Министерство образования и науки Российской Федерации

ФГБОУ ВПО «Кубанский государственный технологический университет»

# Кафедра информационных систем и программирования

БАЗЫ ДАННЫХ

## Методические указания по выполнению лабораторных работ

## для студентов всех форм обучения

направления 09.03.03 Прикладная информатика

# Краснодар

2015

Составитель**:** канд. техн. наук, доцент, Янаева Марина Викторовна

Базы данных: методические указания по выполнению лабораторных работ для студентов всех форм обучения направления: 09.03.03 Прикладная информатика. / Сост. М.В. Янаева; Кубан. гос. технол. ун-т. Кафедра информационных систем и программирования. – Краснодар. 2015. – 171 с. Режим доступа: http://moodle.kubstu.ru (по паролю).

Методические указания по выполнению лабораторных работ составлены в соответствии с требованиями к обязательному минимуму содержания дисциплины «Базы данных», входящей в основную образовательную программу подготовки студентов направления 09.03.03 Прикладная информатика государственного образовательного стандарта высшего профессионального образования, и в соответствии с рабочей программой дисциплины.

Ил. 67. Библиогр.: 10 назв.

Рецензенты: Руководитель отдела телекоммуникаций Краснодарского

регионального информационного центра сети «Консультант

Плюс», канд.техн.наук. Н.Ф. Григорьев;

канд. техн. наук, доцент каф. ИСП КубГТУ

А.Г. Мурлин

© ФГБОУ ВПО «КубГТУ», 2015

Содержание

[Введение 4](#_Toc413884444)

[Лабораторная работа №1 - Язык SQL. Проектирование структуры базы данных 6](#_Toc413884445)

[Лабораторная работа №2 - Модификация данных в MS SQL Server 14](#_Toc413884448)

[Лабораторная работа №3 - Выборка данных (оператор SELECT). 18](#_Toc413884451)

[Лабораторная работа №4 - Стандартные соединения и объединения таблиц. 57](#_Toc413884457)

[Лабораторная работа №5 - Представления. 71](#_Toc413884460)

[Лабораторная работа №6 - Хранимые процедуры. 76](#_Toc413884464)

[Лабораторная работа №7 - Триггеры. 81](#_Toc413884468)

[Лабораторная работа №8 - Ограничения. 85](#_Toc413884471)

[Лабораторная работа №9 - Сценарии, пакеты, транзакции, блоки команд. 98](#_Toc413884476)

[Лабораторная работа №10 - Администрирование SQL Server. Определение баз данных и журналов транзакций. Управление доступом к данным 110](#_Toc413884578)

[Лабораторная работа №11 - Администрирование SQL Server. Предоставление разрешений. Архивирование данных 130](#_Toc413884606)

[Лабораторная работа №12 - Восстановление данных. Обмен данными 146](#_Toc413884630)

[Лабораторная работа №13 - Подключение базы данных SQL к проекту VISUAL STUDIO. 159](#_Toc413884640)

[Список рекомендуемой литературы 170](#_Toc413884642)

# Введение

Современные системы управления базами данных (СУБД) основаны на реляционной модели представления данных — в большой степени благодаря простоте и четкости ее концептуальных понятий и строгого математического обоснования. Неотъемлемая и важнейшая часть любой системы, применяющей базы данных, — языковые средства, обеспечивающие возможность доступа и действий над данными, определения их структур, способов использования и интерпретации.

Лабораторные работы предназначены для изучения основ языка SQL - стандартного языка манипулирования данными в СУБД, реализующих реляционную модель данных. Язык SQL распространен повсеместно. Но работать с ним непросто: он сложен, запутан, при написании SQL - команд легко допустить ошибку. Понимание теории, лежащей в основе SQL, - лучший способ гарантировать, что ваш код будет написан правильно, а сама база данных надежна и легко сопровождаема. В методических указаниях демонстрируется, как применить реляционную теорию к повседневной практике работы с SQL. Подробно объясняются различные аспекты этой модели, приводятся многочисленные примеры использования этого языка в соответствии с реляционной теорией. Лабораторные работы не только позволит приобрести начальные знания, но и научат наиболее часто употребляемым мощным средствам языка SQL, используемым опытными программистами. В лабораторных работах демонстрируются SQL –выражения и блоки, различные типы условий, показывается, как посредством соединения таблиц создавать запросы к нескольким таблицам, рассматриваются наборы данных и как они могут взаимодействовать в запросах, демонстрируются встроенные и агрегатные функции, показывается, как и где используются подзапросы. Подробно описаны различные типы соединений таблиц, применение условной логики, работа с транзакциями, индексы и ограничения. Поскольку лучший способ изучения SQL – это практика, в лабораторных работах приводится множество вариантов реальных запросов, охватывающих весь теоретический материал. При этом в определениях и примерах приоритет отдавался простоте и доходчивости материала. При таком подходе не научиться работе с SQL просто невозможно.

При выполнении лабораторных работ должен соблюдаться следующий порядок выполнения работы:

* вначале следует ознакомиться с описанием лабораторной работы;
* получить задание у преподавателя;
* изучить необходимый теоретический материал, пользуясь настоящими указаниями и рекомендованной литературой;
* выполнить задание в СУБД MS SQL Server и среде программирования Visual Studio;
* подготовиться к ответам на теоретические вопросы по теме лабораторной работы;
* оформить отчет.

Все студенты должны предъявить индивидуальный отчет о результатах выполнения лабораторной работы. Допускается предъявление отчета в виде электронного документа.

Отчет должен содержать следующие пункты:

1) Титульный лист.

2) Наименование и цель работы.

3) Краткое теоретическое описание (основные моменты).

4) Задание на лабораторную работу, включающее четкую формулировку задачи.

5) Результаты выполнения работы.

При сдаче отчета студент должен показать знание теоретического материала в объеме, определяемом тематикой лабораторной работы, а также пониманием сущности выполняемой работы.

# Лабораторная работа №1 - Язык SQL. Проектирование структуры базы данных

**1 Цель работы**

Изучить основные принципы работы в СУБД MS SQL Server. Согласно индивидуальному варианту разработать концептуальную схему базы данных и выполнить ее проектирования средствами СУБД.

**2 Основные теоретические сведения**

Язык SQL (Structured Query Language - структурированный язык запросов) представляет собой стандартный высокоуровневый язык описания данных и манипулирования ими в системах управления базами данных (СУБД), построенных на основе реляционной модели данных. SQL относится к классу непроцедурных языков программирования. Программа на языке SQL представляет собой простую линейную последовательность операторов языка SQL. Язык SQL в своем чистом виде операторов управления порядком выполнения запросов к БД (типа циклов, ветвлений, переходов) не имеет.

Операторы языка SQL строятся с применением:

* зарезервированных ключевых слов;
* идентификаторов таблиц и столбцов таблиц;
* логических, арифметических и строковых выражений, используемых для формирования критериев поиска информации в БД и для вычисления значений ячеек результирующих таблиц;
* идентификаторов (имен) операций и функций, используемых в выражениях.

В языке SQL не делается различия между прописными и строчными буквами, т.е., например, строки SELECT, Select, select представляют собой одно и то же ключевое слово. Для конструирования имен таблиц и их столбцов допустимо использовать буквы, цифры и знак подчеркивание, но первым символом имени обязательно должна быть буква. Запрещено использование ключевых слов и имен функций в качестве идентификаторов таблиц и имен столбцов. Оператор начинается с ключевого слова, например CREATE - создать, UPDATE - обновить, SELECT - выбрать, и заканчивается знаком точка с запятой. Оператор записывается в свободном формате и может занимать несколько строк. Допустимыми разделителями лексических единиц в операторе являются:

* один или несколько пробелов,
* один или несколько символов табуляции,
* один или несколько символов типа новая строка.

В приводимых в пособии примерах построения SQL-запросов используется база данных в основном одна и та же база данных «Университет». В ряде отдельных случаев запросы в целях наглядности формируются для таблиц, не входящих данную структуру. В этом случае перед выполнением запроса приводится таблица с исходной информацией.

Итак, спроектируем структуру базы данных «Университет». Для создания базы данных необходимо выбрать команду «Создать базу данных». В открывшемся окне необходимо указать имя БД. Ниже можно указать файлы составляющие проект, а именно: «Файл данных» и «Файл журнала» (рисунок 1).

|  |  |
| --- | --- |
|  |  |

Рисунок 1 – Создание базы данных в MS SQL Server

Для создания таблицы в окне «Обозреватель объектов» выбираем созданную базу данных, и в раскрывшемся списке выбираем пункт «Таблицы», из контекстного меню выбираем пункт «Создать таблицу» (рисунок 2).

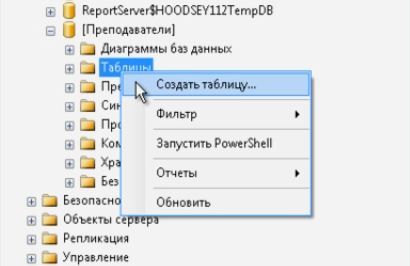


Рисунок 2 – Создание таблицы базы данных

В рабочей области СУБД MS SQL Server откроется форма для создания таблицы, в которой определяем необходимые поля таблицы, тип данных и задаем ключевое поле. Первая колонка – «Имя столбца», вторая колонка – «Тип данных», третья – «Разрешить оставлять поле пустым» (рисунок 3). В столбце «Имя столбца» указывается название поля таблицы, в столбце «Тип данных» – указывается тип столбца. В столбце «Разрешить значения» устанавливается флажок, предоставляющий возможность оставлять ячейку не заполненной (пустой).

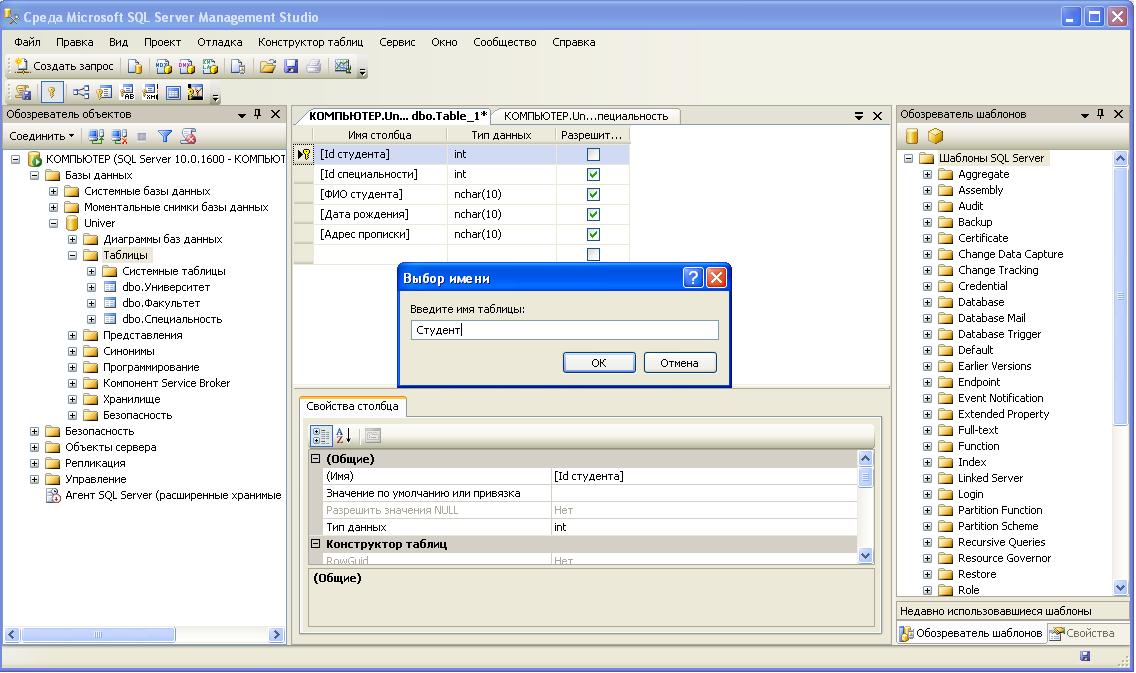
[](file:///C:\Documents%20and%20Settings\Марина.F7A4F5040E59489\Local%20Settings\Temp\RarSFX2\db\tutorial\2\4.jpg)

Рисунок 3 — Проектирование таблицы «Студент»

Ключевое поле создается при помощи команды «Задать первичный ключ» контекстного меню. Для этого необходимо щелкнуть правой кнопкой мыши слева от имени столбца, который будет первичным ключом (рисунок 4).

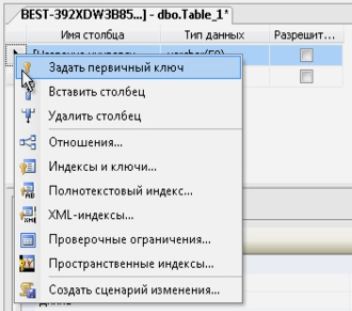


Рисунок 4 – Создание первичного ключа

После создания таблицы ее необходимо сохранить, для этого необходимо из контекстного меню выбрать команду «Сохранить таблицу» и в открывшемся диалоговом окне ввести имя таблицы в соответствующее поле (рисунок 5).

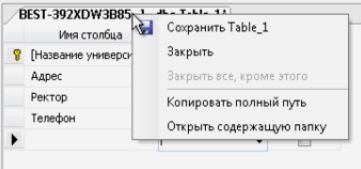


Рисунок 5 – Диалоговое окно сохранения таблицы

Аналогичным образом создаем таблицы «Университет», «Факультет», «Специальность».

После создания таблиц нужно создать связи между ними. Для этого в обозревателе объектов выбираем «Диаграммы баз данных». На всплывшем окне подтверждаем создание необходимых объектов, и в контекстном меню выбираем «Создать диаграмму». Добавляем на диаграмму все раннее созданные таблицы (рисунок 6).

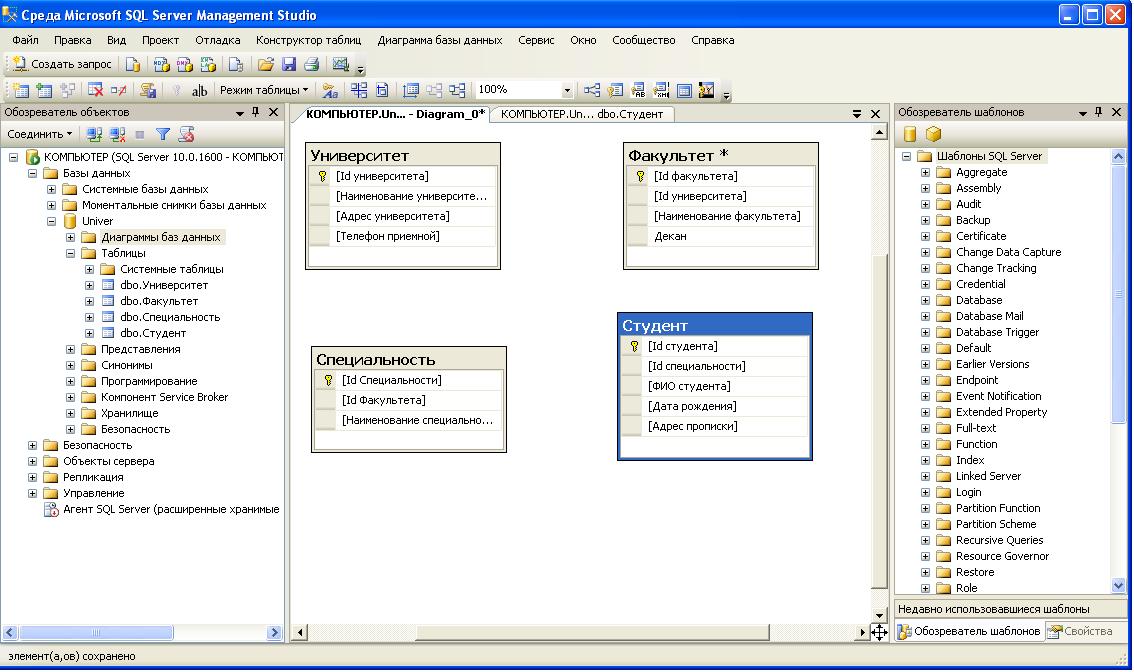
[](file:///C:\Documents%20and%20Settings\Марина.F7A4F5040E59489\Local%20Settings\Temp\RarSFX2\db\tutorial\2\8.jpg)

Рисунок 6 – Создание ER – диаграммы

Создаем связь путем соединения ключевого поля с индексным полем связываемой таблицы, проверяем соответствие полей и нажимаем «ОК». Следует обратить ваше внимание на то, что для установления связей межу таблицами в каждой из них необходимо наличие однотипных полей. Например, таблица «Университет» содержит поле id университета в качестве ключевого и подчиненная ей таблица «Факультет» содержит поле id университета, но уже в качестве простого (индексного) поля. Таким образом, соединение этих двух таблиц по полю id университета позволяет установить между ними тип связи один ко многим.

Аналогичным образом устанавливаем связи между таблицами «Факультет» - «Специальность» по полю id факультета и между таблицами «Специальность» - «Студент» по полю id специальности (рисунок 7).

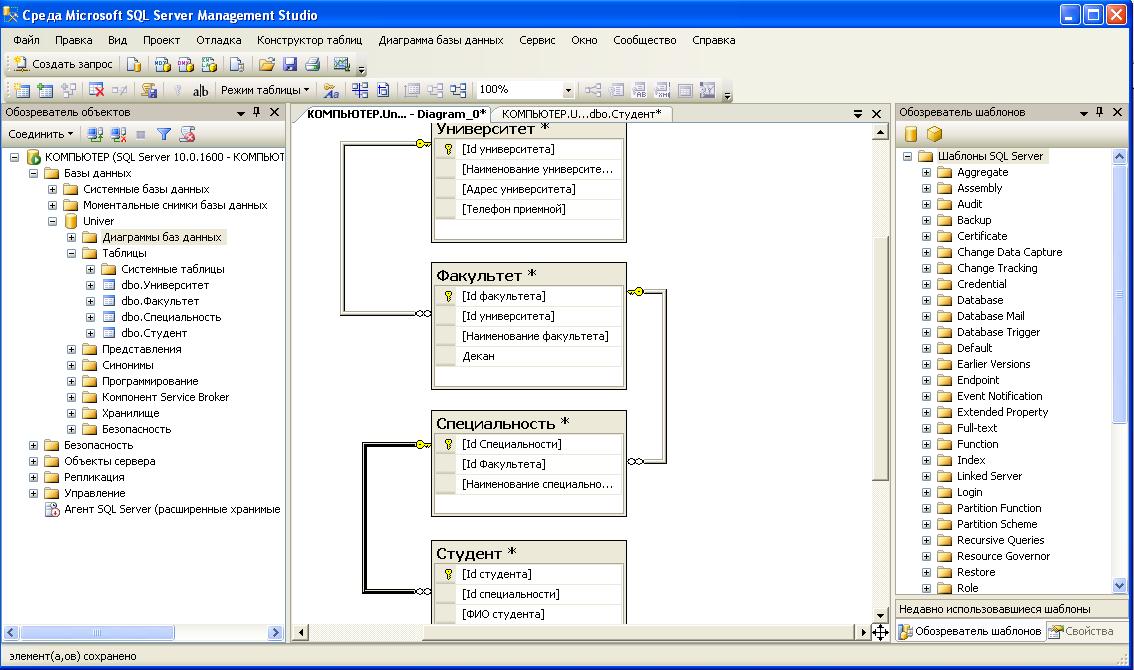
[](file:///C:\Documents%20and%20Settings\Марина.F7A4F5040E59489\Local%20Settings\Temp\RarSFX2\db\tutorial\2\9.jpg)

Рисунок 7 – ER – диаграмма базы данных

В таблице №1 приведены основные типы данных, необходимые при проектировании полей таблиц базы данных.

Таблица №1 - Системные типы данных SQL Server

| **№** | **Тип** | **Описание, занимаемый объем** |
| --- | --- | --- |
| 1 | Bynary | Любые данные в двоичном виде. Используется для хранения файла, до 1,2 Гб |
| 2 | Varbinary | Любые данные в двоичном виде. Используется для хранения файла, до 1,2 гб |
| 3 | Tinyint | Целое положительное число от 0 до 255, 1 байт |
| 4 | Char | Символьное выражение; может содержать любые символы (до 254 для одного поля), 1 байт на символ |
| 5 | Varchar | Символьное выражение; может содержать любые символы (до 254 для одного поля) |
| 6 | Money | Денежное выражение для числовой величины. Выводит число с четырьмя десятичными разрядами и установленным обозначением используемой денежной единицы, 8 байт |
| 7 | Datetime | Дата и время; может содержать время, день, месяц и год, точность 3,33 миллисекунды, 8 байт |

Продолжение таблицы №1

|  |  |  |
| --- | --- | --- |
| 8 | Smalldatetime | Дата и время; точность 2 минута, 4 байта |
| 9 | Bit | Булево выражение ( .T. или .F.), 1 байт |
| 10 | Float | Числовое выражение; может содержать целые или дробные числа со знаком, От 1 до 20 байтов |
| **№** | **Тип** | **Описание, занимаемый объем** |
| 11 | Smallint | Целое число в диапазоне от –32 768 до +32 767, 2 байта |
| 12 | Int | Целое число. Можно хранить числа от –2 147 483 647 до 2 147 483 646, 4 байта |
| 13 | Image | Поле для ссылки на объект OLE, 4 байта |
| 14 | Text | Поле примечаний для ссылки на блок данных, 4 байта |
| 15 | Decimal | Точное числовое значение, которое может иметь до 38 цифр (р); число цифр после запятой (s) не может превышать р |
| 16 | Numeric | Точное числовое значение, которое может иметь до 38 цифр (р); число цифр после запятой (s) не может превышать р |
| 17 | Real | Число с плавающей точкой одинарной точности. Можно хранить числа в интервале от 1.401298Е-45 до 3.402823Е38, 4 байта |

Следует отметить, что одна из задач проектирования базы состоит в обеспечения способа идентификации различных объектов, другими словами система должна отличать друг от друга отдельные строки таблицы. Первичный ключ это столбец, однозначно определяющий строку в таблице. В случае, когда в таблице существует несколько одинаковых значений, уникальный первичный ключ поможет их различить.

Для связи данных между таблицами необходимо создать отношения. Отношения бывают:

* **Один к одному** - определение связи «один к одному» полностью соответствует ее названию. Связью «один к одному» называется такая связь, из наличия которой следует, что если имеется какая-то одна строка в одной таблице, то должна быть точно одна соответствующая ей строка в другой таблице.
* **Один ко многим** - Связью «один ко многим» называется такая связь, из наличия которой следует, что если имеется какая-то одна строка в одной таблице, то должны быть несколько соответствующих ей строк в другой таблице.
* **Многие ко многим** - Связь «многие ко многим» характеризуется тем, что на обеих сторонах связи может присутствовать несколько согласующихся строк, а не только одна.

Связь «Один ко многим» является типичной связью между таблицами, где одна таблица по вторичному ключу ссылается на вторую таблицу с индексным ключом.

Связь «Многие ко многим» характеризуется тем, что на обеих сторонах связи может присутствовать несколько согласующих строк.

# 3 Задания для самостоятельной работы

Создайте базу данных. Создайте журнал транзакций и назначьте устройство журналу транзакций для вашей базы данных. Разработайте и создайте необходимый набор таблиц. Заполните таблицы в соответствии с заданием. Постройте ER – диаграмму вашей базы данных.

