

Пример 1: Создайте временную таблицу, которая имеет один столбец типа GEOMETRY. Добавьте в таблицу пространственный объект LINESTRING с координатами (1 1, 3 3, 2 4, 2 0, 1 1). Создайте SQL-запрос, выбирающий все данные из заданной таблицы:

```
CREATE TABLE #ST
(
	D GEOMETRY
)

INSERT
	#ST
VALUES
('LINESTRING(1 1, 3 3, 2 4, 2 0, 1 1)')

SELECT
	D
FROM
	#ST

DROP TABLE #ST
```

Пример 2: Создайте временную таблицу, которая имеет один столбец типа GEOMETRY. Добавьте в таблицу пространственный объект POLYGON с координатами (-3 7, 10 24, 30 16, 13 18, 4 0, 14 -6, 6 -13, 0 -4, -14 -13, -22 -39, -20 -7, -7 3, -11 7, -12 9, -11 10, -9 10, -3 7). Создайте SQL-запрос, выбирающий все данные из заданной таблицы:

```
CREATE TABLE #ST
(
	D GEOMETRY
)

INSERT
	#ST
VALUES
('POLYGON((-3 7, 10 24, 30 16,
			13 18, 4 0, 14 -6,
			6 -13, 0 -4, -14 -13,
			-22 -39, -20 -7, -7 3,
			-11 7, -12 9, -11 10,
			-9 10, -3 7))')
```

```
SELECT
        D
FROM
        #ST

DROP TABLE #ST
```

4. Задание

1. Создайте временную таблицу, состоящую из двух полей типа geometry и varchar(20). Заполните таблицу заданными полигонами и их названиями. Выберите все поля из таблицы.

Номер участка	Долгота	Широта
Участок №6		
Точка №1	21369,11	21603,40
Точка №2	21584,64	21162,81
Точка №3	21587,40	21027,71
Точка №4	20727,78	21009,83
Точка №5	20720,45	21369,76
Точка №6	20898,96	21373,41
Участок №3		
Точка №1	20728,49	20975,13
Точка №2	20740,38	20393,97
Точка №3	21107,99	20401,49
Точка №4	21103,40	20625,44
Точка №5	21160,39	20626,63
Точка №6	21153,08	20983,82
Участок №4		
Точка №1	21588,12	20992,79
Точка №2	21597,74	20441,54
Точка №3	21269,66	20435,81
Точка №4	21266,28	20629,74
Точка №5	21209,25	20628,75
Точка №6	21203,01	20985,98
Участок №7		
Точка №1	22775,32	21461,21
Точка №2	23234,54	21075,88
Точка №3	22977,79	20610,45
Точка №4	22637,02	20798,44
Точка №5	22549,38	21051,51

Номер участка	Долгота	Широта
Участок №8		
Точка №1	22624,91	19970,80
Точка №2	22854,97	20387,52
Точка №3	22398,62	20639,24
Точка №4	22168,68	20222,45
Участок №11		
Точка №1	23621,55	20255,27
Точка №2	23855,79	20314,44
Точка №3	23827,55	20426,38
Точка №4	23946,32	20641,48
Точка №5	23569,81	20849,18
Точка №6	23524,91	20850,59
Точка №7	23295,70	20435,08
Участок №12		
Точка №1	23596,67	19981,86
Точка №2	23366,76	19565,04
Точка №3	22944,29	19798,09
Точка №4	23174,24	20214,89
Участок №13		
Точка №1	23974,68	20627,60
Точка №2	23863,62	20422,64
Точка №3	23928,00	20159,98
Точка №4	23792,27	19913,93
Точка №5	24068,07	19761,76
Точка №6	24412,47	20386,08
Участок №14		
Точка №1	24045,03	19732,76
Точка №2	23622,57	19965,83
Точка №3	23392,62	19549,03
Точка №4	23815,10	19315,97
Участок №5 зап		
Точка №1	21638,11	20993,74
Точка №2	21649,39	20442,59
Точка №3	21793,84	20445,55
Точка №4	22017,66	20851,20
Точка №5	21754,09	20996,13
Участок №5 вост		
Точка №1	22348,79	20662,78
Точка №2	22059,86	20822,21
Точка №3	21854,05	20449,22
Точка №4	21856,16	20390,95