**Варианты заданий:**

1. База данных - «Изделие на складе». Таблицы: «Изделие», «Поставщик», «Потребитель», «Заказ».
2. База данных - «Университет». Таблицы: «Университет», «Факультеты», «Кафедры», «Предметы».
3. База данных - «Вуз». Таблицы: «Вуз», «Специальность», «Прием студентов».
4. База данных - «Специальности вуза». Таблицы: «Факультет», «Специальность», «Группа студентов».
5. База данных - «Поликлиника». Таблицы: «Поликлиника», «Врач», «Пациент», «Заболевание».
6. База данных – «Биржа труда». Таблицы: «Вакансии», «Организации», «Трудоустраиваемый».
7. База данных – «Автостраховая компания». Таблицы: «Автомобиль», «Владелец», «Страховка».
8. База данных - «Сессия». Таблицы: «Факультет», «Группа», «Студент», «Экзамен».
9. База данных - «Подписка». Таблицы: «Издание», «Подписка», «Подписчик».
10. База данных - «Школа». Таблица: «Школа», «Класс», «Ученик».
11. База данных – «ГАИ». Таблицы: «Автомобиль», «Владелец», «Техосмотр».
12. База данных - «Склад товаров». Таблицы: «Товар», «Потребитель», «Поставщик», «Поставка».
13. База данных - «Грузоперевозка». Таблицы: «Грузоотправитель», «Грузополучатель», «Груз».
14. База данных - «Налоги». Таблицы: «Предприятие», «Сотрудники», «Инспектор».
15. База данных – «Общежитие». Таблицы: «Общежитие», «Комната», «Студент».
16. База данных - «Недвижимость». Таблицы: «Агентство», «Объект продажи», «Покупатель», «Продавец».
17. База данных – «Книжный магазин». Таблицы: «Издания», «Издательство», «Склад», «Отдел».
18. База данных – «Медицинское учреждение». Таблицы: «Пациент», «Отделение», «Врач», «Пациент».
19. База данных - «РЭП». Таблицы: «РЭП», «Квартирный дом», «Квартира», «Жильцы».
20. База данных - «Производство изделия». Таблицы: «Изделие», «Заявка на изготовление», «Потребитель».

# 4 Контрольные вопросы

1. Что такое ядро базы данных?
2. Что такое устройство баз данных?
3. Приведите синтаксис команды создания устройства.
4. Что представляет собой журнал транзакций?
5. Перечислите основные достоинства языка SQL?
6. Дайте определение логическому имени.
7. Приведите пример команды создания базы данных.
8. Какие системные и числовые типы данных MS SQL Server вам знакомы.
9. Приведите пример команды создания таблицы базы данных.

# Лабораторная работа №2 - Модификация данных в MS SQL Server

**1 Цель работы**

Ознакомиться со структурой и синтаксисом основных команд модификации данных. Реализовать основные команды модификации на своем примере базы данных.

**2 Основные теоретические сведения**

ВSQL для выполнения операций ввода данных в таблицу, ихизменения и удаления предназначены три команды языка манипулирования данными: INSERT (вставить), UPDATE (обновить), DELETE (удалить).

**INSERT** — оператор языка SQL, который позволяет добавить строки в таблицу, заполняя их значениями. Значения можно вставлять перечислением с помощью слова values и перечислив их в круглых скобках через запятую или оператором select.

Команда INSERT осуществляет вставкув таблицу новой строки. Впростейшем случае она имеет вид:

INSERT INTO <имя таблицы> VALUES (<значение>, <значение>,);

При такой записи указанные в скобках после ключевого слова VALUES значения вводятся в поля добавленной в таблицу новой строки в том порядке, вкотором соответствующие столбцы указаны при создании таблицы.

Например, ввод новой строки в таблицу «Университет» может быть осуществлен следующим образом:

INSERT INTO [Univer].[dbo].[Университет]

([id университета],

[Наименование университета],

[Адрес университета],

[Телефон приемной])

VALUES (‘666’, ‘КубГТУ’, ‘Красная’, ‘2500245’)

Если вам нужно ввести пустое значение (NULL), вы вводите его точно так же, как и обычное значение.

Например: INSERT INTO Университет VALUES ('100', 'КубГТУ', 'Null','Null')

Вслучаях, когда необходимо ввести значения полей в порядке, отличном от порядка столбцов или требуется ввести значения не во все столбцы, следует использовать следующую форму команды INSERT:

INSERT INTO Университет ([id университета], [Наименование университета]) VALUES (101, 'КубГТУ');

Столбцам, наименования которых не указаны в приведенном в скобках списке, автоматически присваивается значение по умолчанию, если оно назначено при описании таблицы, либо значение NULL.

С помощью команды INSERT можно извлечь значение из одной таблицы и разместить его в другой, например, запросом следующего вида:

INSERT INTO [Факультет]

SELECT \*

FROM Университет

WHERE [id университета]= **'101';**

**Использование подзапросов в INSERT**

Применение оператора INSERT с подзапросом позволяет загружать сразу несколько строк в одну таблицу, используя информацию из другой таблицы. В то время как оператор INSERT, использующий VALUES, добавляет только одну строку, INSERT с подзапросом добавляет в таблицу столько строк, сколько подзапрос извлекает из другой таблицы. При этом количество и тип возвращаемых подзапросом столбцов должно соответствовать количеству и типу столбцов таблицы, в которую вставляются данные.

Например, пусть таблица Студент1 имеет структуру, полностью совпадающую со структурой таблицы Студент. Запрос, позволяющий заполнить таблицу Студент1 записями обо всех студентах с id специальности = 230101 из таблицы Студент, выглядит следующим образом:

INSERT INTO Студент1

SELECT \*

FROM Студент

WHERE [id специальности] = **'230101';**

**Использование подзапросов, основанных на таблицах внешних запросов**

Предположим, существует таблица Студенты, в которой хранятся сведения о студентах, обучающихся в том же городе, в котором они живут. Можно заполнить эту таблицу данными из таблицы Студент, используя связанные подзапросы, следующим образом:

INSERT INTO Студенты

SELECT \*

FROM Студент А

WHERE Город IN

(SELECT Город

FROM Университет В

WHERE A.[id университета] = B. [id университета]);

Удалениестрок из таблицы осуществляется с помощью команды **DELETE**. Для удаления из таблицы сразу нескольких строк, удовлетворяющих некоторому условию, можно воспользоваться предложением WHERE, например:

DELETE Univer.dbo.Университет

Where [id университета]=’666’;

Данный пример удалит строку в таблице «Университет», где «Id университета» будет равен «666».

или

DELETE Univer.dbo.Университет

Where [id университета] in

(Select [id университета] from Univer.dbo.[Университет] where [id университета] in (666));

**Использование подзапросов с DELETE**

Пусть филиал университета в Сочи ликвидирован и требуется удалить из таблицы Студент записи о студентах, которые там учились. Эту операцию можно выполнить с помощью запроса:

DELETE

FROM Студент

WHERE [id университета] IN

(SELECT [id университета]

FROM Университет

WHERE Город= **'Сочи'**);

В предикате предложения FROM (подзапроса) нельзя ссылаться на таблицу, из которой осуществляется удаление. Однако можно ссылаться на текущую строку из таблицы, являющуюся кандидатом на удаление, то есть на строку, которая в настоящее время проверяется в основном предикате.

**UPDATE** — оператор языка SQL, позволяющий обновить значения в заданных столбцах таблицы.

UPDATE <объект> SET <присваивание1 [, присваивание2, ...]> [WHERE <условие>] [OPTION <х1 [, х2, ...]>] <объект> — объект, над которым выполняется действие (таблица или представление(views)) <присваивание> — присваивание, которое будет выполняться при каждом выполнении условия <условие>, или для каждой записи, если отсутствует раздел where <условие> — условие выполнения команды <х> — инструкция программе как исполнить запрос.

Пример выполнения:

UPDATE [Univer].[dbo].[Факультет]

SET [Наименование факультета]=’Механический’

FROM [Univer].[dbo].[Факультет], [Univer].[dbo].[Университет]

WHERE ([Университет].[Наименование университета] = ‘КубГТУ’) and ([Факультет].[Наименование факультета]=’Механический’);

# 3 Задания для самостоятельной работы

1. Осуществите добавление нескольких записей в ваши таблицы при помощи команды INSET.
2. Осуществите вставку пустого значения в одну из таблиц.
3. Удалите по одной записи из каждой таблицы при помощи команды DELETE.
4. Осуществите удаление записи с вложенным подзапросом из любой таблицы вашей базы данных.
5. Осуществите модификацию данных ваших таблиц при помощи команды UPDATE.
6. Осуществите модификацию данных любой таблицы с помощью команды UPDATE с предложением FROM.
7. Осуществите обновление данных в нескольких таблицах.

## 4 Контрольные вопросы

1. Перечислите основные команды MS SQL для модификации данных.
2. Приведите пример использования команды INSET для вставки записей в единственную таблицу.
3. Каким образом осуществляется вставка пустого значения в ячейку таблицы?
4. Приведите пример удаления единственной записи таблицы.
5. Приведите пример удаления множества записей.
6. Каким образом осуществляется удаление с вложенным подзапросом?
7. При помощи какой команды осуществляется модификация данных в MS SQL Server?
8. Приведите пример использования команды UPDATE с предложением FROM.
9. Приведите пример обновления множества записей.
10. Приведите пример обновления записей с подзапросом.
11. Приведите пример обновления записей нескольких таблиц.
12. Можно ли использовать скалярные выражения для модификации данных?
13. Для чего используется оператор ALTER.
14. Приведите пример удаления выбранного объекта MS SQL Server.

# Лабораторная работа №3 - Выборка данных (оператор SELECT).

**1 Цель работы**

Ознакомиться со структурой и синтаксисом оператора выборки данных. Выполнить реализацию основных типов запросов на примере своей СУБД.

**2 Основные теоретические сведения**

Оператор SELECT (выбрать) языка SQL является самым важным и самым часто используемым оператором. Он предназначен для выборкиинформации из таблиц базы данных. Синтаксис оператора SELECT выглядит следующим образом:

SELECT [DISTINCT] <список атрибутов>

FROM <список таблиц>

[WHERE <условие выборки>]

[ORDER BY <список атрибутов>]

[GROUP BY <список атрибутов>]

[HAVING <условие>]

[UNION <выражение с оператором SELECT>J;

Вквадратных скобках указаны элементы, которые могут отсутствовать в запросе. Ключевое слово SELECT сообщает базе данных, что данное предложение является запросом на ***извлечение*** информации. После слова SELECT через запятую перечисляются ***наименования полей*** (список атрибутов), содержимое которых запрашивается. Обязательным ключевым словом в предложении-запросе SELECT является слово FROM. За ключевым словом FROM указывается список разделенных запятыми имен таблиц, из которых извлекается информация.

**Например:**

SELECT [Фамилия Имя Отчество], [Дата рождения], [Адрес прописки]

FROM [Univer].dbo.Студент;

Приведенный запрос осуществляет выборку всех значений «Фамилия Имя Отчество», «Дата рождения», «Адрес прописки» из таблицы «Студент».

Если необходимо вывести значения всех,столбцов таблицы, то можно вместо перечисления их имен использовать символ **«\*»** (звездочка).

**Например:**

SELECT \*

FROM Студент;

В данном случае результатом выполнения запроса будет вся таблица Студент. Следует заметить, что результат данного запроса может содержать строки с одинаковыми значениями атрибутов. Для исключения из результата SELECT - запроса повторяющихся записей используется ключевое слово DISTINCT (отличный). Если запрос SELECT извлекает множество полей, то DISTINCT исключаетдубликаты строк, в которых значения всехвыбранных полей идентичны.

**Например:**

SELECT DISTINCT [Адрес прописки]

FROM Студент;

Ключевое слово ALL (все), в отличие от DISTINCT, оказывает противоположное действие, то есть при его использовании повторяющиеся строки включаютсяв выходные данные. Режим, задаваемый ключевым словом ALL, действует по умолчанию, поэтому в реальных запросах для этих целей оно практически не используется.

**Например:**

Напишите данный запрос выбирает все данные из таблицы STUDENT, расположив столбцы таблицы в следующем порядке: KURS, SERNAME, NAME, STIPEND.

select all KURS, SERNAME, NAME, STIPEND

from STUDENT;

Использование в операторе SELECT предложения, определяемого ключевым словом WHERE (где), позволяет задавать какие строкиуказанных таблиц должны быть выбраны*.* В таблицу, являющуюся результатом запроса, включаются только те строки, для которых условие, указанное в предложении WHERE, принимает значение истина.

**Например**: запрос выводит запрашиваемые поля определенного студента, номер которого равен «101» из таблицы «Студент» базы данных рассмотренной в теме №1.

SELECT [ФИО Студента], [Дата Рождения], [Адрес прописки]

FROM [Univer].[dbo].[Студент]

WHERE [id студента]=101;

Результатом запроса будет следующая информация:

|  |  |  |
| --- | --- | --- |
| ФИО | Дата рождения | Адрес прописки |
| Иванов Николай Павлович | 10.09.1980 | Краснодар ул. Коммунаров 21 |

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

**Например:**

SELECT [Фамилия Имя Отчество], [Дата рождения], [Адрес прописки]

FROM [Univer].dbo.Студент

WHERE [Id студента] <> 101;

Результатом выполнения запроса будет информация обо всех студентах, Id которых не равен 101.

**Например:** следующий запрос к таблице STUDENT выводит список фамилий (SURNAME), имен (NAME), номеров курсов (KURS) всех студентов со стипендией, большей или равной 100 и живущих в Воронеже.

select SERNAME, NAME, KURS

from STUDENT

where STIPEND > 100 and CITY = 'Воронеж';

При задании логического условия в предложении WHERE могут быть использованы операторы IN, BETWEEN, LIKE, is NULL. Операторы IN (равен любому из списка) и NOT IN (не равен ни одному из списка) используются для сравнения проверяемого значения поля с заданным списком. Этот список значений указывается в скобках справа от оператора IN. Построенное с использованием IN условие считается истинным, если значение поля, имя которого указано слева от IN, в точности совпадает с одним из значений, перечисленных в списке, указанном в скобках справа от IN. Условие, построенное с использованием NOT IN, считается истинным, если значение поля, имя которого указано слева от NOT IN, не совпадаетни с одним из значений, перечисленных в списке, указанном в скобках справа от NOT IN.

**Например:**

Получим из таблицы Студент сведения о студентах, имеющихId специальности 230101 и 080801.

SELECT \*

FROM Студент

WHERE [Id специальности] IN (230101, 080801);

Получим сведения о студентах, не имеющихId специальности 230101 и 080801.

SELECT \*

FROM Студент

WHERE [Id специальности] NOT IN (230101, 080801);

Оператор BETWEEN используется для проверки условия вхождения значения поля в заданный интервал, то есть вместо списка значений атрибута этот оператор задает границы его изменения.

**Например**:

Получим сведения о студентах, не имеющихId студента в пределах между 50 и 102.

SELECT \*

FROM Студент

WHERE [Id студента] BETWEEN 50 AND 102;

**Например:**

Получим запрос для вывода находящихся в таблице EXAM\_MARKS (экзаменационные оценки) номеров предметов обучения, экзамены по которым сдавались между 10 и 20 января 2012 года.

SELECT [SUBJ ID]

FROM [EXAM MARKS]

WHERE [EXAM DATE] between 2012-01-20 and 2012-01-10;

Оператор BETWEEN может использоваться как для числовых, так и для символьных типов полей.

Оператор LIKE применим только к символьным полям типа CHAR или VARCHAR. Этот оператор просматривает строковые значения полей с целью определения, входит ли заданная в операторе LIKE подстрока в символьную строку – значение проверяемого поля. Для выборки строковых значений по заданному образцу подстроки можно применять шаблон искомого образца строки, использующий следующие символы:

* символ подчеркивания «\_», указанный в шаблоне, определяет возможность наличия в указанном месте одного любого символа;
* символ «%» допускает присутствие в указанном месте проверяемой строки последовательности любых символов произвольной длины.

**Например:**

Осуществим выборку информации из таблицы «Студент» всех студентов, фамилии которых начинаются на букву «Н»

SELECT \*

FROM Студент

WHERE [Фамилия Имя Отчество] LIKE ‘Н%’;

Следующий запрос выбирает сведения о студентах, у которых имена начинаются на буквы «И» или «С».

SELECT \*

FROM Студент

WHERE [Фамилия Имя Отчество] like 'С%' or [Фамилия Имя Отчество] like 'И%';

Запрос выполняет вывод фамилии первого в алфавитном порядке (по фамилии) студента, фамилия которого начинается на букву «И».

select SERNAME

from STUDENT

group by SERNAME

having SERNAME like 'И%';

Следует отметить, что операторы сравнения «=, <, >, <=, >=, <>» и операторы IN, BETWEEN и LIKE нельзя использовать для проверки содержимого поля на наличие в нем пустого значения NULL. Для этих целей предназначены специальные операторы is NULL (является пустым) и IS NOT NULL (является не пустым).

**Например:**

Пусть имеется таблица EXAM\_MARKS, содержащая сведения об экзаменационных оценках. Следующий запрос предназначен для выбора из таблицы EXAM\_MARKS записей, в которых отсутствуют значения оценок (поле MARK).

select \*

from [EXAM MARKS]

where MARK is null;

**Например:**

Следующий запрос предназначен для вывода из таблицы EXAM\_MARKS записей, имеющих в поле MARK значения оценок.

select \*

from [EXAM MARKS]

where MARK is not null;

# Использование агрегирующих функций.

Агрегирующие функции позволяют получать из таблицы сводную информацию, выполняя операции над группой строк таблицы. Для задания в SELECT – запросе агрегирующих операций используются следующие ключевые слова:

* COUNT определяет количество строк или значений поля, выбранных посредством запроса и не являющихся NULL-значениями;
* SUM вычисляет арифметическую сумму всех выбранных значений данного поля;
* AVG вычисляет среднее значение для всех выбранных значений данного поля;
* МАХ вычисляет наибольшее из всех выбранных значений данного поля;
* MIN вычисляет наименьшее из всех выбранных значений данного поля.

Аргументами агрегатных функций могут быть как столбцы таблицы, так и результаты выражений над ними. При этом выражение может быть сколько угодно сложным.

**Функция AVG ().**

Данные, содержащиеся в таблице должны иметь числовой тип, потому что функция AVG вначале суммирует все значения в столбце, а затем делит полученную сумму на количество этих значений.

**Функция SUM().**

Для вычисления суммы значений, содержащихся в столбце, используется функция SUM. При этом столбец должен иметь числовой тип данных, потому что результат, возвращаемый этой функцией, имеет тот же тип данных, что и столбец.

**Функции MAX и MIN.**

Для нахождения максимального и минимального значения в столбце, используются агрегатные функции MAX и MIN. При этом столбец может содержать числовые и строковые значения, либо значения даты/времени.

**Функция COUNT.**

Данная функция подсчитывает количество значений в столбце, при этом тип данных может быть любым.

**Функции MAXVALUE и MINVALUE** в чем-то похожи на агрегатные функции MAX и MIN, однако выбирают максимальное и минимальное значение не из множества значений строк в одном столбце, а из значений, заданных в списке аргументов.

**Функция CASE**

Выражение CASE позволяет возвращать разные значения в зависимости от условия.

**Функция COALESCE**

Возвращает первое выражение из списка аргументов, не равное NULL.

**Функция NULLIF**

Данная функция позволяет при выборке делать замену заданного значения на NULL.

Приведем примеры использования агрегированных функций.

Пусть имеется таблица с экзаменационными оценками студентов EXAM MARKS .

Запрос предназначен для подсчета количества студентов, сдававших экзамен по предмету обучения с идентификатором, равным 22.

select count ([STUDENT ID])

from [EXAM MARKS]

where [SUBJ ID] = 22;

Запрос позволяет подсчитать в таблице EXAM\_MARKS количество различных предметов обучения.

select count(DISTINCT[SUBJ ID])

from [EXAM MARKS];

Запрос выполняет выборку для каждого студента значения его идентификатора и минимальную из полученных студентом оценок.

select [STUDENT ID], MIN(MARK)

from [EXAM MARKS]

group by [STUDENT ID];

Запрос, осуществляет выборку для каждого студента значения его идентификатора и максимальную из полученных им оценок.

select [STUDENT ID], MAX(MARK)

from [EXAM MARKS]

group by [STUDENT ID];

Запрос выполняет вывод (для каждого предмета обучения) наименования предмета и максимального значения номера семестра, в котором этот предмет преподается.

select [SUBJ NAME], MAX (SEMESTER)

from SUBJECT

group by [SUBJ NAME], SEMESTER;

Запрос выполняет вывод данных для каждого конкретного дня сдачи экзамена о количестве студентов, сдававших экзамен в этот день.

select [EXAM DATE], COUNT ([STUDENT ID])

from [EXAM MARKS]

group by [EXAM DATE],[STUDENT ID];

Запрос получения среднего балла каждого курса по каждому предмету.

select DISTINCT KURS, AVG (CAST(MARK AS float))

from [EXAM MARKS]

cross join STUDENT

group by KURS, MARK;

Запрос получения среднего балла для каждого студента.

select DISTINCT ([STUDENT ID]), AVG (CAST (MARK AS float))

from [EXAM MARKS]

group by [STUDENT ID], MARK;

Запрос получения среднего балла по каждому экзамену.

select DISTINCT ([EXAM ID]), AVG (CAST(MARK AS float))

from [EXAM MARKS]

group by [EXAM ID], MARK ;

Запрос для определения количества студентов, сдававших каждый экзамен.

select DISTINCT([EXAM ID]), COUNT([STUDENT ID])

from [EXAM MARKS]

group by [EXAM ID], [STUDENT ID] ;

Рассмотрим ряд примеров с использованием пустых значений (NULL) в агрегирующих функциях и упорядочивания выходных полей с использованием конструкции ORDER BY.

Предположим, что стипендия всем студентам увеличена на 20%. Напишем запрос к таблице STUDENT, выполняющий вывод номера студента, фамилию студента и величины увеличенной стипендии. Выходные данные упорядочить: а) по значению последнего столбца (величине стипендии); б) в алфавитном порядке фамилий студентов.

а)

select [STUDENT ID], NAME, SERNAME, STIPEND \* 100

from STUDENT

order by STIPEND;

б)

select [STUDENT ID], NAME, SERNAME, STIPEND \* 100

from STUDENT

order by SERNAME;

Напишем запрос, который по таблице EXAM\_MARKS позволяет найти а) максимальные и б) минимальные оценки каждого студента выводя их вместе с идентификатором студента.

а)

select [STUDENT ID], MAX(MARK)

from [EXAM MARKS]

group by [STUDENT ID], MARK;

б)

select [STUDENT ID], MIN(MARK)

from [EXAM MARKS]

group by [STUDENT ID], MARK;

Напишем запрос, выполняющий вывод списка предметов обучения в порядке а) убывания семестров и б) возрастания отводимых на предмет часов. Поле семестра в выходных данных должно быть первым, за ним должны следовать имя предмета обучения и идентификатор предмета.

а)

select SEMESTER, [SUBJ NAME], [SUBJ ID]

from SUBJECT

order by SEMESTER DESC;

б)

select SEMESTER, [SUBJ NAME], [SUBJ ID]

from SUBJECT

order by HOUR;

Напишем запрос, который выполняет вывод суммы баллов всех студентов для каждой даты сдачи экзаменов и представляет результаты в порядке убывания выбранных сумм.

select [EXAM DATE], SUM(CAST(MARK AS float))

from [EXAM MARKS]

group by [EXAM DATE], MARK

order by MARK DESC;

Напишем запрос, который выполняет вывод а) среднего, б) минимального, в) максимального баллов всех студентов для каждой даты сдачи экзаменов и представляет результаты в порядке убывания этих значений.

а)

select [EXAM DATE], AVG(CAST(MARK AS float))

from [EXAM MARKS]

group by [EXAM DATE], MARK

order by MARK DESC;

б)

select [EXAM DATE], MIN(MARK)

from [EXAM MARKS]

group by [EXAM DATE], MARK

order by MARK DESC;

в)

select [EXAM DATE], MAX(MARK)

from [EXAM MARKS]

group by [EXAM DATE], MARK

order by MARK DESC;

# Использование числовых функций

Числовые функции возвращают числовые значения на основании значений типа заданного в аргументе. Числовые функции используются для обработки данных, а также в условиях поиска. Основные типы числовых функций приведены в таблице

Таблица №2 – Использование числовых функций

|  |  |
| --- | --- |
| **Тип функции** | **Запрос** |
| **1** | **2** |
| RAND() - возвращает случайное число от 0 до 1 | SELECT RAND () |
| ABS (число) – возвращает абсолютное значение | SELECTABS(-3) |
| SIGN (число) - знаковая функция (возвращает 1 для положительного числа, 0 – для нуля, 1 –для отрицательного числа). | SELECT SING(5) |

Продолжение таблицы №2

|  |  |
| --- | --- |
| **1** | **2** |
| LOG (основание, число) - логарифм числа по указанному основанию EXP (число) - экспоненциальная функция («e» в степени аргумента) | SELECT LOG (EXP(10)) |
| LOG10 (число) - десятичный логарифм числа | SELECT LOG10(100) |
| PI() - константа π = 3.1459... | SELECT PI() |
| POWER (число, степень) - возведение числа в степень | SELECT POWER(4, 2) |
| SQRT (число) – возвращает квадратный корень | SELECT SQRT(4) |
| FLOOR (число) - округление до целого числа вниз | SELECT FLOOR(4,5) |
| CEILING (число) - округление до целого числа вверх | SELECT CEILING (4,5) |
| ROUND (число, точность) - округление до указанного количества знаков после запятой | SELECT ROUND(4.5555, 2) |
| SIN (число) - синус (аргумент задается в радианах) | SELECT SIN(0,5) |
| COS (число) - косинус (угол определяется в радианах, результат в диапазоне от -1 до 1) | SELECT COS(0,5) |
| TAN (число) - тангенс (аргумент задается в радианах) | SELECT TAN(0,5) |
| COT (число) – котангенс | SELECT COT(0,5) |
| ASIN (число) - арксинус (число должно быть в диапазоне от -1 до 1, результат от –π/2 до π/2) | SELECT ASIN(0,5) |
| ACOS (число) арккосинус (число должно быть в диапазоне от -1 до 1, результат от 0 до π) | SELECT ACOS(0,5) |
| ATAN (число) - арктангенс (возвращает результат в диапазоне от –π/2 до π/2) | SELECT ATAN(0,5) |

Арифметические операции для преобразования числовых данных делятся на:

* унарный (одиночный) оператор «—» (знак минус): изменяет знак числового значения, перед которым он указан, на противоположный.
* бинарные операторы «+», «—», «\*» и «/»: предоставляют возможность выполнения арифметических операций сложения, вычитания, умножения и деления.

**Пример** - выборка с математической операцией для таблицы студент, рассмотренной в теме №1.

SELECT [Фамилия Имя Отчество]=[ФИО студента], [Дата рождения], [Номер специальности]=[id специальности]+1

FROM [Univer].[dbo].[Студент]

WHERE [id студента]<>101

**Строковые функции.**

Строковые функции используют в качестве аргумента строку символов, а в качестве результата возвращают символьную строку или числовое значение.

Для выделения подстроки указанной длины из строкового выражения, начиная с заданной позиции, используется функция **SUBSTRING**, имеющая следующий формат:

SUBSTRING (<строковое\_выражение> FROM позиция [FOR длина]),

где позиция – позиция, начиная с которой выполняется выделение, например 1 для первого символа в строке; длина – количество выделяемых символов.

**Например**, для вывода id студента и первых трех символов их фамилии можно использовать следующий запрос:

SELECT [id студента], SUBSTRING ([ФИО студента] FROM 1 for 3) AS Fio3

FROM [Univer].[dbo].[Студент]

**Операция конкатенации строк**

Операция конкатенации «+» позволяет соединять («склеивать») значения двух или более столбцов символьного типа или символьных констант в одну строку. Эта операция имеет синтаксис: «значимое символьное выражение» + «значимое символьное выражение».

**Функции преобразования символов в строке:**

* **LOWER** — перевод в строчные символы (нижний регистр): LOWER (строка)
* **UPPER** — перевод в прописные символы (верхний регистр): UPPER (строка)
* **INITCAP** — перевод первой буквы каждого слова строки в прописную (заглавную): INITCAP (строка)

Приведем примеры запросов.

Составим запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала один столбец, содержащий последовательность разделенных символом «;» значений всех столбцов этой таблицы, и при этом текстовые значения должны отображаться прописными символами (верхний регистр), то есть быть представленными в следующем виде:

10; КУЗНЕЦОВ; БОРИС; 0; БРЯНСК; 8/12/1981; 10.

select UPPER ([STUDENT ID] + ';'+ SERNAME +';'+ NAME +';'+ STIPEND +';'+ KURS +';'+ CITY)

from STUDENT;

Результаты запроса представлены на рисунке 8.

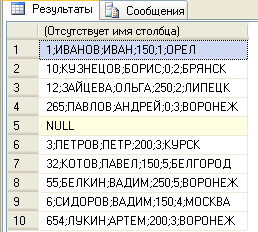


Рисунок 8 – Результаты выполнения запроса

с использованием функции UPPER

Составим запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала всего один столбец в следующем виде:

б.кузнецов; место жительства-брянск; родился:8-дек-1981.

select LOWER(SUBSTRING(NAME, 1, 1) + '.' + SERNAME) +'; '

+ 'местожительства' + ' - ' + LOWER(CITY) + '; ' + 'родился' + ' - '

+ CAST(BIRTHDAY AS char(11))

from STUDENT

where SERNAME = 'Кузнецов';

Результаты запроса представлены на рисунке 9.

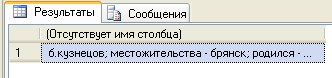


Рисунок 9 – Результаты выполнения запроса

с использованием функции LOWER

Составим запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала всего один столбец в следующем виде:

Борис Кузнецов родился в 1981 году.

select NAME + ' ' + SERNAME + ' родился ' + CAST(BIRTHDAY AS varchar)

from STUDENT

where SERNAME = 'Кузнецов';

Результаты запроса представлены на рисунке 10.

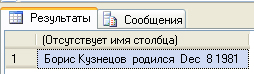


Рисунок 10 – Результаты выполнения запроса

Отобразить информацию для студентов 1, 2 и 4-го курсов таким образом, чтобы фамилии и имена были выведены прописными буквами.

select UPPER (NAME + ' ' + SERNAME) + ' родился ' + CAST(BIRTHDAY AS varchar)

from STUDENT

where KURS IN (1, 2, 4);

Результаты запроса представлены на рисунке 11.

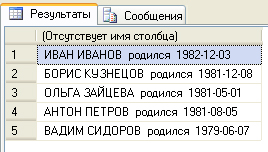


Рисунок 11 – Результаты выполнения запроса

Составим запрос для таблицы UNIVERSITY таким образом, чтобы выходная таблица содержала всего один столбец в следующем виде:

Код-10; ВГУ-г. ВОРОНЕЖ; Рейтинг=296.

SELECT 'Код' + '-' + [UNIV ID] + '; '

+ [UNIV NAME] + ' - ' + 'г.' + UPPER (CITY) + '; ' + 'Рейтинг =' + RATING

from UNIVERSITY

where [UNIV ID] = 10;

Результаты запроса представлены на рисунке 12.

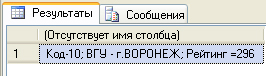


Рисунок 12 – Результаты выполнения запроса

Уточним условие предыдущего примера с условием, что значения рейтинга требуется округлить до первого знака (например, значение 382 округляется до 400).

SELECT 'Код' + '-' + [UNIV ID] + '; '

+ [UNIV NAME] + ' - ' + 'г.' + UPPER (CITY) + '; '

+ 'Рейтинг =' + ROUND(RATING,-1)

from UNIVERSITY

where [UNIV ID] = 10;

Для выделения строки в обратной последовательности (начиная с конца) используется функция **REVERSE**. Функция имеет следующий формат: REVERSE (строковое выражение).

**Например:**

SELECT [Фамилия Имя Отчество]=[ФИО студента],

[rev.spec]=REVERSE ([id специальности]),

[Номер специальности]=[id специальности]+1

FROM [Univer].[dbo].[Студент]

WHERE [id студента] <>101;

Предложение **ORDER BY** применяется для упорядочения выходных данных объединения запросов так же, как и для отдельных запросов.

**Например:**

SELECT [Фамилия Имя Отчество]=[ФИО студента],

[rev.spec]=REVERSE ([id специальности]),

[Номер специальности]=[id специальности]+1

FROM [Univer].[dbo].[Студент]

ORDER BY [id студента] <>101;

Рассмотрим подробнее основные возможности сортировки **ORDER BY** на примере данных таблицы «Товар» (рисунок 13).

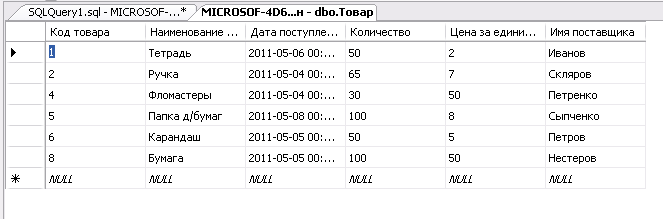


Рисунок 13 – Структура таблицы «Товар»

Включив в запрос SELECT предложение ORDER BY, можно отсортировать результаты запроса. Предложение ORDER BY состоит из ключевого слова ORDER BY, за которым следует список элементов сортировки. Ключевое слово DESC означает сортировку по убыванию. Если указать необязательное и используемое по умолчанию ключевое слово ASC, то сортировка будет произведена по возрастанию. Например, для вывода наименований товаров по убыванию дат поступления с кодом товара равным 5 и ценой за единицу равной 8, следует использовать следующий запрос:

Select [наименование товара], количество, [дата поступления]

From товар

Where [код товара]=5 and [цена за единицу (руб)]=8

Order by количество desc;

Результат запроса представлен на рисунке 14.

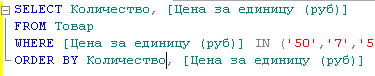


Рисунок 14 – Результат выполнения запроса c ORDER BY

Можно производить сортировку по нескольким столбцам. Например,

требуется вывести значения цены за единицу товара со значениями '50', '7', '5'. Результаты запроса упорядочить по возрастанию цен, а затем по количеству. Соответствующий запрос представлен на рисунке 15.

В списке элементов сортировки для столбцов с именами и без имен (например, вычисляемых столбцов) можно указывать их порядковый номерв списке возвращаемых элементов предложения SELECT. Примером этому может служить запрос, представленный на рисунке 16.



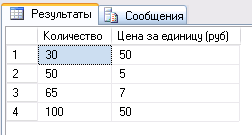
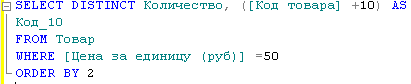


Рисунок 15 – Запрос и результаты его выполнения



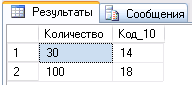


Рисунок 16 – Запрос и результаты его выполнения

В предложении ORDER BY можно указать псевдоним столбца, например Код\_10 в предыдущем примере. Результат выполнения будет совпадать с результатом выполнения предыдущего запроса.

Сортировку по номеру элемента в списке можно использовать при выборке всех столбцов из таблицы. Пусть, например, требуется вывести все данные о товарах, отсортированные по количеству товаров, имеющихся в наличии. Результаты запроса SELECT \* FROM Товар ORDER BY 4 представленs на рисунке 17.

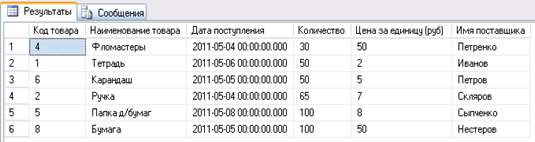
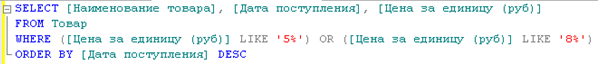


Рисунок 17 – Результаты запроса ORDER BY

**Примечание.** Если задано NULLS FIRST, то для сортировки не может быть использован индекс (использование индексов будет описано далее).

Например, для вывода наименований товара и дат поступления, значения цен которых начинаются с «5» или с «8», отсортированных по убыванию дат выполнения с NULL – значениями вначале можно использовать запрос, представленный на рисунке 18.



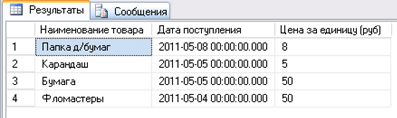


Рисунок 18 – Запрос и результаты его выполнения

**Предложение GROUP BY**

Запрос, включающий в себя предложение GROUP BY, называется запросом с группировкой, поскольку он объединяет строки исходной таблицы в группы и для каждой группы строк генерирует одну строку.

Элементы, указанные в предложении GROUP BY, называются элементами группировки*,* и именно они определяют, по какому признаку строки делятся нагруппы. При этом группой называется набор строк, имеющих одинаковоезначение в элементе (элементах) группировки.

Синтаксис предложения GROUP BY имеет следующий вид:

GROUP BY <элемент\_группировки1> [, <элемент\_группировки2>] …,

Фактически в качестве элемента группировки может выступать любой возвращаемый элемент, указанный в предложении SELECT, кроме значений агрегатных функций. В выражение, представляющее собой <элемент\_группировки>, могут входить скалярные функции, агрегатные функции из различных контекстов или это может быть любая CASE-операция. Использование предложения GROUP BY имеет смысл только при наличии в списке возвращаемых элементов предложения SELECT хотя бы одного вычисляемого столбца или агрегатной функции. Агрегатная функция берет столбец значений и возвращает одно значение. 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. Например, если необходимо вычислить среднее значение цены за единицу товара из таблицы Товар, то можно воспользоваться запросом, приведенном на рисунке 19.



Результаты запроса:

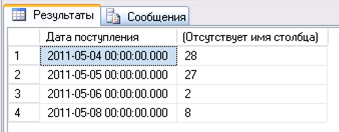
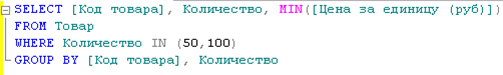


Рисунок 19 – Запрос и результаты его выполнения

Возможна группировка результатов запроса на основании порядкового номера возвращаемого элемента в предложении SELECT. Таким образом, запрос **SELECT [Дата поступления], AVG([Цена за единицу (руб)]) FROM Товар GROUP BY 1** выдаст такой же результат, как и предыдущий запрос.

В SQL можно группировать результаты запроса на основании двух или более элементов. Например, необходимо для каждого товара (по коду товара) необходимо вывести минимальную цену за единицу продукции. Для этого нужно сгруппировать таблицу Товар по коду товара и количеству товара в наличии, применяя к значениям цен агрегатную функцию MIN (рисунок 20).



Результаты запроса:

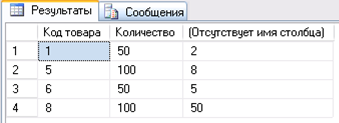


Рисунок 20 – Запрос с функцией MIN и результаты его выполнения

Как следует из предыдущих примеров, на запросы с группировкой накладываются следующие ограничения.

1. В предложении GROUP BY должны быть указаны столбцы или выражения, которые используются в качестве возвращаемых элементов предложения SELECT (за исключением агрегатных функций).

2. Все элементы списка возвращаемых столбцов должны иметь одно значение для каждой группы строк. Это означает, что возвращаемым элементом в предложении SELECT может быть:

* константа;
* агрегатная функция, возвращающая одно значение для всех строк, входящих в группу;
* элемент группировки, который по определению имеет одно и то же значение во всех строках группы;
* функция, которая используется в качестве элемента группировки;
* выражение, включающее в себя перечисленные выше элементы.

**Предложение HAVING**

Предложение HAVING запроса SELECT применяется для наложения условий на строки, возвращаемые при использовании предложения GROUP BY. Оно состоит из ключевого слова HAVING, за которым следует <условие\_поиска>:

<условие\_поиска> ::= [NOT] <условие\_поиска1>

[[AND|OR][NOT] <условие\_поиска2>]…, где <условие\_поиска> позволяет исключить из результата группы, не удовлетворяющие заданным условиям. Условие поиска совпадает с условием поиска, рассмотренным выше для предложения WHERE. Однако в качестве значения часто используется значение, возвращаемое агрегатными функциями. Результат совместной работы HAVING с GROUP BY аналогичен результату работы запроса SELECT с предложением WHERE с той разницей, что HAVING выполняет те же функции над строками (группами) возвращаемого набора данных, а не над строками исходной таблицы. Из этого следует, что предложение HAVING начинает свою работу после того, как предложение GROUP BY разделит базовую таблицу на группы. В противоположность этому использование предложения WHERE приводит к тому, что сначала отбираются строки из базовой таблицы и только после этого отобранные строки начинают использоваться.

Например, чтобы для каждого из товара, количество которого больше 1, вывести сумму таковых в наличии, нужно выполнить запрос, приведенный на рисунке 21.



Результаты запроса:

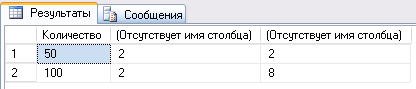


Рисунок 21 – Запрос и результаты его выполнения

Работа этого запроса заключается в следующем. Вначале GROUP BY из таблицы Товар формирует группы, состоящие из одинаковых значений поля Количество. После этого в предложении HAVING происходит подсчет числа строк, входящих в каждую группу, в результирующую таблица включаются все группы, которые содержат более одной строки. Следует отметить, что если задать условие COUNT(\*)>1 в предложении WHERE, то такой запрос потерпит неудачу, так как предложение WHERE производит оценку в терминах одиночной строки, а агрегатные функции оцениваются в терминах групп строк. В то же время из этого не следует, что предложение WHERE не используется с предложением HAVING.

Следует учесть, что предложение HAVING должно ссылаться только на агрегатные функции и элементы, выбранные GROUP BY.

Например, следующий запрос потерпит неудачу:

SELECT Количество, MAX ([Цена за единицу]) FROM Товар

GROUP BY Количество

HAVING [Код товара] = 1;

Поле [Код товара] не может быть использовано в предложении HAVING, потому что оно может иметь (и действительно имеет) больше чем одно значение на группу вывода.

Запрос на рисунке 22 является корректным.



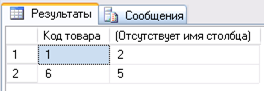
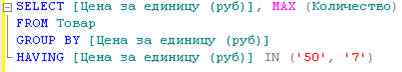


Рисунок 22 – Запрос и результаты его выполнения

Если необходимо узнать максимальные количество товаров в наличии, цена которых начинается на «50» и «7», то можно выполнить запрос, представленный на рисунке 23.



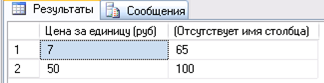


Рисунок 23 – Запрос и результаты его выполнения

# Вложенные подзапросы

В предложении SELECT могут использоваться простые соотнесенные вложенные запросы. При использовании простого подзапроса возвращенный им результат вставляется во все строки, формируемые внешним запросом. В предложении SELECT может использоваться только <скалярный\_подзапрос>, то есть подзапрос, который возвращает только одно значение.

Пусть имеются две таблицы (рисунок 24) и необходимо по каждому абоненту вывести среднее значение его оплаты, а также среднее значение начислений по всем абонентам.

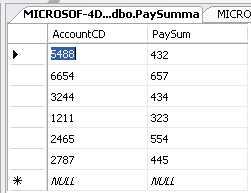
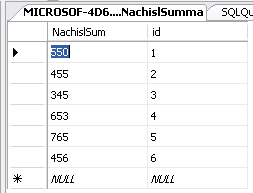


Рисунок 24 – Исходные данные

Запрос может выглядеть следующим образом:

SELECT AccountCD, AVG (PaySum) AS AVG\_Pay,

(SELECT AVG (NachislSum) FROM NachislSumma )

AS AVG\_All\_Nachisl

FROM PaySumma

GROUP BY AccountCD

Результат выполнения запроса представлены на рисунке 25.

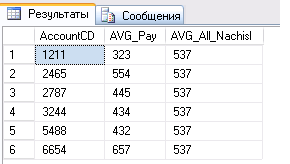


Рисунок 25 – Результаты выполнения запроса

Как следует из этого примера, связь между значением, возвращаемым простым вложенным запросом (среднее значение всех начисленных сумм по всем абонентам), и значениями внешнего запроса фактически отсутствует.

В предложении FROM могут быть определены две и более производные таблицы. Например, требуется вывести среднее количество ремонтных заявок, приходящихся на одного абонента. Для этого нужно определить общее количество ремонтных заявок, общее число абонентов и поделить полученное количество заявок на число абонентов.

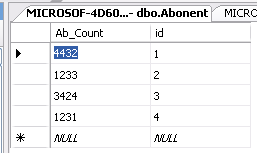
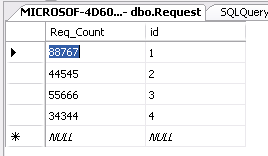


Рисунок 26 – Исходные данные

Запрос к таблицам, представленным на рисунке 26, будет выглядеть следующим образом:

SELECT (CAST (R.Req\_Count AS NUMERIC(5,2)) / A.Ab\_Count)

AS Req\_on\_Ab

FROM (SELECT COUNT (\*) FROM Abonent) AS A (Ab\_Count),

(SELECT COUNT (\*) FROM Request) AS R (Req\_Count)

Результат выполнения запроса представлены на рисунке 27.

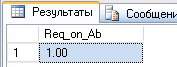


Рисунок 27 – Результаты выполнения запроса

Чтобы получить данные о заказах, все количество которых не будет превышать 80 и исключить из этой группы заказ с номером 76, предыдущий запрос можно модифицировать для таблице, представленной на рисунке 28, следующим образом:

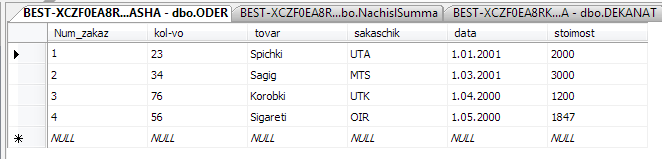


Рисунок 28 – Исходные данные

SELECT \*

FROM ODER

WHERE num\_zakaz IN

(SELECT num\_zakaz

from ODER

where [kol-vo] < 80)

and num\_zakaz NOT IN

(SELECT num\_zakaz from ODER

where [kol-vo] = 76 GROUP BY num\_zakaz);

Результат выполнения запроса представлен на рисунке 29.

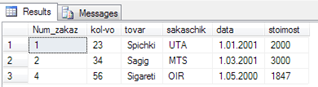


Рисунок 29 – Результат выполнения запроса

Во вложенном запросе можно использовать агрегатные функции. Допустим, необходимо вывести абонентов и значения их начислений за 2001 год (исходные данные представлены в таблице на рисунке 30), превышающие среднюю сумму начислений по всем абонентам за этот год.

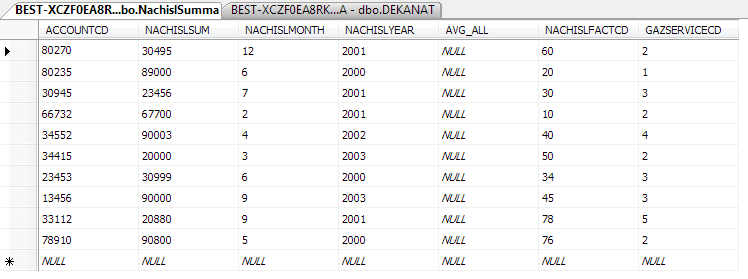


Рисунок 30– Исходные данные

Запрос будет иметь следующий вид:

SELECT AccountCD, NachislSum, NachislMonth, NachislYear,(SELECT AVG(NachislSum)

FROM NachislSumma

GROUP BY NachislYear

HAVING NachislYear=2001) AS Avg\_All

FROM NachislSumma

WHERE NachislSum > (SELECT AVG(NachislSum)

FROM NachislSumma

GROUP BY NachislYear

HAVING NachislYear=2001)

AND NachislYear=2001

ORDER BY 1;

Результат выполнения запроса представлен на рисунке 31.