Номер участка	Долгота	Широта
Точка №5	22118,83	20246,02
Участок №7 А		
Точка №1	23231,72	20470,38
Точка №2	23004,06	20595,96
Точка №3	23257,96	21056,22
Точка №4	23460,94	20885,90
Участок № б/н		
Точка №1	22899,36	20403,01
Точка №2	23127,02	20277,41
Точка №3	23217,83	20442,02
Точка №4	22990,17	20567,60
Участок № б/н		
Точка №1	22869,16	20419,68
Точка №2	22536,51	20603,15
Точка №3	22627,15	20767,87
Точка №4	22959,86	20584,33
Участок № б/н		
Точка №1	22407,55	20676,04
Точка №2	22559,71	20927,48
Точка №3	22608,36	20789,50
Точка №4	22513,51	20617,58
Участок №8		
Точка №1	22651,93	19957,66
Точка №2	22881,85	20374,45
Точка №3	23109,51	20248,85
Точка №4	22879,56	19832,05
Участок № б/н		
Точка №1	23618,51	20009,77
Точка №2	23650,78	20068,26
Точка №3	23610,13	20229,12
Точка №4	23284,66	20408,66
Точка №5	23193,85	20244,05
Участок № б/н		
Точка №1	23635,13	20229,91
Точка №2	23860,44	20286,85
Точка №3	23891,24	20165,00
Точка №4	23761,22	19929,80
Точка №5	23638,63	19996,93
Точка №6	23676,61	20065,78
Озеро		
Точка №1	21637,28	21029,09

Номер участка	Долгота	Широта
Точка №2	21634,42	21174,83
Точка №3	21414,03	21625,36
Точка №4	22150,41	21985,58
Точка №5	22751,91	21480,87
Точка №6	22515,53	21054,35
Точка №7	22544,07	20975,30
Точка №8	22385,66	20688,12
Точка №9	21763,52	21031,31

Практическая работа

1. Создать таблицы (не менее 5-и) по своему варианту.

2. Заполнить таблицы (справочные таблицы – не менее 5-и строк, операционные таблицы – не менее 20-и).

3. Создать запросы:

a. Запрос на выбор всех данных по двум полям таблицы;

b. Запрос на выбор всех *неповторяющихся* данных по одному полю таблицы;

c. Запрос на выбор всех полей и записей таблицы, сгруппированных по значению одного поля, с использованием условия на группу (секции GROUP BY, HAVING) и с заголовками колонок, заданными в запросе;

d. Запрос на выбор всех *неповторяющихся* записей по одному полю таблицы с колонкой, образованной агрегирующей функцией SUM и озаглавленной в соответствии со смыслом;

e. Выбор нескольких (не всех) полей таблицы, отсортированных по *убыванию*;

f. Выбор произвольного количества полей таблицы с добавлением поля, являющегося результатом арифметического выражения, в котором участвуют значения поля таблицы;

g. Запрос на выбор всех записей по одному полю таблицы с колонкой, образованной агрегирующей функцией SUM и озаглавленной в соответствии со смыслом; выбор записей с использованием условия диапазона (between);

h. Запрос на выбор всех записей по произвольному количеству полей таблицы с использованием агрегирующей функции AVG и условием на отбор записей, заданном в секции WHERE;

i. Запрос на выбор двух полей таблицы с вычислением третьего поля по данным таблицы и сортировкой по убыванию по первому полю, по возрастанию по второму и по убыванию по третьему;

j. Запрос на выборку данных таблицы с условием сравнения по шаблону LIKE;

k. Запрос с отбором по условию и сортировкой по убыванию одного из полей, а также добавлением поля, содержащего для всех записей константу, определенную при конструировании запроса;

l. Запрос с использованием агрегирующих функций MIN и MAX;

m. Запрос с использованием сложного условия с логическими операторами AND, OR и сортировкой.