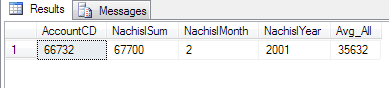
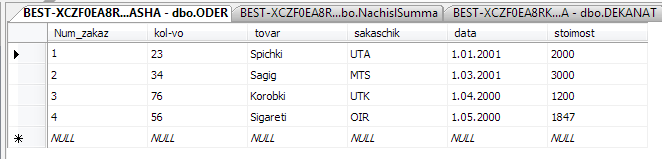


Рисунок 31 – результат выполнения запроса

В этом примере вложенный запрос выполняется один раз, возвращая среднее значение поля NachislSum за 2001 год. Затем это значение последовательно сравнивается с каждой выбираемой строкой из таблицы NachislSumma.

Рассмотрим еще два примера. Для вывода погашенных ремонтных заявок с наиболее поздней датой поступления можно (исходные данные представлены в таблице на рисунке 32), используя следующий запрос:

Рисунок 32 – Исходные данные

SELECT \* FROM ODER

WHERE [kol-vo] = (SELECT MAX ([kol-vo])

FROM ODER

WHERE num\_zakaz < 5);

Результат выполнения запроса представлен на рисунке 33.

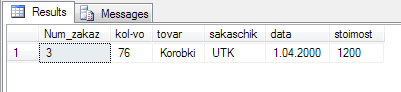


Рисунок 33 – Результат выполнения запроса

Вложенные запросы можно применять в предложении HAVING. Они могут использовать свои собственные агрегатные функции (если эти функции не возвращают многочисленных значений). Также в подзапросе, включенном в условие поиска предложения HAVING внешнего запроса, могут использоваться свои собственные предложения GROUP BY и HAVING. Следует помнить, что аргументы, указанные в HAVING, должны присутствовать в качестве аргументов и в GROUP BY.

Например, для подсчёта числа абонентов с максимальными значениями оплаты за 2000 год (исходные данные представлены в таблице на рисунке 34) можно использовать следующий запрос:

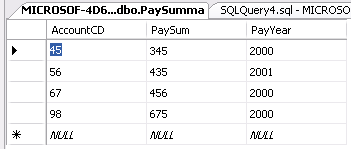


Рисунок 34 – Исходные данные

SELECT COUNT(DISTINCT AccountCD), PaySum

FROM PaySumma

GROUP BY PaySum

HAVING PaySum = (SELECT MAX(PaySum)

FROM PaySumma WHERE PayYear = 2000)

Результат выполнения запроса представлен на рисунке 35:



Рисунок 35 – Результат выполнения запроса

Функции **LEFT** и **RIGHT** используются для выделения нужного количества символов из начала или конца определенной строки соответственно и имеют следующий формат:

LEFT (строковое выражение, длина),

RIGHT (строковое выражение, длина),

где строковое выражение – выражение, из которого будут выделяться

символы; длина – количество выделяемых символов в начале (для LEFT) или конце (для RIGHT) строки.

Функция **OVERLAY** заменяет в исходной строке подстроку, начинающуюся с номера позиция и имеющую размер длина, на значение строки для замены. Функция имеет следующий формат:

**OVERLAY** («исходная строка» PLACING «строка для замены» FROM позиция [ FOR длина ] ), где «исходная строка» и «строка для замены» могут представлять собой строковое выражение.

Если длина не указана, то по умолчанию принимается длина в символах строки для замены (CHAR\_LENGTH («строка для замены»)).

Функция **REPLACE** заменяет все вхождения «подстроки» в «строковое выражение» на указанную «строку для замены». Имеет следующий формат:

REPLACE (строковое выражение, подстрока, строка для замены).

К скалярным функциям относится также функция **TRIM**. Эта функция возвращает строку аргумента, удаляя символы (по умолчанию – пробелы) из начала и/или конца строки.

Для вызова функции используется следующий синтаксис:

TRIM ( [ [LEADING | TRAILING | BOTH ] [удаляемая подстрока ]FROM ] «строковое выражение»).

LEADING указывает на то, что надо удалить указанную подстроку из начала строки, TRAILING – из конца строки, BOTH – из начала и конца строки. При использовании функции следует учитывать следующее:

1) если LEADING, TRAILING или BOTH не указаны, то принимается BOTH по умолчанию;

2) если удаляемая подстрока не определена, то за нее принимается по умолчанию пустая строка;

3) если LEADING, или TRAILING, или BOTH и/или удаляемая подстрока указаны, то после слова FROM должно быть обязательно указано строковое выражение, из которого удаляются символы;

4) указание строки, из которой удаляются символы, после слова FROM не может быть использовано самостоятельно (если LEADING, TRAILING, BOTH и удаляемая подстрока не указаны).

Существуют функции **LPAD** и **RPAD**, которые дополняют строку аргумента слева (LPAD) или справа (RPAD) указанной последовательностью символов (строка заполнитель) до заданного размера (длина). В случае если «строка заполнитель» не указана, для дополнения используется пробел. «Строка заполнитель» обрезается, когда результирующая строка достигает заданной длины. Функции имеют следующий формат:

LPAD(«строковое выражение», длина, [строка заполнитель]),

RPAD(«строковое выражение», длина, [строка заполнитель]).

Для определения символа по известному коду используется функция **ASCII\_CHAR**. Функция имеет следующий формат:

ASCII\_CHAR (код символа)

Код должен лежать в диапазоне от 0 до 255. Следует учесть, что функция возвращает символ для кодировки NONE.

Рассмотренные выше строковые функции возвращают результат в виде строки символов. Существует также ряд строковых функций, которые в качестве результата возвращают числовое значение.

Для определения кода первого символа в указанной строке используется функция **ASCII\_VAL**. Функция имеет следующий формат:

ASCII\_VAL (строка)

Функция вернет ноль, если указана пустая строка.

Для определения позиции первого вхождения заданной подстроки в строку можно использовать функцию **POSITION**. Функция имеет следующий формат:

POSITION (подстрока IN строковое выражение)

Функция возвращает ноль, если подстрока отсутствует внутри строки.

**Функции даты и времени.**

Функции выполняют различные действия над входными значениями времени и даты и возвращают строковое, числовое значение или значение в формате даты и времени.

Для получения значений текущей даты и системного времени сервера используются следующие функции:

* Получение текущей системной даты и времени:

SELECT CURRENT\_TIMESTAMP;

Получение текущего системного времени:

SELECT CONVERT (TIME, CURRENT\_TIMESTAMP)

* Функция **DATEADD** возвращает значение типа дата, время или дата/время, увеличенное или уменьшенное (если количество отрицательное) по сравнению с исходным значением на заданное количество лет, месяцев, дней, недель, часов, минут или секунд. Синтаксис имеет следующий вид: DATEADD (datepart , number , date ).

SELECT DATEADD(MONTH, 2, ’04-05-2006’)

Следует отметить следующее: YEAR, MONTH, DAY и WEEKDAY в качестве временного отрезка не могут использоваться со значениями типа время (например, тип TIME); HOUR, MINUTE и SECOND в качестве временного отрезка не могут использоваться со значениями типа дата (например, тип DATE); все значения временного отрезка могут быть использованы для типов дата/время (тип TIMESTAMP).

* Для определения величины временного промежутка от первого заданного значения типа дата, время или дата/время до второго может использоваться функция **DATEDIFF**. Синтаксис выглядит следующим образом: DATEDIFF («временной отрезок», «значение1», «значение2» ), где «временной отрезок» имеет тот же синтаксис, что и в функции DATEADD.

SELECT DATEDIFF(YEAR, ‘2002-12-31 23:59:59.9999999’,’2011-01-01 00:00:00.0000000’;

Следует отметить следующее: функция возвращает положительное число, если «значение 2» превышает «значение 1», отрицательное, если «значение 1» превышает «значение 2», и ноль – если значения равны. Если результат вычисления дробный, то выводится округленное значение, сравнение значения типа DATE со значением типа TIME недопустимо, как и для функции DATEADD, определенные временные отрезки могут использоваться только с соответствующим им типом.

**Формирование связанных подзапросов**

При использовании подзапросов во внутреннем запросе можно ссылаться на таблицу, имя которой указано в предложении FROM внешнего запроса. В этом случае такой связанныйподзапрос выполняется по одному разу для каждойстроки таблицы основного запроса. Предложение GROUP BY позволяет группировать выводимые SELECT – запросом записи по значению некоторого поля. Использование предложения HAVING позволяет при выводе осуществлять фильтрацию групп данных. Предикат предложения HAVING оценивается не для каждой строки результата, а для каждой группы выходных записей, сформированной предложением GROUP BY внешнего запроса.

Напишем запрос с подзапросом для получения данных обо всех оценках студента с фамилией «Иванов». Предположим, что его персональный номер неизвестен.

select MARK

from [EXAM MARKS]

where [STUDENT ID] =

(select [STUDENT ID]

from STUDENT

where SERNAME = 'Иванов');

Напишем запрос, выбирающий данные об именах всех студентов, имеющих по предмету с идентификатором 10 балл выше общего среднего балла.

select NAME, SERNAME

from STUDENT

where [STUDENT ID] in

(select [STUDENT ID]

from [EXAM MARKS]

where [SUBJ ID] = 10);

Напишем запрос, который выполняет выборку имен всех студентов, имеющих по предмету с идентификатором 10 балл ниже общего среднего балла

select NAME, SERNAME

from STUDENT

where [STUDENT ID] in

(select [STUDENT ID]

from [EXAM MARKS]

where [SUBJ ID] = 10);

Напишем запрос, выполняющий вывод количества предметов, по которым экзаменовался каждый студент, сдававший более 1 предмета.

select COUNT([SUBJ ID])

from [EXAM MARKS] A

group by [STUDENT ID]

having 1 < (select COUNT([SUBJ ID])

from [EXAM MARKS] B

where A.[STUDENT ID] = B.[STUDENT ID]);

Напишем команду SELECT, использующую связанные подзапросы и выполняющую вывод имен и идентификаторов студентов, у которых стипендия совпадает с максимальным значением стипендии для города, в котором живет студент.

select NAME, [STUDENT ID]

from STUDENT A

where STIPEND =

(select MAX (STIPEND)

from STUDENT B

where A.CITY = B.CITY);

Напишем запрос, который позволяет вывести имена и идентификаторы всех студентов, для которых точно известно, что они проживают в городе, где нет ни одного университета.

select NAME, [STUDENT ID]

from STUDENT A

where CITY not in

(select CITY

from UNIVERSITY B

where A.CITY = B.CITY);

Напишем два запроса, которые позволяют вывести имена и идентификаторы всех студентов, для которых точно известно, что он проживают не в том городе, где расположен их университет.

select NAME, [STUDENT ID]

from STUDENT A

where CITY not in

(select CITY

from UNIVERSITY B

where A.CITY = B.CITY);

**Использование оператора EXISTS**

Используемый в SQL оператор EXISTS генерирует значение **истина** или **ложь,** подобно булеву выражению. Используя подзапросы в качестве аргумента, этот оператор оценивает результат выполнения подзапроса как истинный, если этот подзапрос генерирует выходные данные, то есть в случае существования хотя бы одного найденного значения*.* В противном случае результат подзапроса ложный. Оператор EXISTS не может принимать значение UNKNOWN (не известно).

Напишем запрос с EXISTS, позволяющий вывести данные обо всех студентах, обучающихся в вузах, которые имеют рейтинг выше 300

select \*

from STUDENT as A

where exists

(select RATING

from UNIVERSITY as B

where A.[UNIV ID] = B.[UNIV ID] and RATING > 300);

Результаты запроса представлены на рисунке 36.

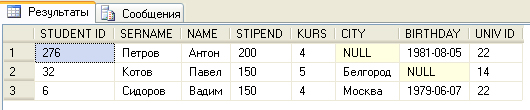


Рисунок 36 – Результаты запроса с использованием EXISTS

Напишем запрос с EXISTS, выбирающий сведения обо всех студентах, для которых в том же городе, где живет студент, существуют университеты, в которых он не учится.

select \*

from STUDENT A

where exists

(select CITY

from UNIVERSITY B

where A.CITY = B.CITY);

Результаты запроса представлены на рисунке 37.



Рисунок 37 – Результаты запроса с использованием EXISTS

Напишем запрос, выбирающий из таблицы SUBJECT данные о названиях предметов обучения, экзамены по которым сданы более чем одним студентом.

select distinct [SUBJ ID]

from [EXAM MARKS] as A

where exists

(select \*

from [EXAM MARKS] as B

where A.[SUBJ ID] = B.[SUBJ ID]

AND A.[STUDENT ID] < > B.[STUDENT ID]);

Напишем запрос, выбирающий данные о названиях университетов, рейтинг которых равен или превосходит рейтинг Воронежского государственного университета.

SELECT \*

FROM UNIVERSITY

WHERE RATING > = ALL

(SELECT RATING

FROM UNIVERSITY

WHERE CITY = 'Воронеж') ;

Результаты запроса представлены на рисунке 38.

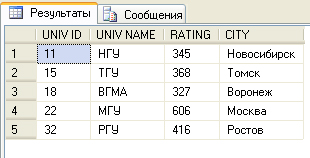


Рисунок 38 – Результаты запроса

Напишем запрос, использующий ALL и выполняющий выборку данных о студентах, у которых в городе их постоянного местожительства нет университета.

select NAME, [STUDENT ID]

from STUDENT A

where CITY < > ALL

(select CITY

from UNIVERSITY B

where A.CITY = B.CITY);

Результаты запроса представлены на рисунке 39.

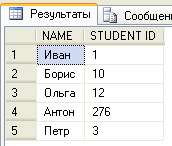


Рисунок 39 – Результаты запроса с использованием ALL

Напишем запрос, выбирающий из таблицы EXAM\_MARKS данные о названиях предметов обучения, для которых значение полученных на экзамене оценок (поле MARK) превышает любое значение оценки для предмета, имеющего идентификатор, равный 10.

SELECT \*

FROM [EXAM MARKS]

WHERE MARK > ANY

(SELECT MARK

FROM [EXAM MARKS]

WHERE [SUBJ ID] = 10);

Результаты запроса представлены на рисунке 40.

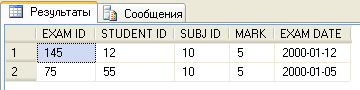


Рисунок 40 – Результаты запроса с использованием ANY

**Связанные подзапросы в предложениях WHERE и HAVING**

При использовании связанного вложенного запроса в условиях поиска предложений WHERE и HAVING он может представлять собой скалярный подзапрос, подзапрос столбца или табличный подзапрос, как и для простых вложенных запросов. Однако так как запрос связанный, внутренний запрос выполняется отдельно для каждой строки внешнего запроса (*текущая строка-кандидат*). Рассмотрим примеры, в которых используются скалярный подзапрос и подзапрос столбца. Подзапросы, представляющие собой табличный подзапрос, будут рассмотрены позднее при изучении предикатов EXISTS и SINGULAR.

Например, чтобы вывести все сведения об абонентах, которые подали заявки на ремонт газового оборудования 17 декабря 2001 года, можно использовать следующий связанный вложенный запрос:

SELECT \* FROM Abonent Out

WHERE '17.12.2001' IN

(SELECT IncomingDate FROM Request Inn

WHERE Out. AccountCD = Inn. AccountCD);.

В этом примере Out и Inn - это псевдонимы таблиц Abonent и Request соответственно (могут задаваться произвольно). Так как значение в поле AccountCD внешнего запроса меняется (при переборе строк), внутренний запрос должен выполняться отдельно для каждой строки внешнего запроса.

В этом примере SQL осуществляет следующую процедуру:

1) выбирает строку c данными об абоненте, имеющем номер лицевого счета «005488» (первая строка) из таблицы Abonent;

2) сохраняет эту строку как текущую строку – кандидат под псевдонимом Out;

3) выполняет вложенный запрос, просматривающий всю таблицу Request, чтобы найти строки, где значение поля Inn.AccountCD – такое же, как значение Out.AccountCD (005488). Затем из каждой такой строки таблицыRequest извлекается поле IncomingDate. В результате вложенный запрос, представляющий собой подзапрос столбца, формирует набор значений поля IncomingDate для текущей строки – кандидата;

4) после получения набора всех значений поля IncomingDate для поля AccountCD= «005488» анализируется условие поиска основного запроса, чтобы проверить, имеется ли значение 17 декабря 2001 в наборе всех значений поля IncomingDate. Если это так (а это так), то выбирается строка с номером лицевого счета «005488» для вывода ее из основного запроса;

5) повторяются пункты 1- 4 (для строки с номером лицевого счета «015527» и т.д.), пока каждая строка таблицы Abonent не будет проверена.

Рассмотрим пример сравнения значения, возвращаемого вложенным запросом, с константой. Вывести информацию об исполнителях ремонтных заявок, назначенных на выполнение четырех и более заявок, можно с помощью следующего запроса:

SELECT \*

FROM Executor E

WHERE 4<= (SELECT COUNT(R.RequestCD) FROM Request R

WHERE E.ExecutorCD = R.ExecutorCD);.

В данном примере связанный подзапрос в условии поиска представляет собой скалярный подзапрос. Он возвращает одно единственное значение (количество ремонтных заявок) для текущей строки – кандидата, выбранной из таблицы Executor. Если это значение больше или равно 4, то текущая строка - кандидат выбирается для вывода из основного запроса. Эта процедура повторяется, пока каждая строка таблицы Executor не будет проверена.

В SQL имеется возможность использовать соотнесенный вложенный запрос, основанный на той же самой таблице, что и основной запрос*.* Это позволяет использовать соотнесенные вложенные запросы для извлечения сложных форм производной информации. Например, вывести размеры начислений, превышающие среднее значение начислений для каждого абонента, можно с помощью следующего запроса:

SELECT F.NachislSum,

(SELECT AVG(D.NachislSum)

FROM NachislSumma D

WHERE F.AccountCD = D.AccountCD) AS AVG\_D,

A.AccountCD, A.Fio

FROM Abonent A, NachislSumma F

WHERE F.NachislSum >

(SELECT AVG(S.NachislSum)

FROM NachislSumma S

WHERE F.AccountCD = S.AccountCD)

AND A.AccountCD = F.AccountCD

ORDER BY 3;

В этом примере производится одновременная оценка среднего значения для всехстрок, удовлетворяющих условию поиска в предложении WHERE вложенного связанного запроса, одной и той же таблицы со значениями строки-кандидата. Выбирается первая строка-кандидат из таблицы NachislSumma и сохраняется под псевдонимом F. Выполняется вложенный запрос, просматривающий ту же самую таблицу NachislSumma с самого начала, чтобы найти все строки, где значение поля S.AccountCD - такое же, как значение F.AccountCD. Затем по всем таким строкам в таблицеNachislSumma вложенный запрос (скалярный подзапрос) подсчитывает среднее значение поля NachislSum. Анализируется условие поиска основного запроса, чтобы проверить, превышает ли значение поля NachislSum из текущей строки - кандидата среднее значение, вычисленное вложенным запросом. Если это так, то текущая строка-кандидат выбирается для вывода. Таким образом, производятся одновременно и вычисление среднего, и отбор строк, удовлетворяющих условию.

**Примечание.** Запрос, использующий агрегатную функцию в условии поиска основного запроса (данная функция является возвращаемым элементом вложенного запроса), нельзя сформулировать с помощью техники соединения таблиц.

Рассмотрим использование соотнесенного вложенного запроса в условии поиска предложения HAVING. Условие поиска предложения HAVING в подзапросе оценивается для каждой группыиз внешнего запроса, а не для каждой строки. Следовательно, вложенный запрос будет выполняться один раз для каждой группы, выведенной из внешнего запроса, а не для каждой строки (как это было при использовании в предложении WHERE).

Например, чтобы подсчитать общие суммы начислений за услуги газоснабжения для абонентов, чьи фамилии начинаются с буквы C, можно использовать следующий соотнесенный вложенный запрос:

SELECT N.AccountCD, SUM(N.NachislSum)

FROM NachislSumma N

GROUP BY N.AccountCD

HAVING N.AccountCD =

(SELECT A.AccountCD FROM Abonent A

WHERE A.AccountCD = N.AccountCD AND A.Fio LIKE 'C%');

Этот запрос выполняется следующим образом. Основной запрос один раз группирует таблицу NachislSumma по полю AccountCD. Затем для каждой группы выполняется связанный вложенный запрос, возвращая единственное значение поля AccountCD таблицы Abonent (т.к. поле AccountCD содержит уникальные значения).

**Предикаты ANY и ALL**

Операции сравнения можно расширить до многократного сравнения с использованием предикатов ANY и ALL. Это расширение используется при сравнении значения определенного столбца со значениями, возвращаемыми вложенным запросом. Предикат ANY, указанный после знака любой из операций сравнения, означает, что будет возвращено TRUE, если хотя бы для одного значения из подзапроса результат сравнения истинен. Предикат ALL требует, чтобы результат сравнения был бы истинен для всех значений, возвращаемых подзапросом.

Рассмотрим использование предиката **ANY**.

Например, требуется вывести информацию об оплатах абонентами за услугу газоснабжения с кодом, равным 2, за период до 2001 года, которые превышают хотя бы одну из сумм, оплаченных за эту же услугу за 2001 год. Запрос будет выглядеть следующим образом:

SELECT \*

FROM PaySumma

WHERE PaySum > ANY (SELECT PaySum

FROM PaySumma

WHERE PayYear=2001 AND

GazServiceCD=2)

AND PayYear<2001 AND GazServiceCD=2;

В этом примере вложенный запрос выполняется один раз, возвращая все значения поля PaySum, для которых истинно условие PayYear=2001 и GazServiceCD=2 (58.7, 250, 20, 20, 80…). Затем значения, выбранные подзапросом, последовательно сравниваются со значением поля PaySum для каждой строки из таблицы PaySumma основного запроса. При первом обнаруженном совпадении сравнение прекращается и соответствующая строка выводится. Условие «> ANY» равносильно утверждению «больше, чем минимальное из существующих», а условие «< ANY» - «меньше, чем максимальное из существующих». Становится очевидным, что такие условия можно записать иначе, используя агрегатные функции MIN и MAX.

Таким образом, предыдущий запрос можно переписать так:

SELECT \*

FROM PaySumma

WHERE PaySum > (SELECT MIN (PaySum)

FROM PaySumma

WHERE PayYear=2001 AND

GazServiceCD=2)

AND PayYear<2001 AND GazServiceCD=2;

Следует отметить, что использование сравнения «= ANY» эквивалентно использованию предиката IN.

Рассмотрим использование предиката **ALL**.

Например, требуется вывести информацию о ремонтных заявках абонентов, даты подачи заявок которых позднее, чем заявки любых абонентов с неисправностью с кодом, равным 2. Запрос будет выглядеть следующим образом:

SELECT \* FROM Request

WHERE IncomingDate > ALL (SELECT IncomingDate

FROM Request

WHERE FailureCD=2);

Если требуется вывести информацию о ремонтных заявках абонентов, даты выполнения заявок у которых позднее, чем даты выполнения заявок любых абонентов по неисправности с кодом, равным 2, то запрос будет выглядеть следующим образом:

SELECT \* FROM Request

WHERE ExecutionDate > ALL (SELECT ExecutionDate

FROM Request

WHERE FailureCD=2);

В процессе выполнения данного запроса подзапросом формируется набор значений поля ExecutionDate, взятых из строк, где FailureCD=2. В результате условие поиска внешнего запроса будет выглядеть следующим образом: ExecutionDate > ALL (24.10.1998, 11.10.2001, 14.09.2001).

Условие «> ALL» равносильно утверждению «больше, чем максимальное», а условие «< ALL» - «меньше, чем минимальное». Становится очевидным, что такие условия можно записать иначе, используя агрегатные функции MAX и MIN. Таким образом, предыдущий запрос можно переписать так:

SELECT \* FROM Request

WHERE IncomingDate > (SELECT MAX (IncomingDate)

FROM Request

WHERE FailureCD=2);

Результат выполнения будет таким же, как и в предыдущем примере. Следует отметить, что использование сравнения «<> ALL» эквивалентно использованию предиката NOT IN, независимо от того, простой или связанный подзапрос используется. Рассмотрим использование связанного подзапроса с предикатом ALL. Пусть требуется вывести названия неисправностей, по которым все ремонтные заявки подавались позднее 1 мая 2001 года. Запрос будет выглядеть следующим образом:

SELECT d.FailureNM

FROM Disrepair d

WHERE '01.05.2001' < ALL (SELECT r.IncomingDate

FROM Request r

WHERE d.FailureCD=r.FailureCD);

**Предикат SINGULAR**

Совместно с подзапросами можно использовать предикат SINGULAR, который проверяет, возвращает ли подзапрос в точности одно значение. Если возвращается NULL или более одного значения, то SINGULAR дает ложь (а NOT SINGULAR - истину). Предикат SINGULAR похож на предикат ALL, за исключением того, что он проверяет наличие одного и только одного соответствующего значения в наборе. С предикатом SINGULAR могут использоваться как простые, так и соотнесенные подзапросы, однако часто использование простого запроса не имеет логического смысла. Следующий запрос отыскивает всех абонентов, которые имеют только одну ремонтную заявку:

SELECT \* FROM Abonent A

WHERE SINGULAR (SELECT RequestCD FROM Request R

WHERE A.AccountCD=R.AccountCD);

Если использовать в предыдущем запросе предикат NOT SINGULAR, то будут выведены абоненты, у которых имеется более одной ремонтной заявки или вообще нет заявок:

SELECT \*

FROM Abonent A

WHERE NOT SINGULAR (SELECT RequestCD FROM Request R

WHERE A.AccountCD=R.AccountCD);.

# 3 Задания для самостоятельной работы

Реализуйте все основные типы выборок, рассмотренных в данной лабораторной работе для таблиц базы данных своего варианта.

## 4 Контрольные вопросы

1. Общая структура команды SELECT.
2. Где и для каких целей применяется выборка?
3. Как просмотреть результаты выборки?
4. Приведите примеры условий формирования условий отбора?
5. Перечислить операторы, которые могут применяться в предложении WHERE?
6. Дайте определения функции агрегирования.
7. Перечислить функции агрегирования, опишите их назначение?
8. Какое предложение применяется для сортировки данных выборки, условия его применения?
9. Каким образом осуществляется выборка из нескольких страниц?
10. Приведите пример создания таблицы на основе выборки.

# Лабораторная работа №4 - Стандартные соединения и объединения таблиц.

**1 Цель работы**

Ознакомиться со структурой и синтаксисом основных конструкция формирования соединений.

**2 Основные теоретические сведения**

Оператор UNION используется для объединения выходных данных двух или более SQL – запросов в единое множество строк и столбцов.

**Например,** для того чтобы получить в одной таблице фамилии и идентификаторы студентов и преподавателей из Краснодара, можно использовать следующий запрос:

SELECT SURNAME, STUDENT\_ID

FROM STUDENT

WHERE CITY = 'Краснодар'

UNION

SELECT SURNAME, TICHER\_ID

FROM TICHER

WHERE CITY **=** 'Краснодар';

Обратите внимание на то, что символом «;» (точка с запятой) оканчивается только последний запрос. Отсутствие этого символа в конце SELECT – запроса означает, что следующий за ним запрос так же, как и он сам, является частью общего запроса с UNION. Использование оператора UNION возможно только при объединении запросов, соответствующие столбцы которых совместимы по объединению*,* то есть соответствующие числовые поля должны иметь полностью совпадающие тип и размер, символьные поля должны иметь точно совпадающее количество символов. Если NULL – значения запрещены для столбца хотя бы одного любого подзапроса объединения, то они должны быть запрещены и для всех соответствующих столбцов в других подзапросах объединения.

Пример использования оператора UNION для базы данных «Университет».

USE [Univer]

Select [наименование университета]

From университет

Union

Select [фио студента]

From студент;

В отличие от обычных запросов UNION автоматически исключает из выходных данных дубликаты строк, например, в запросе совпадающие наименования городов будут исключены.

SELECT CITY

FROM STUDENT

UNION

SELECT CITY

FROM TICHER;

Если все же необходимо в каждом запросе вывести все строки независимо от того, имеются ли такие же строки в других объединяемых запросах, то следует использовать во множественном запросе конструкцию с оператором UNION ALL. Так, в запросе дубликаты значений городов, выводимые второй частью запроса, не будут исключаться.

SELECT CITY

FROM STUDENT

UNION ALL

SELECT CITY

FROM TICHER;

Приведем еще один пример использования оператора UNION. Пусть необходимо составить отчет, содержащий для каждой даты сдачи экзаменов сведения по каждому студенту, получившему максимальную или минимальную оценки.

SELECT 'Максимальная оценка', A.STUDENT\_ID, SURNAME, MARK, EXAM\_DATE

FROM STUDENT A, EXAM\_MARKS В

WHERE (A.STUDENT\_ID = B.STUDENT\_ID

AND B.MARK = (SELECT MAX(MARK) FROM EXAM\_MARKS С

WHERE C.EXAM\_DATE = B.EXAM\_DATE))

UNION ALL

SELECT ‘Минимальная оценка’**,** A.STUDENT\_ID, SURNAME, MARK, EXAM\_DATE FROM STUDENT A, EXAM\_MARKS В

WHERE (A.STUDENT\_ID = B.STUDENT\_ID

AND B.MARK = (SELECT MIN(MARK)

FROM EXAM\_MARKS С

WHERE C.EXAM\_DATE = В.EXAM\_DATE));

Для отличия строк, выводимых первой и второй частями запроса, в них вставлены текстовые константы 'Максимальная оценка' и 'Минимальная оценка'.

Рассмотрим пример использование UNION с ORDER BY. Предложение ORDER BY применяется для упорядочения выходных данных объединения запросов так же, как и для отдельных запросов.

**Например,** упорядочим выходные данных запроса по фамилиям студентов и датам экзаменов:

SELECT'Максимальная оценка', A.STUDENT\_ID, SURNAME, E.MARK, E.EXAM\_DATE

FROM STUDENT A, (SELECT B.STUDENT\_ID, B.MARK, B.EXAM\_DATE FROM EXAM\_MARKS B,

(SELECT MAX (MARK) AS MAX\_MARK, C.EXAM\_DATE

FROM EXAM\_MARKS С GROUP BY C.EXAM\_DATE) D

WHERE B.EXAM\_DATE=D.EXAM\_DATE

AND B.MARK=MAX\_MARK) E

WHERE A.STUDENT\_ID=E.STUDENT\_ID

UNION ALL

SELECT ‘Минимальная оценка’,A.STUDENT\_ID, SURNAME, E.MARK, E.EXAM\_DATE

FROM STUDENT A, (SELECT B.STUDENT\_ID, B.MARK, B.EXAM\_DATE FROM EXAM\_MARKS B,

(SELECT MIN(MARK) AS MIN\_MARK, C.EXAM\_DATE

FROM EXAM\_MARKS С

GROUP BY C.EXAM\_DATE) D

WHERE В.EXAM\_DATE=D.EXAM\_DATE

AND B.MARK=MIN\_MARK) E

WHERE A.STUDENT\_ID=E.STUDENT\_ID

ORDER BYSURNAME,E.EXAM\_DATE;

**JOIN** — оператор языка SQL, позволяющий объединять записи из двух или более таблиц базы данных. Входит в оператор FROM и отдельно от него не используется.

В число этих конструкций входят следующие:

* **INNER JOIN** — внутреннее соединение.
* **OUTER JOIN** — внешнее соединение, как левое (LEFT), так и правое (RIGHT).
* **FULL JOIN** — полное соединение.
* **CROSS JOIN** — перекрестное соединение.

Ключевое слово INNER в запросе может быть опущено, так как эта опция в операторе JOIN действует по умолчанию.

Оператор UNION используется для объединения выходных данных двух или более SQL-запросов в единое множество строк. Использование оператора UNION возможно только при объединении запросов, соответствующие столбцы которых совместимы по объединению*,* то есть соответствующие числовые поля должны иметь полностью совпадающие тип и размер, символьные поля должны иметь точно совпадающее количество символов. Если NULL-значения запрещены для столбца хотя бы одного любого подзапроса объединения, то они должны быть запрещены и для всех соответствующих столбцов в других подзапросах объединения.

Допустим, что имеется таблица Customers, в которой собраны все данные об именах и адресах заказчиков. Требуется выполнить запрос, позволяющий отобрать информацию обо всех заказчиках, заказы которых выполняются в настоящее время.

SELECT DISTINCT с.CustomerID, с.CompanyName

FROM Customers с

INNER JOIN Orders о

###### ON c.CustomerID = o.CustomerlD

Обратите внимание на то, что в запросе используется ключевое слово DISTINCT, поскольку достаточно знать только количество заказчиков, сделавших заказы (причем достаточно только одно­го упоминания каждого заказчика), а не количество заказов. Если бы в этом запросе отсутствовало ключевое слово DISTINCT, то была бы возвращена отдельная строка, относящаяся к каждому заказчику для каждой строки из таблицы Orders, в которой имеется информация об этом заказчике.

Теперь попытаемся получить информацию об общем количестве заказчиков и для этого вызовем на выполнение следующий простой запрос с агрегирующей функцией COUNT:

SELECT COUNT (\*) AS "No. Of Records" FROM Customers

Применение конструкции INNER JOIN приводит к исключению строк в связи с тем, что не обнаруживаются соответству­ющие им строки в другой таблице, а использование конструкции WHERE приводит к исключению строк из возвращаемого набора, поскольку эти строки не соответствуют сданным критериям.

Применение конструкции JOIN такого типа, как OUTER JOIN, скорее можно счи­тать исключением, а не правилом:

Чаще всего при выборке данных с использованием оператора соединения не­обходимо обеспечить, чтобы данные соответствовали всем заданным критери­ям, а этого позволяет добиться только конструкция INNER JOIN. Многие разработчики, использующие язык SQL, осваивают лишь внутреннее соединение, осуществляемое с помощью конструкции INNER JOIN, но так и не заходят глубже; иными словами, многие разработчики просто не умеют пользо­ваться разновидностью оператора соединения с конструкцией OUTER. Цели, которые позволяет достичь применение конструкции OUTER JOIN, часто достижимы с помощью других методов. Разработчики зачастую просто забывают о том, что может использоваться по­добная конструкция.

При использовании конструкции INNER JOIN исключают­ся все строки, не соответствующие всем заданным критериям, а при использовании конструкции OUTER существует возможность включить в результирующий набор строки, которые со­ответствуют хотя бы одному из заданных критериев.

Следует отметить, что ключевое слово OUTER является необязательным, достаточно лишь включить ключевое слово LEFT или RIGHT (например, LEFT JOIN). Таблица, имя кото­рой упоминается перед ключевым словом JOIN, рассматривается как левая таблица, LEFT, а таблица, имя которой следует за ключевым словом JOIN, — как правая табли­ца, RIGHT.

Как уже было сказано, применение операторов с конструкцией OUTER JOIN приво­дит к получению не только тех данных, которые соответствуют всем критериям, но и данных, соответствующих лишь некоторым критериям, а то, какие именно строки, соответствующие лишь некоторым критериям, будут включены в результирующий на­бор, зависит от выбора той или иной стороны соединения. При использовании кон­струкции LEFT OUTER JOIN включается вся информация из таблицы, указанной слева ОТ ключевого слова JOIN, а при использовании конструкции RIGHT OUTER JOIN— вся информация из таблицы, указанной справа от этого ключевого слова. Рассмотрим практические примеры применения запросов с левым и правым внешним соедине­ниями, позволяющие лучше понять сказанное.

Предположим, что необходимо узнать, какие скидки предоставляются покупа­телю, величину каждой скидки и названия магазинов, в которых эти скидки предоставляются. В базе данных находятся таблицы discounts и stores. Эти таблицы имеют общий столбец, stor\_id, поэтому можно попытаться непо­средственно выполнить их соединение. Оператор, составленный с применением обычной конструкции INNER JOIN, должен выглядеть примерно таким образом:

###### SELECT discounttype, discount, s.stor\_name

FROM discounts d

JOIN stores s

ON d.stor\_id = s.stor\_id

Для получения сведений о каждой скидке и о тех магазинах, в которых они при­меняются, достаточно только изменить тип конструкции JOIN в запросе следующим образом:

SELECT discounttype, discount, s.stor\_name

FROM discounts d

LEFT OUTER JOIN stores s

ON d.stor\_id = s.stor\_id

Теперь рассмотрим, что произойдет после перехода к применению соединения типа RIGHT OUTER JOIN:

SELECT discounttype, discount, s.ator\_\_name

FROM discounts d

RIGHT OUTER JOIN stores s

ON d.stor\_id = s.stor\_\_id

Даже несмотря на то, что эта модификация на первый взгляд кажется весьма не­значительной, фактически она приводит к резкому изменению состава результирую­щего набора. Если теперь оператор SELECT \* будет выполнен применительно к таблице stores, то обнаружится, что в состав результатов запроса включены все строки из таблицы stores, причем при наличии соответствующей строки в таблице discounts отобра­жается относящаяся к этой строке информация о скидке. А во всех остальных слу­чаях столбцы, взятые из таблицы discounts, заполняются NULL – значениями. Итак, если допустить, что таблица discounts всегда будет упоминаться в запросе в первую очередь, а таблица stores – во вторую, то, чтобы получить информацию обо всех скидках, нужно использовать конструкцию LEFT JOIN, а для ознакомления с инфор­мацию обо всех магазинах - конструкцию RIGHT JOIN.

Как и многие конструкции в языке SQL, конструкция FULL JOIN по существу выполняет именно то действие, о ко­тором говорит ее название. Эта конструкция согласует данные в таблицах, имена которых находятся по обе стороны от ключевого слова JOIN, и вводит в окончатель­ные результаты все строки, независимо от того, с какой стороны соединения они определены. Конструкции FULL JOIN относятся к числу тех языковых средств, которые вызы­вают восхищение во время их изучения, но в дальнейшем почти не применяются. Основное назначение этой конструкции состоит в том, что она позволяет увидеть полную связь между данными в таком виде, в котором не дается преимущество ни левой, ни правой стороне. Эта конструкция применяется, если есть необходимость ознакомиться с каждой строкой всех таблиц, вводящихся по обе стороны от ключевого слова JOIN, без каких-либо исключений. По-видимому, если одно и то же соединение может быть применено и в форме левого, и в форме правого соединения, то лучше всего использовать полное соединение, имеющее форму конструкции FULL JOIN. Эта конструкция не только дает возможность получить все согласующиеся строки с учетом того поля (полей), на котором основано соединение, но и те строки, которые имеются только в таблицах, находящихся на левой стороне, притом что столбцы, относящиеся к правой стороне заполняются NULL-значениями. Наконец, та же операция возвращает все строки, имеющиеся только в таблицах, заданных с правой стороны, а вместо значений полей таблиц, относящихся к левой стороне, подставляются NULL-значения.

Вначале выполним соединение двух первых таблиц с использованием конструкции FULL JOIN:

###### SELECT a.Address, va.AddressID

FROM VendorAddress va

FULL JOIN Address a

ON va.AddressID = a.AddressID

Операторы с конструкциями CROSS JOIN обладают действительно необычными особенностями. Соединения CROSS JOIN отличаются от соединений других типов тем, что в них отсутствуют операции ON, а также тем, что в них происходит соедине­ние каждой строки таблиц, находящихся с одной стороны от ключевого слова JOIN, с каждой строкой таблиц, находящихся с другой стороны от ключевого слова JOIN. Короче говоря, в конечном итоге формируется декартово произведение всех строк, заданных по обе стороны от ключевого слова JOIN. Операторы с конструкцией CROSS JOIN имеют такой же синтаксис, как и любые другие операторы JOIN, за ис­ключением того, что в них используется ключевое слово CROSS (вместо INNER, OUTER или FULL), а операция ON отсутствует. Ниже приведен краткий пример.

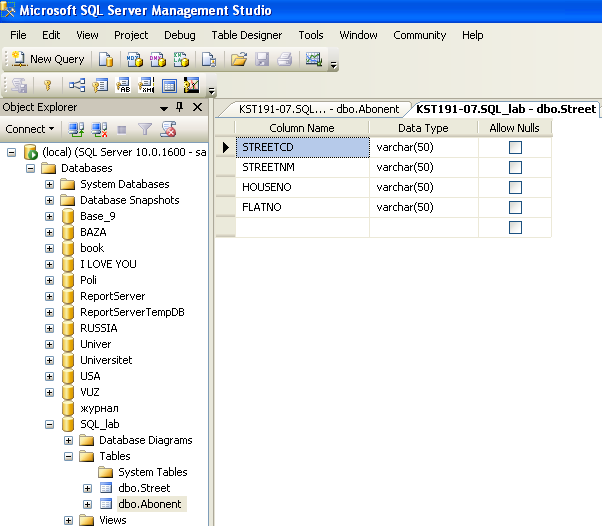
SELECT v.VendorName, a.Address

FROM Vendors v

CROSS JOIN Address a

**Декартово произведение:**

Создадим две таблицы, таблицу Abonent и таблицу Street (рисунок 41). Для получения декартова произведения таблиц в предложении FROM необходимо указать перечень перемножаемых таблиц, а в предложении SELECT – все их столбцы.



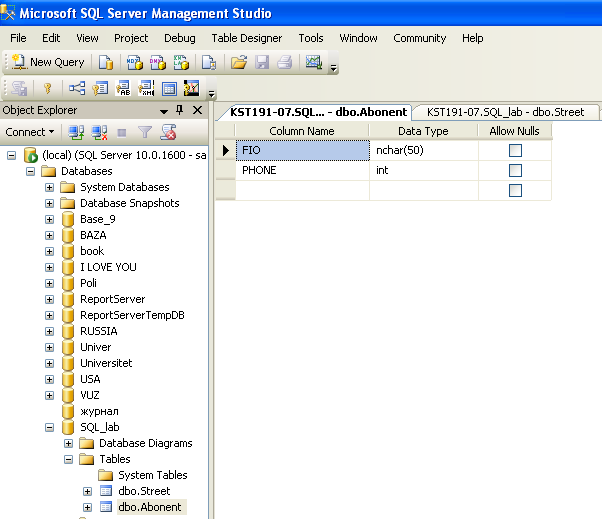


Рисунок 41 – Структура таблиц Abonent и Street

Перемножим таблицы Abonent и Street и получим результирующую таблицу:

SELECT Abonent.\*, Street.\*

FROM Abonent, Street;

Результат работы запроса приведен на рисунке 42.

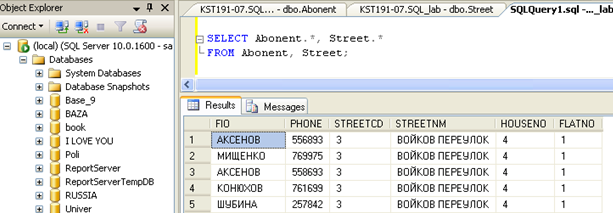


Рисунок 42 – Декартово произведение таблиц Abonent и Street

Такой же результат может быть получен, если использовать следующий запрос на явное соединение:

SELECT Abonent.\*, Street.\*

FROM Abonent CROSS JOIN Street;

Результат работы запроса приведен на рисунке 43.

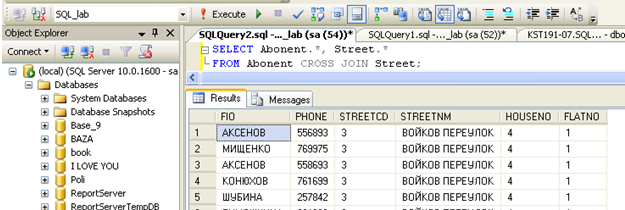


Рисунок 43 - Результат выполнения запроса явного соединения.

Таким образом, декартово произведение двух таблиц – это набор всевозможных комбинаций строк из двух таблиц.

Пример объединения CROSS JOIN для базы данных рассмотренной в теме №1.

Use [univer]

Select [наименование университета], [фио студента]

From университет

Cross join студент;

**Эквисоединение:**

Для получения эквисоединения таблиц необходимо для декартова произведения таблиц установить имеющее смысл соответствие на основе равенства между столбцами соединяемых таблиц.

**Например,** запрос на неявное эквисоединение таблиц Abonent и Street будет выглядеть следующим образом:

SELECT Abonent.\*, Street.\*

FROM Abonent, Street

WHERE Abonent.StreetCD = Street.StreetCD;

Результат работы запроса приведен на рисунке 44.

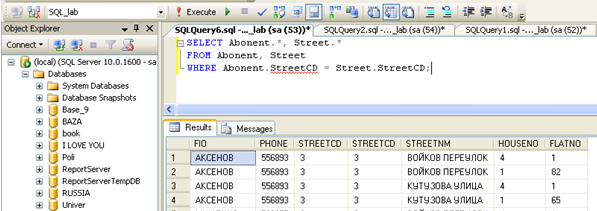


Рисунок 44 **–** Эквисоединение таблиц Abonent и Street

Такой же результат может быть получен, если использовать следующий запрос на явное соединение:

SELECT Abonent.\*, Street.\*

FROM Abonent JOIN Street

ON Abonent.StreetCD = Street.StreetCD;

Результат работы запроса приведен на рисунке 45.

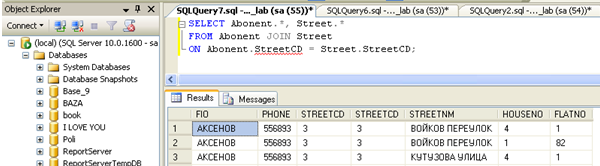


Рисунок 45 – Результат выполнения явного соединения

**Естественное соединение таблиц:**

Для получения естественного соединения таблиц необходимо в эквисоединении таблиц исключить дубликаты повторяющихся столбцов (столбцов, входящих в условие соединения). Для предыдущего примера естественное соединение таблиц Abonent и Street по столбцу StreetCD выглядит следующим образом (рисунок 46):

SELECT AccountCD, Street.StreetCD, StreetNM, HouseNo, FlatNo, Fio, Phone

FROM Abonent, Street

WHERE Abonent.StreetCD = Street.StreetCD;

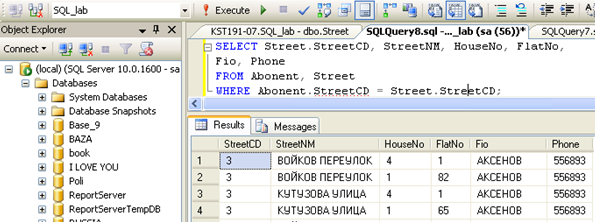


Рисунок46 **-** Естественное соединение таблиц Abonent и Street

Такой же результат может быть получен, если использовать следующий запрос на явное естественное соединение:

SELECT AccountCD, StreetCD, StreetNM, HouseNo, FlatNo, Fio, Phone

FROM Abonent NATURAL JOIN Street;

**Композиция:**

Для создания композиции таблиц нужно исключить из вывода все столбцы, по которым проводилось соединение таблиц, например следующим образом (рисунок 47):

SELECT AccountCD, StreetNM, HouseNo, FlatNo, Fio, Phone

FROM Abonent, Street

WHERE Abonent.StreetCD = Street.StreetCD;

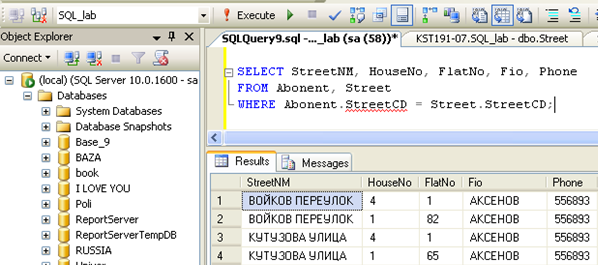


Рисунок 47 – Результат выполнения композиции

**Тета-соединение:**

Тета-соединение предназначено для тех случаев, когда необходимо соединить две таблицы на основе некоторых условий, отличных от равенства. Например, получить тета – соединение таблиц Abonent и Street можно следующим образом (рисунок 48):

SELECT Abonent.\*, Street.\*

FROM Abonent, Street

WHERE Abonent.StreetCD < Street.StreetCD;

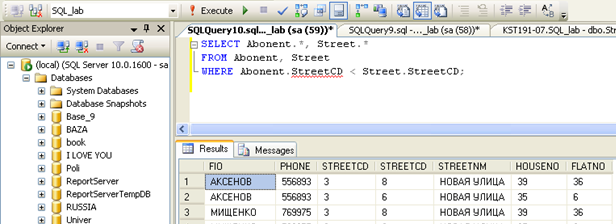


Рисунок 48 – Результат выполнения Тета-соединениея

**Соединение таблицы со своей копией:**

Некоторые многотабличные запросы используют отношения, существующие внутри одной из таблиц. Чтобы обратиться к одной и той же таблице внутри одного запроса, используется псевдоним таблицы, определяемый непосредственно после имени таблицы в предложении FROM запроса SELECT. Например, чтобы найти все пары абонентов, проживающих на одной и той же улице, можно использовать следующее неявное соединение таблицы Abonent со своей копией:

SELECT F.Fio, S.Fio

FROM Abonent F, Abonent S

WHERE F.StreetCD = S.StreetCD AND F.Fio < S.Fio;

В этом примере для таблицы Abonent определены два псевдонима: F (First) и S (Second). Эти псевдонимы будут существовать, пока выполняется запрос. Дополнительное условие поиска "F.Fio < S.Fio" предназначено для удаления из ТРЗ повторяющихся строк, появляющихся в результате того, что запрос выбирает все комбинации строк с одинаковым кодом улицы.