ВАРИАНТЫ ЗАДАНИЙ

№	Задание	Дополнительные возможности системы
	Базы данных	*Ориентировочные таблицы:*
1.	«Студенческая библиотека»	• «Список студентов» • «Список книг» • «Деятельность»
2.	«Страховая фирма»	• «Виды страховок» • «Клиенты\объекты» • «Страховая деятельность»
3.	«Агентство недвижимости»	• «Объекты недвижимости» • «Продажи» • «Покупки»
4.	ГИБДД (Государственная инспекция безопасности дорожного движения)	• «Список водителей» • «Список автомобилей» • «Штрафы»
5.	«Деканат ВУЗа»	• «Список студентов» • «Список предметов» • «Сессия»»
6.	«Отдел кадров производственного пред-приятия»	• «Сотрудники» • «Штатное расписание» • «Отделы» • «Цеха»
7.	«Фирмы покупки и продажи автомобилей»	• «Продажи» • «Покупки» • «Автомобили»
8.	«Гостиница»	• «Номера» • «Счета» • «Клиенты»
9.	«Расчет квартплаты ТСЖ»	• «Список жильцов» • «Оплаты» • «Тарифы»
10	«Железнодорожные кассы»	• «Продажи» • «Посадочные места» • «Направления»
11	«Авиапассажирские перевозки»	• «Рейсы» • «Самолеты» • «Продажи»
12	«Музей»	• «Экспонаты» • «Авторы» • «Экспозиции»

№	Задание	Дополнительные возможности системы
	Базы данных	*Ориентировочные таблицы:*
13	«Спортивные комплексы района»	• «Нормативы» • «Спортсмены» • «Соревнования»
14	«Экзаменационная сессия»	• «Предметы» • «Оценки» • «Студенты»
15	«Турагентство»	• «Туры» • «Продажи»
16	«Аптека»	• «Товары» • «Поставщики» • «Продажи»
17	«Сборка и реализация компьютеров»	• «Продукция» • «Клиенты» • «Заказы»
18	«Продуктовые магазины района»	• «Продажи» • «Отделы» • «Товары»
19	«Больница» (одного отделения)	• «Больные» • «Диагнозы» • «Врачи»
20	«Видеотека»	• «Артисты» • «Фильмы» • «Продажи»

Библиографический список

1. Алапати С.Р. Oracle Database 11g: Руководство администратора баз данных / Пер. с англ. М.: Вильямс, 2010.
2. Андон Ф., Резниченко В. Язык запросов SQL: Учебный курс. СПб.: Питер; Киев: Издательская группа BHV, 2006.
3. Аносов А. Критерии выбора СУБД при создании информационных систем. URL: http://www.interface.ru/home.asp?artId=2147 дата обращения 24.05.2020).
4. Бондарь А.Г. Microsoft SQL Server 2014. СПб.: БХВ-Петербург, 2015.
5. Борри Х. Fireblrd: Руководство разработчика баз данных / Пер. с англ. СПб.: БХВ-Петербург, 2006.
6. Виейра Р. Программирование баз данных Microsoft SQL Server 2005 для профессионалов / Пер. с англ. М.: ООО «И.Д. Вильямс», 2008.
7. Голицына О.Л., Максимов Н.В., Попов И.И. Базы данных: Учеб. пособие. М.: Форум; ИНФРА-М, 2012.
8. Гринвальд Р., Стаковьяк Р., Стерн Дж. Oracle 11g. Основы / Пер. с англ. 4-е изд. СПб.: СимволПлюс, 2009.
9. Грофф Дж., Вайнберг П., Оппель Э. SQL: Полное руководство / Пер. с англ. 3-е изд. М.: ООО «И.Д. Вильямс». 2015.
10. Дейт К. Введение в системы баз данных. М.: Наука, 1980. 464 с.
11. Дейт К.Дж. Введение в системы баз данных / Пер. с англ. 8-е изд. М.: Издательский дом «Вильямс», 2005.
12. Ковязин А.Н., Востриков С.М. Мир InterBase: архитектура, администрирование и разработка приложений баз данных в InterBase/FireBird/Yafill. 4-е изд. М.: КУДИЦ-ОБРАЗ, 2006.
13. Коннолли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика: учеб. пособие; пер. с англ. 2-е изд. М.: Вильямс, 2000. 1120 с.
14. Кристофидес Н. Теория графов. Алгоритмический подход / Пер. с англ. М.: Мир, 1978. 432 с.
15. Логинов Д. Почему мне нравится FireBird. URL: http://www.loginovprojects.ru/index.php?page=whyfirebird (дата обращения 09.01.2020).
16. Мейер М. Теория реляционных баз данных. М.: Мир, 1987. 608 с.
17. Официальный сайт компании IBase.ru. URL: http://www.ibase.ru (дата обращения 15.05.2020).
18. Официальный сайт компании Microsoft. URL: http://www.microsoft. com (дата обращения 31.05.2020).
19. Пери Дж., Пост Дж. Введение в Oracle 10g / Пер. с англ. М.: ООО «И.Д. Вильямс», 2006.