Такой же результат может быть получен, если использовать следующий запрос на явное соединение:

SELECT F.Fio, S.Fio

FROM Abonent F JOIN Abonent S ON F.StreetCD = S.StreetCD

WHERE F.Fio < S.Fio;

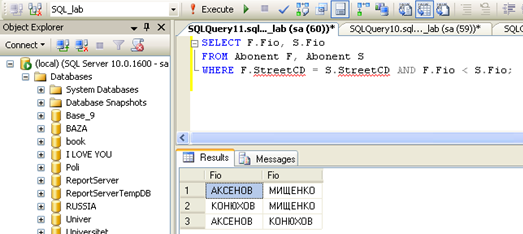


Рисунок 49 - Результат выполнения неявного соединения

таблицы Abonent со своей копией

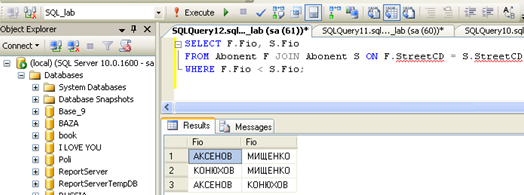


Рисунок 50 - Результат выполнения явного соединения

**Внешнее соединение таблиц**

При использовании внутреннего (INNER) соединения таблиц соединяются только те их строки, в которых совпадают значения полей, задаваемые в запросе предложением WHERE. Однако во многих случаях это может привести к нежелательной потере информации. Рассмотрим пример запроса на выборку списка фамилий студентов с полученными ими оценками и идентификаторами предметов. При использовании внутреннего соединения в результат запроса не попадут студенты, которые еще не сдавали экзамены, и которые, следовательно, отсутствуют в таблице EXAMJMARKS. Если же необходимо иметь записи об этих студентах в выдаваемом запросом списке, то можно присоединить сведения о студентах, не сдававших экзамен, путем использования оператора UNION с соответствующим запросом. Например, следующим образом:

SELECT SURNAME, CAST MARK AS CHARdV CAST SUBJ\_ID AS CHAR (10)

FROM STUDENT,EXAM\_MARKS

WHERE STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID

UNION

SELECT SURNAME, CAST NULL AS CHAR(1), CAST NULL AS CHAR(10)

FROM STUDENT

WHERE NOT EXIST

(SELECT \*

FROM EXAM\_MARKS

WHERE STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID);

Нужный результат может быть получен и путем использования внешнего соединения, точнее, одной из его разновидностей - левого внешнего соединения, с применением которого запрос будет выглядеть следующим образом:

SELECT SURNAME, MARK

FROM STUDENT LEFT OUTER JOIN EXAM\_MARKS

ON STUDENT.STUDENT ID = EXAM MARKS.STUDENT ID;

При использовании левого соединения расширение выводимой таблицы осуществляется за счет записей входной таблицы, имя которой указано слева от оператора JOIN.

Приведенный выше запрос может быть реализован и с применением правого внешнего соединения. Он будет иметь следующий вид:

SELECT SURNAME, MARK

FROM EXAM\_MARKS RIGHT OUTER JOIN STUDENT

ON EXAM\_MARKS.STUDENT\_ID = STUDENT.STUDENT\_ID;

Здесь таблица STUDENT, за счет записей которой осуществляется расширение выводимой таблицы, указана справа от оператора JOIN.

Иногда возникает необходимость включения в результат запроса записей из обеих (правой и левой) соединяемых таблиц, для которых не удовлетворяется условие соединения. Такое соединение называется полным внешним соединением и осуществляется указанием в запросе ключевых слов FULL OUTER JOIN или UNION JOIN.

# 3 Задания для самостоятельной работы

Для проектируемой базы данных создайте соединения следующих типов:

* Соединение с конструкцией JOIN;
* Соединение с конструкцией INNER JOIN;
* Соединение с помощью конструкции OUTER JOIN;
* Соединение с помощью конструкции FULL JOIN;
* Соединение с помощью конструкции CROSS JOIN;
* С помощью операции UNION.

## 4 Контрольные вопросы

1. Дайте определение соединению.
2. Перечислите основные конструкции для формирования соединений.
3. Приведите пример использования конструкции JOIN.
4. Приведите пример использования конструкции INNER JOIN.
5. Чем отличаются результаты выполнения соединений конструкций JOIN и INNER JOIN?
6. Перечислите общие свойства конструкции INNER JOIN и конструкции WHERE.
7. Дайте определение связующей таблице?
8. Для чего и в каком случае используются связующие таблицы?
9. Приведите пример соединения с использованием конструкции OUTER JOIN.
10. Приведите пример соединения с использованием конструкции FULL JOIN.
11. Приведите пример соединения с использованием конструкции CROSS JOIN.
12. Приведите структуру и синтаксис операции UNION.

# Лабораторная работа №5 – Представления.

**1 Цель работы**

Ознакомиться со структурой и понятием представлений, принципом их реализации.

**2 Основные теоретические сведения**

Представление – виртуальный объект в базе данных. Практика показывает, что представления используются, либо слишком часто, либо слишком редко, т.е. очень редко встречаются базы, где бы использование представлений было бы оправданно. Однако представления позволяют достичь следующих целей: сократить кажущуюся сложность базы данных для конечных пользователей, обеспечить доступ пользователей ко всем необходимым данным и вместе с тем запретить доступ к некоторым столбцам, к которым доступ обычных пользователей нежелателен, предусмотреть в базе дополнительные средства индексации, повышающие производительность всей базы. В действительности же представление – это не что иное, как хранимый запрос. Удобство состоит в том, что они обеспечивают выборку данных из одной или нескольких таблиц, обеспечивают согласование данных.

Представление является объектом базы данных, но, в отличие от таблиц, не является физическим хранилищем данных. Его можно представить как хранимое выражение выборки с минимальным набором свойств. Удаление всего представления как объекта никак не повлияет на данные, на основе которых оно построено. В то же время удаление всех записей в представлении может удалить эти записи в исходных таблицах.

Синтаксическая структура оператора – комбинация рассматриваемых ранее операторов CREATE , а так же оператора SELECT.

CREATE VIEW <название\_представления>

AS

SELECT <описание\_оператора…>

Приведем пример простейшего представления, которое будет выводить все имена, фамилия и отчества всех студентов:

CREATE VIEW FIOStud

AS

SELECT StudentFirstName, StudentLastName, StudentPatronymic

FROM Students

GO

Теперь проверим полученный результат, путём выборки всей информации из созданного представления:

SELECT \*

FROM FIOStud

Результат представлен на рисунке 51.

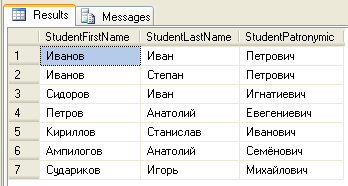


Рисунок 51 – Результат работы представления

Представление можно заменить следующим запросом:

SELECT StudentFirstName, StudentLastName, StudentPatronymic

FROM Students

В запросах участвующих в создании представлений допускается использование любых конструкций, таких как WHEREи др. У пользователя, создающего представление, должны быть разрешения на выборку из соответствующих таблиц. Можно создать представление на основе других представлений либо представлений и таблиц. Но есть ограничения. Нельзя использовать в выборке: предложения ORDER BY, COMPUTE и COMPUTE BY; предложение INTO; временную таблицу. Кроме того, не следует использовать внешние объединения при создании представлений. Представление может ссылаться максимум на 250 столбцов. Команда CREATE VIEW не может использоваться совместно с другими командами SQL Server в одном пакете команд. Нельзя создавать представления внутри хранимых процедур. Оператор UNION невозможно использовать при создании представлений.

Для просмотра выражения, на основе которого создано представление, можно использовать SQL Enterprise Manager или системную процедуру sp\_helptext. В качестве параметра системной хранимой процедуре передается имя представления. В то же время помните, что если представление создано с помощью опции ENCRYPTION, то просмотреть текст выражения выборки с помощью sp\_helptext нельзя.

Чтобы получить информацию о таблицах и представлениях, на основе которых создано текущее представление, используйте системную хранимую процедуру sp\_depends. Эта хранимая процедура выводит информацию об иерархии объектов, принявших участие в создании текущего объекта. Она имеет очень простой синтаксис: ЕХЕС sp\_depends <Имя объекта>.

С помощью представлений пользователи могут иметь доступ и изменять только те данные, которые они видят. Остаток базы данных для них недоступен и невидим. К примеру, некоторым пользователям вы решили не давать доступа к столбцам sum\_ и key\_customer в таблице account. Вы можете создать представление (назовите его real\_account), в котором будут выбираться все столбцы, кроме указанных выше, а затем дать пользователям полное разрешение на операции с этим представлением. И только избранным пользователям вы даете доступ к настоящей таблице. Это будет выглядеть следующим образом:

CREATE VIEW reai\_account

AS SELECT number\_record, account, key\_auto, date\_write, sold

WITH CHECK OPTION

REVOKE ALL

ON account

FROM PUBLIC

GRANT ALL

ON real\_account

TO PUBLIC

GRANT ALL

ON account

TO piter, paul, Andrew

Представления удобны тем, что их можно создавать в любой момент, выдавая разрешение с помощью команды GRANT или отменяя его с помощью REVOKE. Представления могут модифицировать данные в исходных таблицах, при этом помните, что никаких промежуточных таблиц нет – изменения сразу попадают в исходные таблицы.

Изменения, которые производятся с помощью представлений, могут воздействовать только на один объект. Пусть представление создано на основе двух таблиц, order\_ и customer.

CREATE VIEW cust\_Order

AS

SELECT customer.name\_customer, customer.address, customer.tel, order.key\_model

FROM customer, order\_

WHERE customer.key\_customer = order\_.key\_customer

Если в запросе, который используется для создания представления, присутствуют выражения, встроенные функции или функции агрегирования, то изменить данные вы не сможете. Что вполне логично, так как нет столбцов в исходных таблицах, связанных со столбцами в представлении. В следующем примере используется функция агрегирования:

CREATE VIEW total\_sum

AS

SELECT customer.name\_customer, customer.address,

customer.tel. Sum (account.summa)

AS SumOfAc

FROM customer, account

WHERE customer.key\_customer = account.key\_customer

GROUP BY customer.name\_customer, customer.address, customer.tel;

Данный запрос выбирает имена клиентов вместе с их адресами и телефонами, а также суммы по всем платежным документам. В полученном представлении мы можем менять значения во всех столбцах, кроме последнего, который является итогом работы функции агрегирования.

Запись не может быть добавлена в представление, если столбцы, которые не входят в представление, не принимают значение NULL или не имеют значения по умолчанию. Если создать представление с помощью выражения SELECT \*, а затем поменять структуру исходных таблиц, новые столбцы не появятся в представлении. То есть звездочка интерпретируется как названия столбцов только во время создания представления. Единственный выход – удалить представление и создать его заново. Для представлений нет команд изменения структуры. Если вы вынуждены поменять список полей и выражений, отображаемых с помощью представления, то удалите его и создайте заново. Если представление ссылается на таблицы, которые были удалены, то при запуске представления вы получите сообщение об ошибке. Если исходные таблицы или представления были удалены, то настоятельный совет разработчиков Microsoft SQL Server таков: вначале удалите представление, а затем создайте его заново. Чтобы не путать представления с таблицами, рекомендуется давать им имена, которые сразу отличали бы их от таблиц.

Создадим представление, которое будет упрощать выборку такой информации, как например поиск самого старшего студента во всей базе данных:

CREATE VIEW OldestStudent\_vw

AS

SELECT sp.SpecName, stud.StudentFirstName, det.StudentBirthday

FROM Spec AS sp

INNER JOIN Students AS stud

ON stud.IdSpec = sp.IdSpec

INNER JOIN StudentsDetails AS det

ON stud.IdStudent = det.IdStudent

WHERE det.StudentBirthday = (

SELECT min(StudentBirthday)

FROM StudentsDetails

)

GO

Проверим созданное представление:

## SELECT \* FROM OldestStudent\_vw

С функциональной возможности использование представлений почти полностью аналогично таблицам. К ним так же можно применять операторы INSERT, UPDATE и DELETE.Однако при их использовании следует учитывать, что если в представлении используется оператор JOIN, то обязательно использование ключевого слова INSTEAD OF, для указания вместо каких столбцов следует вносить новые значения. Для внесения изменений в существующие представления можно использовать ALTER VIEW - для редактирования; CREATE VIEW – для создания новых представлений; DROP VIEW – для удаления представлений.

# 3 Задания для самостоятельной работы

1. Для проектируемой базы данных создайте представления нескольких типов.
2. Внесите изменения в поля базы данных, использую представления.

## 4 Контрольные вопросы

1. Дайте определение представлению.
2. Какие данные использует представление?
3. Приведите пример простого представления.
4. Как можно использовать представления для корректировки данных?

# Лабораторная работа №6 - Хранимые процедуры.

**1 Цель работы**

Ознакомиться с понятиями хранимых процедур и овладеть навыками по их созданию и использованию.

**2 Основные теоретические сведения**

Хранимая процедура — это специальный вид процедуры, который выполняется сервером баз данных. Хранимые процедуры пишутся на процедурном языке, который зависит от конкретной СУБД. Они могут вызывать друг друга, читать и изменять данные в таблицах, и их можно вызвать из клиентского приложения, работающего с базой данных. Хранимые процедуры обычно используются при выполнении часто встречающихся задач (например, сведение бухгалтерского баланса). Они могут иметь аргументы, возвращать значения, коды ошибок и иногда наборы строк и колонок. Однако последний тип процедур поддерживается не всеми СУБД. Хранимые процедуры *–* это откомпилированные SQL - инструкции, которые хранятся на сервере. При их использовании следует принимать во внимание следующее:

* В связи с тем, что процедурный кэш работает по принципу хранения либо самых ранних использовавшихся данных либо недавно использовавшихся данных, хранимая процедура остается в кэше до тех пор, пока не будет вытеснена оттуда другой часто исполняемой процедурой.
* Проверка синтаксических ошибок и компиляция происходят при первом запуске хранимой процедуры на исполнение.
* Процедуры могут выполняться любыми приложениями, что облегчает контроль над целостностью данных.
* Процедуры запускаются приложением, а не SQL Server.
* Процедуры могут либо выбирать данные, либо модифицировать их, но не то и другое одновременно.
* Хранимые процедуры могут использоваться как механизм безопасности. Пользователю предоставляется право запускать хранимую процедуру, но не право непосредственного доступа к данным таблицы.
* Процедуры могут принимать и возвращать параметры.
* Процедуры создаются как для постоянного, так и для временного использования (в течение текущего сеанса работы с SQL Server).

**Системные хранимые процедуры.** В процессе работы многие системные хранимые процедуры необходимы как быстрое средство манипулирования информацией из системных таблиц. Многие задачи администрирования SQL Server выполняются с помощью этих заранее определенных системных процедур, но помните, что вы тоже можете создать системные хранимые процедуры. По умолчанию системная хранимая процедура начинается с префикса sp\_. Для создания своих собственных системных хранимых процедур используйте тот же префикс. Системные процедуры могут исполняться в любой базе данных.

**Расширенные хранимые процедуры.** Для прямого доступа к системным ресурсам Windows NT в SQL Server встроены специальные средства – *расширенные хранимые процедуры* (extended stored procedure). Они позволяют обращаться к функциям, написанным в виде динамических библиотек Windows – DLL, что существенно повышает скорость их выполнения. Расширенные хранимые процедуры отличаются тем, что их имя начинается с символов хр\_. Эти процедуры используются для поддержки интегрированной модели безопасности и системы оповещения SQL Mail. Кроме того, есть пользовательские расширенные процедуры и процедуры общего назначения. Пользовательские расширенные хранимые процедуры могут быть написаны программистом с помощью Microsoft Open Data Services (MODS). MODS – это специальный 32-разрядный интерфейс программирования для разработки приложений доступа клиентов SQL Server к другим источникам данных.

Хранимые процедуры создаются с помощью команды CREATE PROCEDURE. Процедуры можно создавать только в текущей базе данных. Команда CREATE PROCEDURE не может смешиваться с другими SQL -командами в одном пакете. Локальные и глобальные временные хранимые процедуры похожи на временные таблицы. Для того чтобы процедура стала временной, необходимо добавить знак # перед ее именем (локальная процедура) или два знака # (глобальная процедура). Полное имя, включая знаки # и ##, не может превышать 20 символов. Хранимая процедура представляет собой сценарий, который храниться в базе данных и может принимать и передавать параметры, которые фактически не могут быть использованы в обычных сценариях.

Простейший синтаксис операторов создания хранимых процедур:

CREATE PROCEDURE <название\_процедуры>

AS

<Код\_процедуры, например (SELECT <описание\_оператора…>)>

Проще всего описать синтаксис хранимой процедуры на примере, например возвращающий все поля таблицы базы данных.

USE Universitet

GO

CREATE PROCEDURE spStudents

AS

SELECT \* FROM Students

В этом сценарии заслуживает особого внимания то, что оператору CREATE предшествует оператор GO. Дело в том, что большинство операторов CREATE не допускает включение других операторов, кроме оператора создания хранимой процедуры. После создания хранимой процедуры вызовем её на исполнение, чтобы ознакомиться с полученными результатами:

EXEC spStudents

Результат выполнения хранимой процедуры представлен на рисунке 52.

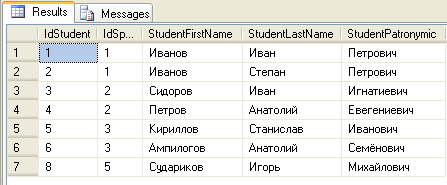


Рисунок 52 – Результат выполнения хранимой процедуры

Хранимые процедуры поддерживают входные и выходные параметры.

Для объявления параметра необходимо знать следующее:

* Имя
* Тип данных
* Заданное по умолчанию значение
* Обозначение входного параметра

Объявление параметров происходит следующим образом:

@имя\_параметра [AS] тип\_параметра [= default|NULL] [OUTPUT|OUT]

Правила формирования имён такие же, как и для других объектов. Имена должны начинаться со знака «@».

Создадим процедуру, которая будет выводить всех студентов, обучающихся на определённом факультете, факультет будет передаваться параметром.

USE Universitet

GO

CREATE PROCEDURE spStudOnFack

@FackN varchar(50) = NULL

AS

SELECT Fack.FackName, Stud.StudentFirstName, Stud.StudentLastName, Stud.StudentPatronymic, StudDet.StudentAddress, StudDet.StudentBirthday

FROM

Fack

#### INNER JOIN Spec As sp

ON sp.IdFack = Fack.IdFack

## INNER JOIN Students AS Stud

ON Sp.IdSpec = Stud.IdSpec

INNER JOIN StudentsDetails AS StudDet

ON Stud.IdStudent = StudDet.IdStudent

WHERE (Fack.FackName = @FackN)

Теперь проверим, как работает наша хранимая процедура – запустим её с помощью команды EXEC:

EXEC spStudOnFack 'КТАСиЗИ'

Результат работы хранимой процедуры 53



Рисунок 53 – Результаты работы хранимой процедуры

В данном случае синтаксис команды CREATE PROCEDURE бедет выглядеть следующим образом:

CREATE PROCEDURE [<Владелец>. ] <Имя процедуры> [; <Номер> ]

[(<Параметр 1> [, <Параметр 2>] . . . [<Параметр 255>] ) ]

[{FOR REPLICATION} | {WITH RECOMPILE}

[([WITH] | [,]} ENCRYPTION]]

AS <Выражение выборки>

В названии процедуры разрешается использовать символ «;» вслед за которым может идти целое число. При этом группе процедур можно присвоить одно имя – различаться они будут по номерам. Это удобно для удаления процедур с помощью команды DROP PROCEDURE. Очень часто процедуры внутри одной группы именуются именно таким образом.

Чтобы запустить процедуру достаточно выполнить следующую команду:

EXECUTE <Имя процедуры> [Параметр 1], … [Параметр N]

Создадим простейшую хранимую процедуру, которая будет осуществлять поиск фамилий преподавателей по вводимой маске:

CERATE PROCEDURE Search\_lastname

@Name CHAR(10)

AS

SELECT T\_lastname

FROM Teacher

WHERE T\_lastname LIKE '%Name%'

Теперь вам достаточно запустить эту процедуру с именем клиента и получить необходимую выборку.

EXECUTE Search\_lastname ‘аврил’

Или:

EXECUTE Search\_lastname

@name = ‘аврил’

Результатом выборки T\_lastname будет к примеру следующая запись:

Гаврилов

(1 row(s) affected)

# 3. Задания для самостоятельной работы

Реализуйте три - четыре хранимых процедур для проектируемой вами базы данных.

## 4 Контрольные вопросы

1. Дайте определение хранимой процедуре.
2. Что понимается под системными хранимыми процедурами?
3. Что понимается под расширенными хранимыми процедурами?
4. Приведите пример создания хранимой процедуры.

# Лабораторная работа №7 - Триггеры.

**1 Цель работы**

Ознакомиться со структурой и синтаксисом организации триггеров.

**2 Основные теоретические сведения**

**Триггер** *–* это специальный тип хранимой процедуры, которая запускается автома­тически системой SQL Server при модифицировании какой-либо таблицы одним из трех операторов: UPDATE, INSERT или DELETE. Триггеры, как другие хранимые процедуры, могут содержать простые или сложные операторы SQL. В отличие от других типов хранимых процедур триггеры запускаются автоматически при указан­ных модификациях данных, их нельзя запустить вручную по имени. Триггер создается по одной таблице базы данных, но он может осуществлять доступ и к другим таблицам и объектам других баз данных. Триггеры нельзя создать по временным таблицам или системным таблицам, а только по определенным пользователем таблицам или представлениям.

Существует пять типов триггеров: UPDATE, INSERT, DELETE, INSTEAD OF и AFTER.

Как следует из названий, триггер UPDATE активизируется, когда выпол­няются изменения в какой-либо таблице, триггер INSERT активизи­руется, когда происходит вставка данных в таблицу и триггер DELETE активизиру­ется, когда из таблицы удаляются данные. Триггер INSTEAD OF выполняется вместо операции вставки, обновления или удаления. Триггер AFTER активизируется пос­ле какой-либо запускающей операции и обеспечивает механизм управления поряд­ком выполнения нескольких триггеров.

Операции обновления, вставки и удаления называются событиями модифика­ции данных. Можное создать триггер, который активизируется при возникнове­нии более чем одного события модификации данных.