20. Петкович Д. Microsoft® SQL Server 2012. Руководство для начинающих / Пер. с англ. СПб.: БХВ-Петербург, 2013.

21. Полякова Л. Основы SQL: электронный курс. URL:http://www.intuit.ru/studies/courses/5/5/info (дата обращения 19.04.2020).

22. Пржиялковский В. Контекст сеанса в Oracle. URL: http://citforum.ru/database/oracle/ session_context/ index.shtml дата обращения 30.03.2020).

23. Руководство по языку SQL СУБД FireBird 4.0 / Под ред. Д. Симонова. URL: http://www.ibase.ru/files/firebird/ Firebird_4_0_Language_ Reference_RUS.pdf (дата обращения 15.06.2020).

24. Селко Д. Стиль программирования Джо Селко на SQL / Пер. с англ. М.: Русская редакция; СПб.: Питер, 2006.

25. Словарик по FireBird. URL: http://www firebirdsql.su (дата обращения 18.02.2020).

26. Ульман Дж. Основы систем баз данных. М.: Финансы и статистика, 1983. 334 с.

27. Функции по категориям. URL: http://oracleplsql.ru (дата обращения 30.06.2020).

28. Bonie H. FireBird 3.0 Release Notes, 2013.

29. Chen P.P.-S. The Entity-relationship Model: Toward a Unified View of Data // SIGIR Forum. 1975. Vol. 10, no. 3. Pp. 9–9. DOI:10.1145/1095277.1095279.

30. Codd E.F. A Relational Model of Data for Large Shared Data Banks // Commun. ACM. 1970. Vol. 13, no. 6. Pp. 377–387. DOI:10.1145/362384.362685.

31. Fagin R. The Decomposition Versus Synthetic Approach to Relational Database Design // Proceedings of the 3rd International Conference on Very Large Data Bases. Vol. 3. VLDB Endowment, 1977. Pp. 441–446. (VLDB ’77).

32. Kossmann D., Stocker K. Iterative Dynamic Programming: A New Class of Query Optimization Algorithms // ACM Trans. Database Syst. 2000. Vol. 25, no. 1. Pp. 43–82. DOI: 10.1145/352958.352982.

33. Lane P. Oracle Database Globalization Support Guide, 11g Release 2 (11.2) / P. Lane. URL: https://docs.oracle.com/cd/ E11882_01/server.112/ e10729/toc.htm (дата обращения 17.06.2020).

34. Lorentz D., Roeser M.B. Oracle Database SQL Language Reference, 11g Release 1 (11.1). URL: https://docs.oracle.com/cd/ B28359_01/server.111/ b28286/ toc.htm (дата обращения 20.05.2020).

35. Moore S. Oracle Database Express Edition 2 Day Developer's Guide, 11g Release 2 (11.2). URL: https://docs.oracle.com/cd/E17781_01 /appdev.112/e18147/toc.htm (дата обращения 20.05.2020).

36. Moore S. Oracle Database PL/SQL Language Reference, 11g Release 2 (11.2). URL: https://docs.oracle.com/cd/E11882_01/ appdev.112/e25519 (дата обращения 12.02.2020).

37. Murray C. Oracle Database Express Edition 2 Day DBA, 11g Release 2 (11.2). URL: http://docs.oracle.com/ cd/ E17781_01/server.112/e18804.pdf (дата обращения 17.06.2020).

38. Nanda A. Efficient PL/SQL Coding // Arup Nanda. Oracle Database 11g: The Top New Features for DBAs and Developers. URL: http://www.oracle.com/technetwork/ articles/sql/11g-efficient-coding-093640.html (дата обращения 17.04.2020).

39. Oracle Pipelined Table Functions. URL: https://oracle-base.com/ articles/misc/pipelined-table-functions (дата обращения 06.06.2020).

40. The Notions of Consistency and Predicate Locks in a Database System / K.P. Eswaran [et al.] // Commun. ACM. 1976. Vol. 19, no. 11. Pp. 624–633. DOI: 10.1145/360363.360369.