Триггеру удовлетворяют следующие правила:

1. Триггеры запускаются только после завершения оператора, который вызвал их активизацию.
2. Если какой-либо оператор пытается выполнить операцию, которая нарушает ка­кое – либо ограничение по таблице или является причиной какой-то другой ошиб­ки, то связанный с ним триггер не будет активизирован.
3. Триггер рассматривается как часть одной транзакции вместе с оператором, кото­рый вызывает его. Поэтому из триггера можно вызвать оператор отката, и этот оператор выполнит откат как триггера, так и соответствующего события моди­фикации данных. Кроме того, при возникновении серьезной ошибки, такой как разъединение с пользователем, SQL Server автоматически выполнит откат всей транзакции.
4. Триггер активизируется только один раз для одного оператора, даже если этот оператор влияет на несколько строк данных.
5. При активизации триггера результаты возвращаются вызываю­щей программе, как и при использовании хранимых процедур.

Начиная с SQL Server 2000, в его состав включены два новых триггера: триггер INSTEAD OF и триггер AFTER. Триггер INSTEAD OF выполняется вместо запуска оператора SQL. Тем са­мым переопределяется действие запускающего оператора. Можно задать по од­ному триггеру INSTEAD OF на один оператор INSERT, UPDATE или DELETE. Триггер INSTEAD OF можно задать для таблицы и/или представления. Можно использовать каскады триггеров INSTEAD OF, определяя представления поверх пред­ставлений, где каждое представление имеет отдельный триггер INSTEAD OF. Триггер AFTER активизируется после успешного выполнения всехопераций, указанных в запускающем операторе или операторах SQL. Сюда включается весь кас­кад действий по ссылкам и все проверки ограничений.

Триггеры создаются с помощью команды CREATE TRIGGER. Команда указывает, для какой таблицы создается триггер и при наступлении какого события триггер должен сработать. Удаление триггера осуществляется с помощью команды DROP TRIGGER.

Команда CREATE TRIGGER имеет следующий синтаксис:

CREATE TRIGGER [<Имя владельца>.]<Имя триггера>

ON [<Имя владельца>.]<Имя таблицы>

FOR { INSERT | UPDATE | DELETE [, ...]}

AS <Команда>

Для того чтобы узнать, какие триггеры связаны с таблицей, выполните хранимую процедуру:

sp\_depends <Имя таблицы>

Необходимо учитывать, что есть команды, которые нельзя использовать в триггерах:

1. Все команды CREATE применительно к объектам DATABASE, TABLE, INDEX, PROCEDURE, DEFAULT, RULE, TRIGGER и VIEW;
2. Все команды DROP;
3. ALTER TABLE и ALTER DATABASE;
4. TRUNCATE TABLE;
5. GRANT и REVOKE;
6. UPDATE STATISTICS;
7. RECONFIGURE;
8. LOAD DATABASE и LOAD TRANSACTION;
9. все команды DISK;
10. SELECT INTO (из-за того, что она создает таблицу).

Когда происходит выполнение триггера для вставки новой записи (по команде INSERT), новые записи добавляются к таблице с триггером и к таблице, которая именуетсяinserted. Последняя представляет собой логическую таблицу, которая содержит копию всех записей, которые добавляются в таблицу. Таблица inserted может быть проверена триггером. Она позволяет сравнивать добавляемые записи с записями в таблице. Записи в таблице inserted всегда являются копиями одной или нескольких записей в результирующей таблице. Преимущества использования inserted заключаются в том, что вам нет необходимости хранить информацию в переменных для проверки каких-либо значений. Вообще говоря, тело триггера (команды, которые выполняются при его вызове) может быть абсолютно произвольным, а структура тел триггеров аналогична.

Приведем пример создания триггера на примере вставки для таблицы Student\_c. Необходим триггер, который будет срабатывать на операцию добавления фамилии и имени коммерческого студента. В его задачу будет входить добавление аналогичных записей в таблицу с общим списком студентов:

CREATE TRIGGER Table\_insert

ON Student\_c

FOR INSERT

AS

INSERT INTO Student (S\_lastname, S\_firstname)

VALUES (Inserted.Lastname, Inserted.Firstname)

Когда срабатывает триггер на удаление (по команде DELETE), удаленные записи размещаются в специальной таблицеdeleted. Необходимо учитывать следующие особенности триггера на удаление:

* Таблица deleted и (результирующая) таблица базы данных не имеют общих записей.
* Когда запись добавляется к deleted, она больше не существует в основной таблице.
* Основная таблица размещается в пространстве, зарезервированном для текущей базы данных.
* Таблицаdeleted всегда хранится в кэше.
* Таблицы deleted и inserted — логические. Структурно они всегда похожи на таблицы, для которых определен триггер. Они содержат информацию, которую изменил пользователь.

Примером для данного триггера может стать аналогичный вышеприведенному триггер, срабатывающий на удаление записей из таблицы Student\_c – удаляющий аналогичные записи в таблице Student. При необходимости можно усложнять триггер для срабатывания при каких либо условиях:

CREATE TRIGGER Table\_delete

ON Student\_c

FOR DELETE

###### AS

DELETE S\_lastname

FROM Student, Deleted

WHERE EXISTS Student.S\_lastname = Deleted.Lastname

Когда выполняется команда UPDATE для таблицы, которая имеет соответствующий триггер, исходные записи перемещаются в таблицу deleted, в то время как измененные записи помещаются в таблицу inserted. Сама таблица тоже обновляется. При этом учитывайте следующее:

* После того как все записи будут модифицированы, загружаются таблицы deleted и inserted, после чего срабатывает триггер на модификацию данных текущей таблицы.
* Триггер может обращаться как к таблицам deleted и inserted, так и к модифицируемой, для того чтобы определить, как много данных было обновлено и какие действия следует предпринять.

# 3 Задания для самостоятельной работы

Создайте примеры триггеров вставки, модификации и удаления.

## 4 Контрольные вопросы

1. Дайте определения триггеру.
2. Приведите примеры возможных триггеров?
3. Описать синтаксис набора команд создания триггеров?
4. Что необходимо учитывать при использовании триггеров?

# Лабораторная работа №8 – Ограничения.

**1 Цель работы**

Ознакомиться со структурой и понятием ограничений, принципом их реализации.

**2 Основные теоретические сведения**

Ограничение — это, прежде всего, формулировка требований к данным. Ограничения уста­навливаются на уровне столбца или таблицы и гарантируют соответствие данных опреде­ленным правилам обеспечения целостности данных.

**Ограничения домена** распространяются на один или несколько столбцов. Под этимиограничениями подразумеваются способы обеспечения того, чтобы какой – то конкретный столбец или ряд столбцов соответствовал определенным критериям. Эти ограничения применяются при вставке или обновлении строки без учета того, что в таблице имеются какие-либо иные другие строки; интерес представляют только данные рассматриваемого столбца.

**Ограничения сущности** полностью относятся к отдельным строкам. В действи­тельности в ограничении этого типа не рассматривается весь столбец как таковой, интерес представляет только какая – то конкретная строка. Самым наглядным ограни­чением этого типа является такое ограничение, согласно которому в каждой строка таблицы должно присутствовать уникальное значение одного столбца или комбина­ции столбцов. На первый взгляд может показаться, что такое определение ограничения сущности, согласно которому, допустим, какой – то столбец должен содержать уникальные значения, полностью совпадает с определением ограничения домена. Но фактически дело обстоит иначе. В формулировке ограничения сущности ничего не сказано о том, что данные столбца должны соответствовать какому – то определенному формату или что значе­ния в столбце должны быть больше или меньше какой-то величины. Единственное, что сказано, относится только к определенной строке и предъявляемое к ней требование состоит лишь в том, чтобы содержащееся в ней значение больше не встреча­лось в какой-либо другой строке в той же таблице. Ограничения такого рода будут рассматриваться в контексте описания ограниче­ний PRIMARY KEY и UNIQUE.

**Ограничения ссылочной целостности** создаются в том случае, если значения в одном столбце должны согласовываться со значениями в другом столбце. Предположим, что разрабатывается приложение, предназначенное для приема заказов на товары, и что это приложение должно обеспечивать получение оплаты с помощью кредитных карточек. А чтобы иметь возможность получения платежей от компании — эмитента кредитных карточек, необходимо заключить с этой компанией торговое соглашение в определенной форме. Кроме того, требуется, чтобы служащие, принимающие заказы, не брали для оплаты кредитные карточки, выпущенные компаниями, от которых не обеспечивается получение платежей. Именно в таком случае вступают в силу ограничения ссылочной целостности. С помощью этого огра­ничения может быть создана таблица, которая в дальнейшем именуется таблицей до­мена или поисковой таблицей. Таблицей домена называется таблица, единственным назначением которой является предоставление ограниченного списка допустимых значений. В данном случае может быть создана таблица, которая выглядит так, как показано в таблице №3.

# Таблица №3 – Пример таблицы домена

|  |  |
| --- | --- |
| **CreditCardID** | **Поле CreditCard** |
| 1 | VISA |
| 2 | MasterCard |
| 3 | Discover Card |
| 4 | American Express |

После этого появляется возможность создать одну или несколько таблиц, которые ссылаются на столбец CreditCardID таблицы домена. Чтобы иметь возможность использовать средства обеспечения ссылочной целостности, необходимо предусматриватьв любой таблице, которая определяется какссылающаяся на таблицу CreditCard, такой столбец, который согласуется столбцом CreditCardID таблицы CreditCard. С этого момента в каждой строке, вставляемой в таблицу, ссылающуюся на ограничение, должно присутствовать только такое значение, которое имеется в заданном списке домена. Дополнительная информация по этой теме будет приведена при описании ограничений FOREIGN KEY ниже в данной главе.

**Ограничения Primary key**

Первичные ключи представляют собой уникальные идентификаторы для каждой строки. Столбец первичного ключа должен содержать уникальные значения. Первичные ключи очень важны для нормальной эксплуатации реляционной базы данных, поэтому яв­ляются наиболее фундаментальными объектами базы данных по сравнению со всеми прочими ключами и ограничениями. Таблица может иметь не больше одного первичного ключа. Кроме того, как уже было сказано выше, в базе данных редко встречаются такие таблицы, для которых не требуется первичный ключ. Таблицы без первичного клю­ча встречаются не просто редко, а очень редко. Наличие в базе данных таблицы, не имею­щей первичного ключа, полностью противоречит представлениям о реляционном характере данных, хранимых в базе данных, поскольку из этого следует, что возможность установить связь с какой-либо конкретной строкой этой таблицы отсутствует. Данные, представлен­ные в таблице, не имеющей первичного ключа, не имеют также каких-либо отличительных признаков. Безусловно, нередко встречаются ситуации, в которых многочисленные строки таблицы ло­гически идентичны друг другу, но это не означает, что возможность создания первичного ключа для такой таблицы отсутствует. В подобных обстоятельствах может быть пред­усмотрено искусственное создание ключа того или иного типа. Подобный подход чаще всего реализуется с использованием столбца идентификации, но в некоторых ситуациях в боль­шей степени оправдано применение идентификаторов GUID.

Первичный ключ гарантирует уникальность сочетания значений столбцов, объяв­ленных как принадлежащие к этому первичному ключу. Сами эти уникальные значе­ния служат в качестве идентификаторов для каждой строки в таблице. Для создания первичного ключа по существу применяются два способа. Первичный ключ может быть либо создан с помощью команды CREATE TABLE во время создания таблицы, либо введен в действие впоследствии с помощью команды ALTER TABLE.

Рассмотрим один из операторов CREATE TABLE. В качестве первичного ключа обозначим столбец CustomerNo. Для за­дания ограничения PRIMARY KEY, достаточно ввести информацию об ограни­чении сразу после определения столбца (столбцов), который должен войти в состав первичного ключа. В данном случае достаточно задать ключевое слово PRIMARY KEY:

CREATE TABLE Customers

(

CustomerNo int IDENTITY NOT NULL

PRIMARY KEY,

CustomerName varchar (30) NOT NOLL,

Addressl varchar(30) NOT NULL,

Address2 varchar(30) NOT NULL,

City varchar(20) NOT NULL,

State char (2) NOT NULL,

Zip varchar(10) NOT NULL,

Contact varchar(25) NOT NULL,

Phone char (15) NOT NULL,

FedlDNo varchar(9) NOT NULL,

DatelnSystem smalldatetime NOT NULL

)

Кроме объявления таблицы с первичным ключом, предусмотрена возможность создания первичного ключа на существующей таблице.

USE Accounting

ALTER TABLE Employees

ADD CONSTRAINT PK\_EmployeeID

PRIMARY KEY (EmployeelD)

**Ограничения FOREING KEY**

Внешние ключи не только обеспечивают целостность данных, но и создают связи между таблицами. После задания внешнего ключа на таблице устанавливается зави­симость между таблицей, для которой определяется внешний ключ, и таблицей, на которую ссылается внешний ключ. После задания на ссылающейся таблице внешнего ключа любая строка, вставляемая в эту таблицу, должна соответствовать одному из следующих двух условий: иметь согласующуюся с ней строку в столбце (столбцах), которому соответствует внешний ключ таблицы, указанной в ссылке, или иметь значение столбца (столбцов) внешнего ключа, равное NULL. Такое определе­ние может показаться немного запутанным, поэтому рассмотрим, как оно применя­ется на примере.

Создадим в базе данных Accounting еще одну таблицу и назовем ее Orders. В приведенном ниже сценарии с оператором CREATE заслуживает внимания то, что в таблицах предусмотрено применение и первичного, и внешнего ключа. Из даль­нейшего описания рассматриваемого проекта станет ясно, что существенно важной частью объявления одной из таблиц является спецификация первичного ключа. С другой стороны, объявление внешнего ключа, который задается на таблице в этом сценарии, почти полностью совпадает с объявлением первичного ключа, не считая того, что в этом объявлении указана таблица, на которую ссылается внешний ключ. Синтаксическая структура объявления внешнего ключа предусматривает необходи­мость указания столбца или столбцов, на которые распространяется ограничение FOREIGN KEY, и выглядит примерно таким образом:

<column name> <data type> <nullability>

FOREIGN KEY REFERENCES <table name>(<column name>)

[ON DELETE {CASCADE|NO ACTION|SET NULL|SET DEFAULT}]

[ON UPDATE {CASCADEjNO ACTION|SET NULL I SET DEFAULT}]

Таким образом, применительно к таблице Orders может быть подготовлен сценарий, который выглядит, как показано ниже.

USE Accounting CREATE TABLE Orders

(

OrderlD int IDENTITY NOT NULL

PRIMARY KEY,

CustomerNo int NOT NULL

FOREIGN KEY REFERENCES Customers(CustomerNo),

OrderDate smalldatetime NOT NULL,

EmployeelD int NOT NULL

)

Следует учитывать, что столбец, на который фактически ссылается внешний ключ, должен иметь определенное на нем ограничение PRIMARY KEY или UNIQUE. После успешного применения приведенного выше кода вызовите на выполнение процедуру sp\_help. После этого должно быть обнаружено, что в разделе constra­ints тех результатов, которые получены с помощью процедуры sp\_help, будет со­держаться информация о новом ограничении. Кроме того, для получения еще более подробных сведений об имеющихся ограничениях можно вызвать на выполнение процедуру sp\_helpconstraint, которая также имеет простой синтаксис:

EXEC sp\_helpconstraint <table name>

После применения процедуры sp\_helpconstraint к вновь созданной таблице Orders будет получена информация, позволяющая узнать имена, условия и состоя­ние всех ограничений, заданных на этой таблице. Очевидно, что на данном этапе таблица Orders имеет одно ограничение FOREIGN KEY и одно ограничение PRIMARY KEY.

Итак, определение внешнего ключа было задано непосредственно в объявлении таблицы. Таким образом, после создания этой таблицы внешний ключ становится ее неотъемлемой частью. Поддержание собственной целост­ности должна обеспечивать сама база данных, каждый внешний ключ принудительно вводит одно из ограничений, распространяющихся на хранимые данные, и поэтому гарантирует обеспечение целостности базы данных.

На практике иногда возникают ситуации, когда требуется дополнительно задать на таблице не только первичный ключ, но и какие-либо другие ограничения, в том числе внешние ключи. Для этого применяется процедура, аналогичная созданию пер­вичного ключа.

Зададим еще один внешний ключ на таблице Orders, чтобы обеспечить ввод в поле EmployeelD (предназначенное для хранения идентификаторов служащих, ко­торые вводят заказы) только действительных данных о служащих, которые опреде­лены в таблице Employees. Для этого необходимо предусмотреть уникальную иден­тификацию целевой строки в таблице, указанной в ссылке. Этой цели можно достичь, ссылаясь либо на первичный ключ, либо на столбец с ограничением UNIQUE. В данном случае воспользуемся существующим первичным ключом, который был задан на таблице Employees в одном из примеров, приведен­ных выше, следующим образом:

ALTER TABLE Orders

ADD CONSTRAINT FK\_EmployeeCreatesOrder

FOREIGN KEY (EmployeelD) REFERENCES Employees(EmployeelD)

После этого еще раз вызовите на выполнение процедуру sphelpconstraint применительно к таблице Orders и убедитесь в том, что новое ограничение было успешно задано. Последнее введенное в этом примере ограничение действует точно так же, как и предыдущее, — регламентирует правила, применяемые к данным, вводимым в соответствующий столбец, согласно объявлению самой таблицы. Определение столбца, позволяет, например, вводить строковые данные в столбец числового типа, а ограничение не дает возможности вставлять в таблицу Orders такую строку, в которой значение поля Employee с данными о служащем, ответственным за ввод заказа, не ссы­лается на действительное значение EmployeelD. Если кто-то попытается ввести стро­ку, которая не согласуется со строкой данных о служащем, такая операция вставки з таблицу Orders будет отвергнута в целях обеспечения целостности базы данных.

### Coздание таблицы, ссылающейся на саму себя

Иногда возникает необходимость задать в ограничении столбец, находящийся не в другой таблице, а непосредственно в той же таблице, в которой создается ссылка на исходное ограничение. Это означает, что одна и та же таблица применительно к некоторому ограничению может играть роль и ссылающейся таблицы, и таблицы, указанной в ссылке. Разумеется, подобные ситуации встречаются не очень часто, но достаточно регулярно. Итак, в данном случае рассматривается таблица, в которой имеется ссылка на столбец, представляющий собой столбец идентификации, поэтому необходимо вначале ввести в таблицу хотя бы одну первичную строку и только после этого задавать ограничение:

INSERT INTO Employees

( FirstName, LastName, Title, SSN, Salary, PriorSalary, HireDate, ManagerEmpID, Department

)

VALUES

('Billy Bob', 'Boson', 'Head Cook & Bottle Washer','123-45-6789', 100000,

80000, '1990-01-01', 1, 'Cooking and Bottling'

)

Теперь, после ввода первичной строки, можно приступить к заданию внешнего ключа. В этом варианте создания таблицы, ссылающейся на саму себя, в котором используется оператор ALTER, осуществляемые действия аналогичны тем, которые вы­полняются при уточнении любого другого определения внешнего ключа. Проверим действие следующего оператора на практике:

ALTER TABLE Employees

ADD CONSTRAINT FK\_EmployeeHasManager

FOREIGN KEY (ManagerEmpID) REFERENCES Employees(EmpioyeelD)

В данном операторе есть только одно отличие от оператора CREATE. Но есть и еще один нюанс, состоящий в том, что в данном определении допускается не исполь­зовать ключевое слово FOREIGN KEY и оставлять только конструкцию REFERENCES. К этому времени таблица Employees уже определена, но если бы речь шла о ее создании с самого начала, то на данном этапе можно было бы применить следующий сценарий:

CREATE TABLE Employees

(

EmpioyeelD

PRIMARY KEY, FirstName Middlelnitial LastName Title SSN

Salary PriorSalary

irtt

varchar (25) char (1) varchar (25) varchar (25) varchar (11) money money

IDENTITY

LastRaise AS Salary - PriorSalary, HireDate smalldatetime TerminationDate smalldatetime ManagerEmpID int

REFERENCES Employees(EmpioyeelD), Department varchar (25) NOT NULL

)

Следует отметить, что при попытке уничтожить таблицу Employees в данный момент было бы получено сообще­ние об ошибке. Это связано с тем, что после определения в таблице Orders ссылки на таблицу Employees эти две таблицы становятся, как принято выражаться, "связанными со схемой". Это означает, что в базе данных теперь содержится информация о наличии так называемой зависимости другой таблицы от таблицы Employees. С учетом такой инфор­мации СУБД SQL Server не позволяет уничтожать таблицу, на которую ссылается другая таблица. Чтобы получить возможность выполнить с помощью СУБД SQL Server удаление таблицы Employees (или, по той же причине, таблицы Customers), необходимо вначале уничтожить внешний ключ в таблице Orders. Кроме того, необходимо учитывать, что таблица, созданная с объявлением внешнего клю­ча, ссылающимся на ту же таблицу в ограничении, не позволит ввести даже первоначаль­ную строку, поэтому подобные объявления следует применять лишь с учетом того условия, чтобы ограничение внешнего ключа было задано на столбце, допускающем ввод NULL-значений. Таким образом, появится возможность ввести первую строку, имеющую NULL – значение, в столбце внешнего ключа и тем самым избежать необходимости задания исходной строки.

**Ограничения UNIQUE**

По су­ществу эти ограничения почти полностью соответствуют ограничениям первичного ключа, поскольку требуют наличия уникальных значений во всем указанном в них столбце таблицы. Ограничения UNIQUE часто называют ограничениями альтернативных ключей.Основные различия между ограничениями UNIQUE и ограничениями первичного ключа состоят в том, что в качестве ограниче­ний UNIQUE обычно не применяются уникальные идентификаторы строк таблицы, а, кроме того, на таблице может быть задано несколько ограничений UNIQUE. После того как будет задано ограничение UNIQUE, должно соблюдаться такое усло­вие, чтобы все значения в столбцах, указанных в этом ограничении, были уникальны­ми. При обнаружении попытки обновить или вставить строку со значением, которое уже имеется в столбце с ограничением уникальности, СУБД SQL Server вырабатывает сообщение об ошибке и отклоняет попытку ввести такую строку. В отличие от ограничения первичного ключа, применение ограничения UNIQUE не приводит к тому, что автоматически исключается возможность вставить в соответствующий столбец NULL – значение. В данном случае возможность применения NULL – значений зависит от того, как задана опция null для указанного столбца таблицы. Но следует учитывать, что даже если вставка NULL – значений разрешена, фактически допускается вставка в столбец толь­ко одного такого значения.

Создадим таблицу в базе данных Accounting и назовем ее Shippers:

CREATE TABLE Shippers (

ShipperlD int IDENTITY PRIMARY KEY,

ShipperName varchar(30)

Address varchar(30)

City varchar(25)

State char(2)

Zip varchar(lO)

PhoneNo varchar(14)

UNIQUE

)

После этого необходимо вызвать на выполнение процедуру sp\_helpconstraint применительно к таблице Shippers и проверить, действительно ли таблица Shippers была создана с учетом требуемых ограничений.

Как уже было сказано, ограничение UNIQUE действует в основном по таким же принципам, как ограничения первичных и внешних ключей. Рассмотрим следующий пример создания ограничения UNIQUE на таблице Employees:

ALTER TABLE Employees

ADD CONSTRAINT AK\_EmployeeSSN UNIQUE (SSN)

Сразу же после применения процедуры sphelpconstraint обнаруживается, что ограничение создано в соответствии с запланированным действием и правильно ука­заны столбцы, относящиеся к ограничению. Отметим, что в приведенном выше примере аббревиатура АК, используемая в имени ограни­чения, расшифровывается как Alternate Key (альтернативный ключ). С другой стороны, для обозначения первичных ключей и внешних ключей применяются аббревиатуры РК (Primary Key) и FK (Foreign Key). А в именах ограничений UNIQUE часто используется префикс UQ, или просто U.

**Ограничения CHECK**

Удобным свойством ограничений CHECK является то, что эти ограничения не обя­зательно должны применяться только к какому-то конкретному столбцу. Безусловно, указанные ограничения могут относиться лишь к некоторому столбцу, но также до­пускается их распространение по существу на всю таблицу, в том смысле, что с их помощью может осуществляться проверка значений в одном столбце на основании значений другого столбца. С помощью ограничений CHECK может также осуществляться проверка того, соответствует ли некоторое сочетание значений столбцов заданному критерию. Ограничения CHECK определяются на основании таких же правил, которые рас­пространяются на операции проверки, используемые в конструкции WHERE. Примеры критериев, которые могут применяться в ограничении CHECK, приведены в таблице №4.

Таблица №4 – Примеры применения ограничения CHECK

|  |  |
| --- | --- |
| **Назначение** | **Код SQL** |
| Обеспечение применения в столбце Month только допустимых значений | between I and 12 |
| Правильное форматирование номера карточки социального обеспечения | Like ‘ [0-9] [0-9] [0-9]-[0-9] [0-9]-[0-9] [0-9] ‘ |
| Регламентация списка допустимых значений в поле Shippers | in (‘UPS’, ‘Fed Еx’, ‘USPS’) |
| Регламентация применения только положительных значений цены | UnitPrice >= О |
| Ссылка на другое поле в той же строке | ShipDate >= OrderDate |

В таблице приведена лишь незначительная часть всех возмож­ных примеров, поскольку количество вариантов применения различных операций сравнения в конструкции CHECK является практически бесконечным. В ограничении CHECK может быть задано почти любое такое же выражение, которое допускается за­давать в конструкции WHERE. К тому же ограничения CHECK позволяют достичь гораз­до более высокой производительности по сравнению с альтернативными средствами проверки допустимости данных.

Внесем в таблицу Customers изменение, позволяющее проверять допустимость даты в поле DatelnSystem (это поле системной даты, в котором не может находиться значение будущей даты):

ALTER TABLE Customers

ADD CONSTRAINT CN\_CustomerDateInSystem

CHECK

(DatelnSystem <= GETDATE ())

Теперь попытаемся выполнить вставку строки со значением, нарушающим ограни­чение CHECK; эта попытка должна привести к возникновению ошибки:

INSERT INTO Customers

(CustomerName, Addressl, Address2, City, State, Zip, Contact, Phone, FedlDNo, DatelnSystem) VALUES

CCustomerl', 'Addressl', 'Add2', 'MyCity', 'NY', '55555', 'No Contact', '553-1212', '930984954', '12-31-2049') Msg 547, Level 16, State 0, Line 1

The INSERT statement conflicted with the CHECK constraint

"CN\_CustomerDateInSystem". The conflict occurred in database "Accounting", table "dbo.Customers", column 'DatelnSystem'. The statement has been terminated.

Если после этого будет внесено такое исправление, чтобы данные, вводимые в столбец DatelnSystem, соответствовали критерию, заданному в ограничении CHECK (относились к дате, совпадающей с нынешней или предшествующей ей), то попытка выполнить оператор INSERT завершится успешно.

**Ограничения DEFAULT**

Ограничения DEFAULT принадлежат к одному из двух различных типов инстру­ментальных средств обеспечения целостности данных, которые можно рассматри­вать как относящиеся к заданным по умолчанию значениям. К сожалению, в связи с наличием двух таких разных средств почти с одинаковыми названиями возникает значительная путаница. Ограничения DEFAULT, как и все прочие типы ограничений, предусмотрены в синтаксической структуре определения таблицы. Ограничения DEFAULT указывают, какие действия должны быть выполнены, если происходит вставка новой строки, не содержащей данных, соответствующих тому столбцу, к которому относится это огра­ничение. Вообще говоря, действием обычно является подстановка литерального зна­чения или подстановка одного из нескольких значений, формируе­мых системой, таких как дата, формируемая с помощью функции GETDATE(). Основные особенности ограничений DEFAULT:

* Значения, подстановка которых должна быть выполнена по умолчанию с по­мощью конструкции DEFAULT, используются только в операторах INSERT, а в случае их определения в операторах UPDATE и DELETE игнорируются.
* Если в операторе INSERT для столбца с конструкцией DEFAULT задано какое-либо значение, то предусмотренное по умолчанию значение не используется.
* Если же для такого столбца значение не задано, то всегда используется значе­ние, предусмотренное по умолчанию.

Следует подчеркнуть, что предусмотренные по умолчанию значения предназначе­ны для использования только в операторах INSERT. Но практика показывает, что для многих начинающих разработчиков программ для СУБД SQL Server причина такого положения дел остается непонятной. Тем не менее такая организация работы вполне обоснована — в то время, как происходит вставка строки в таблицу, СУБД SQL Server может использовать только те значения столбцов, которые приведены в операторе INSERT, или значения, заданные по умолчанию. Если же для какого – либо столбца в операторе вставки не указано ни то ни другое, то СУБД SQL Server вставляет в соответствующий столбец таблицы NULL – значение, а если в определении столб­ца имеется конструкция NOT NULL, то СУБД SQL Server отвергает попытку вставки этой строки. Тем не менее, после выполнения оператора вставки строки в таблице уже имеется значение в столбце, для которого задана конструкция DEFAULT, а в опе­раторе UPDATE содержатся новые значения. Если в операторе обновления не пред­усмотрено новое значение для столбца с конструкцией DEFAULT, то СУБД SQL Server оставляет неизменным то значение, которое уже находится в столбце. С другой стороны, если для столбца с конструкцией DEFAULT в операторе обнов­ления предусмотрено значение, то также нет смысла использовать значение, задан­ное по умолчанию, поскольку применяется предоставленное значение.

Синтаксическая структура оператора создания таблицы с ограничением DEFAULT во многом напоминает структуру этого оператора со всеми другими ограничениями столбцов. Для ввода в действие огра­ничения DEFAULT достаточно ввести его в конце определения столбца. Рассмотрим пример применения ограничения DEFAULT.

CREATE TABLE Shippers

(

ShipperlD int IDENTITY

PRIMARY KEY, ShipperName varchar(30)

DatelnSystem smalldatetime DEFAULT GETDATE ()

)

После выполнения сценария с этим оператором CREATE можно снова воспользо­ваться процедурой sphelpconstraint, чтобы ознакомиться с полученными резуль­татами. После этого для проверки того, как происходит подстановка значения, задан­ного по умолчанию, выполним вставку новой записи:

INSERT INTO Shippers

(ShipperName) VALUES

('United Parcel Service')

Затем выполним оператор SELECT применительно к таблицеShippers: SELECT \* FROM Shippers.

Безусловно, конструкция DEFAULT, применяемая для добавления ограничения DEFAULT к существующей таблице, остается такой же, как и при создании таблицы, но в самом операторе модификации таблицы имеется небольшой нюанс. Оператор ALTER и конструкция ADD, предназначенная для добавления ограничения, остаются такими же, как и в других описанных выше операторах, но дополнительно предусма­тривается ключевое слово FOR, которое указывает СУБД SQL Server, какой столбец является целевым для ограничения DEFAULT. Пример такого оператора модифика­ции таблицы приведен ниже.

ALTER TAELE Customers

ADD CONSTRAINT CN\_CustomerDefaultDatelnSystem DEFAULT GETDATEO FOR DatelnSystem

А следующий пример показывает, как задать литеральное значение в качестве применяемого по умолчанию:

ALTER TABLE Customers

ADD CONSTRAINT CN\_CustomerAddress DEFAULT 'UNKNOWN' FOR Address1

В любой таблице может быть задано несколько ограничений DEFAULT, как и всех прочих ограничений, кроме PRIMARY KEY.

Приведем еще два примера создания ограничений.

Создайте таблицу EXAMJVIARKS так, чтобы не допускался ввод в таблицу двух записей об оценках одного студента по конкретным экзамену и предмету обучения и чтобы не допускалось проведение двух экзаменов по любым предметам в один день.

CREATE TABLE exam\_marks1

([exam\_id] numeric(18, 0),

[student id] VARCHAR(50),

subj\_id VARCHAR(50),

mark numeric(18, 0)NOT NULL UNIQUE,

exam\_date date NOT NULL UNIQUE)

Создайте таблицу предметов обучения SUBJECT так, чтобы количество отводимых на предмет часов по умолчанию было равно 36, не допускались записи с отсутствующим количеством часов, поле SUBJ\_ID являлось первичным ключом таблицы и значения семестров (поле SEMESTER) лежали в диапазоне от 1 до 12.

CREATE TABLE SUBJECT2

([subj id] NUMERIC (18,0) PRIMARY KEY,

[subj name] VARCHAR(50),

hour NUMERIC (18,0)NOT NULL CHECK (hour = 36) ,

semestr NUMERIC (18,0) CHECK (semestr < 12))

# 3 Задания для самостоятельной работы

1. Добавьте первичный ключ к одной из существующих таблиц.
2. Создайте таблицу с внешним ключом.
3. Добавьте внешний ключ к одной из существующих таблиц.
4. Осуществите добавление ограничения типа check.
5. Добавьте ограничение в любую из таблиц типа Default.
6. Добавьте ограничение в любую из таблиц типа unique.
7. Создайте таблицу, ссылающуюся саму на себя.

## 4 Контрольные вопросы

1. Дайте определение ограничению.
2. Какие типы ограничений вы знаете?
3. Что такое ограничение домена?
4. Дайте определение ограничению сущности.
5. Какие правила именования ограничений вы знаете?
6. Приведите пример ограничения primary key.
7. Приведите пример ограничения foreign key.
8. Приведите пример таблицы, ссылающейся на саму себя.
9. Приведите пример ограничения unique.
10. Приведите пример ограничения check.
11. Приведите пример ограничения default.

# Лабораторная работа №9 – Сценарии, пакеты, транзакции, блоки команд.

**1 Цель работы**

Ознакомиться с понятиями пакетов, транзакций, блоков команд и принципом блокировки данных. Научиться создавать пакеты и транзакции, производить блокировку данных.

2 Основные теоретические сведения

Пакет – это набор SQL-команд, запускаемых как единое целое. Пакет компилируется только один раз. Пакет заканчивается символом конца пакета (команда GO). Пакеты могут запускаться интерактивно или из файла. Пакет разбирается, оптимизируется, компилируется и выполняется целиком. Если в пакете произойдет ошибка, ни одна из его команд не выполнится. При создании пакетов необходимо учитывать, что некоторые команды могут объединяться в одном пакете, тогда как другие – нет. Например, можно объединять в одном пакете следующие команды:

CREATE DATABASE

CREATE TABLE

CREATE INDEX

Следующие команды объединять нельзя:

CREATE PROCEDURE

CREATE RULE

CREATE DEFAULT

CREATE TRIGGER

CREATE VIEW

Можно создать такой пакет:

CREATE DATABASE...

CREATE TABLE...

GO

Следующий пакет создать нельзя:

CREATE DATABASE...

CREATE TABLE...

CREATE RULE...

CREATE RULE...

GO

Вместо него следует создать несколько пакетов. Это будет выглядеть следующим образом:

CREATE DATABASE...

CREATE TABLE...

GO

CREATE RULE...

GO

CREATE RULE...

GO

При создании пакетов необходимо принимать во внимание следующее:

* Правила и значения по умолчанию не могут быть в пределах одного пакета, и связаны со столбцами, и использованы. То есть хранимые процедуры sp\_bindrule и sp\_bindefault недопустимы в пакете с соответствующими командами INSERT и UPDATE.
* Ограничения на проверку ввода (опция CHECK команд CREATE TABLE и ALTER TABLE) не могут быть, и определены, и использованы в одном пакете. Ограничение CHECK становится актуальным только по завершении пакета.
* В одном пакете нельзя удалить объект и вновь создать его.
* Опции, которые изменяются с помощью команды SET, вступают в действие только после окончания работы пакета. Можно комбинировать команды установок SET и запросы в рамках одного пакета, но нужно помнить, что установки SET не воздействуют на запросы из этого пакета.
* В одном пакете нельзя изменить структуру таблицы и обратиться к её но­вым столбцам.

Транзакция – это последовательность действий, рассматриваемая как единое целое. Каждое выражение SQL – это одно действие, независимо от того, воздействует оно на одну или несколько записей в таблице. В Microsoft SQL Server транзакция может быть безусловной, согласно природе выполняемого выражения, или определенной пользователем. Безусловными транзакциями являются любые команды модификации данных вне рамок определенных пользователем транзакций. При работе с транзакциями необходимо учитывать следующее:

* Все определенные транзакции должны происходить внутри команд BEGIN TRANSACTION и COMMIT TRANSACTION.
* Термин «транзакция» относится только к командам модификации данных.
* Группировка большого количества команд Transact-SQL в одну долго выполняемую транзакцию может увеличить время восстановления (при откате) и привести к проблемам с блокировкой данных и одновременным доступом к ним. Наилучший вариант – команды COMMIT и BEGIN находятся в одном пакете.

Транзакции удобны тем, что все операции, которые входят в них, либо успешно выполняются, либо ни одна из них не завершается. Когда несколько пользователей обращаются к базе данных одновременно, Microsoft SQL Server использует блокировку, для того чтобы гарантировать не перекрывание. Блокировка запрещает пользователям читать данные, которые изменяются другим пользователем, и не позволяет пользователям делать более одного изменения записи за один раз. Блокируются страницы, которые читаются или изменяются во время транзакции, что позволяет избежать проблем при большом количестве транзакций. Уменьшение объема блокировок увеличивает скорость доступа и производительность. Хотя существует много способов уменьшить время и количество блокировок, лучший метод – завершить транзакцию, как только она выполнила все операции, которые планировались. Microsoft SQL Server может применять несколько типов блокировки. В общем случае операции чтения довольствуются мягкой блокировкой, а операции записи требуют монопольной блокировки. Приведенная ниже таблица №5 описывает три типа блокировок, которые использует Microsoft SQL Server для поддержки целостности данных.

Таблица №5 – Типы блокировок

|  |  |
| --- | --- |
| **Тип** | **Описание** |
| Разделяемая | Microsoft SQL Server использует разделяемую блокировку для операций, которые не изменяют и не модифицируют данные, например таких, как выборка с помощью команды SELECT |
| Модификации | Microsoft SQL Server использует блокировку модификации, когда он пытается модифицировать страницу, и позже повышает блокировку модификации на монопольную блокировку страницы, перед тем как действительно совершить изменения |
| Монопольная | Microsoft SQL Server использует монопольную блокировку для модификации данных с помощью таких операций, как UPDATE, INSERT или DELETE |

Несмотря на то, что Microsoft SQL Server выполняет транзакции автоматически, системный администратор может управлять ими с помощью опций команды SELECT и установить уровень изоляции с помощью команды SET. Системный администратор может также управлять блокировкой для всех пользователей, устанавливая пороговый уровень блокировки. Когда запрос выбирает записи из таблицы, SQL Server автоматически генерирует блокировку на уровне страницы. Если запрос выбирает слишком большой процент записей, SQL Server блокирует всю таблицу. Этот процесс называет расширением блокировки (lock escalation); он позволяет ускорить сканирование таблиц и операции с большим набором данных, так как уменьшает чтение при блокировках (locking overhead). Расширение блокировки применяется для команд, но не для транзакций.

Порог блокировки – это максимальное число блокированных страниц перед переходом на табличную блокировку. Если число блокированных страниц выше порогового, то табличная блокировка произойдет, даже если процентный порог расширения блокировки не пройден. Значение порога по умолчанию – 200 заблокированных страниц.

Есть также понятие минимальный порог блокировки. Табличная блокировка произойдет, только если будет достигнут минимальный порог и процентный порог расширения. Минимальный порог расширения блокировки предотвращает переход к табличной блокировке для маленьких таблиц, там, где процентный порог расширения достигается часто. Значение по умолчанию — двадцать заблокированных страниц.

Процентный порог расширения указывает процент заблокированных страниц, необходимый для перехода на табличную блокировку. Значение по умолчанию 0, что указывает на возможность табличной блокировки только при достижении максимального порога расширения.

Уровень изоляции транзакций устанавливается для всего SQL Server. Чтобы задать его, используйте команду SET TRANSACTION ISOLATION LEVEL. Когда вы устанавливаете уровень изоляции транзакции, вы указываете блокировку по умолчанию для всех команд SELECT в сессии. Вы можете переустанавливать уровень для индивидуальных выборок с помощью опций команды SELECT. Чтобы выяснить, какой уровень изоляции использован, применяется команда DBCC USEROPTIONS. В таблице №6 указаны опции командыSET TRANSACTION ISOLATION LEVEL и их назначение.

Так как все транзакции записываются в журнал упреждающего чтения, базы данных SQL Server могут быть восстановлены после таких событий, как прекращение подачи электроэнергии, сбой системного программного обеспечения, «зависание» различных настольных приложений или откат транзакции. Ключевым моментом транзакции является упреждающая запись в журнал транзакций. Когда бы ни поступил запрос на модификацию базы данных, копия, как старого, так и нового состояния части базы данных, затронутой транзакцией, записывается в журнал. Изменения записываются перед тем, как они будут внесены в саму базу данных.

Таблица №6 – Опции команды SET TRANSACTION ISOLATION LEVEL

|  |  |
| --- | --- |
| **Опция** | **Описание** |
| READ COMMITTED | SQL Server будет использовать разделяемую блокировку во время чтения. На этом уровне вы не можете применять «грязное чтение» |
| READ UNCOMMITTED | SQL Server не использует разделяемую блокировку и не поощряет монопольную блокировку. Вы можете экспериментировать с «грязным чтением» |
| REPEATABLE, READ | Указывает, что «грязное чтение», неповторяемые SERIALIZABLE считывания и значения-призраки не могут произойти |

После системных сбоев Microsoft SQL Server использует журнал транзакций для восстановления базы данных в исходном состоянии путем отказа от всех незавершенных транзакций. Помимо этого, SQL Server использует журнал, чтобы удостовериться, что все изменения, связанные с завершенными транзакциями, отражены в базе данных. Пользовательские транзакции, также называемые определенными, дают пользователю контроль над транзакциями. При работе с ними необходимо учесть следующее:

* Используя много команд в одной транзакции, можно улучшить производительность, так как система обрабатывает сразу все команды в транзакции, а не каждую команду в отдельности.
* Каждый пользователь может определить транзакцию. Не требуется никаких разрешений ни для одного выражения транзакции.
* Внутри пользовательских транзакций не могут создаваться временные таблицы.
* Имена транзакций должны удовлетворять правилам именования объектов.
* Некоторые системные процедуры не могут быть использованы внутри транзакций, так как они создают временные таблицы.

Если транзакция должна быть прервана до своего завершения – или по причине сбоев, или из-за действий пользователя, – то все её команды должны быть отменены. Транзакции могут быть прерваны с помощью команды ROLLBACK TRANSACTION. Эта команда должна быть выдана до команды COMMIT TRANSACTION. Можно «откатить» всю транзакцию или её часть. Естественно, никто не может «откатить» транзакцию после её завершения.

Синтаксис команды ROLLBACK TRANSACTION:

ROLLBACK TRANSACTION [<Имя транзакций> | <Имя точки сохранения>]

Точка сохранения представляет собой маркер, размещаемый пользователем внутри транзакции, чтобы отметить место, к которому можно «откатить» транзакцию (не считая, естественно, начала – BEGIN TRANSACTION). Точки сохранения определяются внутри транзакции с помощью команды SAVE TRANSACTION.

Синтаксис команды SAVE TRANSACTION:

SAVE TRANSACTION <Имя точки сохранения>

Глобальная переменная @@TRANSACTION используется для того, чтобы определить, есть ли в текущий момент какие-нибудь открытые транзакции и как много уровней вложенности они имеют. Глобальная переменная @@TRANSACTION равна нулю, когда нет открытых транзакций. Команда BEGIN TRANSACTION увеличивает @@TRANSACTION на одну единицу, a ROLLBACK TRANSACTION уменьшает. Откат к точке сохранения не уменьшает @@TRANSACTION.

Именование вложенных транзакций не воздействует на внешние транзакции. Только имя первой (внешней) транзакции регистрируется системой, и откат к любому другому имени, кроме точки сохранения, генерирует ошибку. Все команды, выполненные перед попыткой ошибочного отката, «откатываются», когда происходит эта ошибка. Ни одна часть транзакции не завершается, пока не выполнится COMMIT TRANSACTION внешней транзакции. Обычно ошибки неявной вложенности транзакций происходят, когда хранимые процедуры или триггеры, имеющие команды BEGIN...COMMIT, вызывают друг друга.

При создании хранимых процедур используют команды управления. Без них отдельные команды выполняются последовательно, в том порядке, как они размещены внутри процедуры. Команды управления и цикла позволяют объединить нужные команды в группу. В таблице №7 приведены команды управления потоком выполнения Microsoft SQL Server.

Таблица №7 – Команды управления и цикла

|  |  |
| --- | --- |
| **Команда** | **Описание** |
| DECLARE | Объявляет локальные переменные и курсоры |
| RETURN | Независимый выход из процедуры |
| RAISERROR | Возвращает сообщение из таблицы sysmessages или динамически созданное пользователем сообщение |
| PRINT | Выводит сообщение об ошибке на экран |
| CASE | Позволяет выражению возвращать разные значения в зависимости от условия |
| BEGIN...END | Определяет блок команд |
| IF...ELSE | Определяет условное и альтернативное выполнение |
| WHILE | Повторяет вычисление до тех пор, пока условие истинно |
| BREAK...CONTINUE | BREAK выходит из самого внутреннего цикла WHILE, a CONTINUE вновь продолжает выполнение цикла WHILE |

Локальные переменные объявляются с помощью команды DECLARE. Начальное значение им присваивается с помощью команды SELECT, а затем они используются внутри пакета или процедуры, в которых объявлены.

Синтаксис команды DECLARE:

DECLARE @<Имя переменной> <Тип данных> [, @<Имя переменной> <Тип данных>...]

Ниже приводится пример объявления и инициализации. Обратите внимание на то, что перед переменными всегда стоит знак @.

DECLARE @fam(30)

SELECT @fam = ' '

WHILE @fam is not null

BEGIN

SELECT @fam = min (first\_name)

FROM customer

WHERE first\_name > @fam

PRINT @fam

END

Команда RETURN прекращает работу запроса или процедуры безусловно. После команды RETURN остальные команды не выполняются. SQL Server резервирует 0 для индикации успешного завершения и отрицательные числа от 1 до 99, чтобы указать на сбой. Если не используется никакого пользовательского значения, SQL Server возвращает то, что считает нужным. На сегодня используются значения от 0 до 14.

Синтаксис команды RETURN:

RETURN [integer\_<Выражение>]

Для того чтобы использовать в дальнейшем возвращенное целое число, необходимо запустить хранимую процедуру в следующем формате:

EXEC[ute] @<Имя переменной> = {[[[<Имя сервера>.]<Имя БД>.]<Имя владельца>. ] <Имя процедуры>[;<Номер процедуры>] | @<Имя переменной>}

Команда RAISERROR возвращает определенное пользователем сообщение об ошибке и устанавливает системный флаг с происшедшей ошибкой. Команда RAISERROR имеет следующий синтаксис:

RAISERROR ({<Идентификатор> | <Сообщение>}, <Уровень>, <Состояние>, <Аргумент 1> [, <Аргумент 2>]]) [WITH LOG]

Опция WITH LOG заносит информацию об ошибке в журналы ошибок и событий. Она необходима для сообщений с уровнем строгости (severity) от 19 до 25 и может быть использована только системным администратором. Расширенная системная хранимая процедура xp\_eventlog предоставляет схожую функциональность.

Команда PRINT возвращает определенное пользователем сообщение клиенту. При её использовании необходимо учитывать следующее: размер сообщения не может превышать 255 символов, локальные или глобальные переменные могут быть типа char или varchar, форматированное сообщение создается с помощью операторов конкатенации.

Синтаксис команды:

PRINT { <'Текст'> | @<Локальная переменная> | @@<Глобальная переменная>}

Следующий пример возвратит число соединений с момента последнего старта SQL Server:

PRINT convert (char(4), @@connections)

Выражение CASE позволяет возвращать разные значения в зависимости от условия. CASE разрешается использовать везде, где допустимы выражения.

Выражение CASE имеет два синтаксиса: простой и поисковый.

Простой синтаксис:

CASE <Выражение>

WHEN <Вариант 1> THEN <Выражение 1>

[[WHEN <Вариант 2> THEN <Выражение 2>] [...]]

[ELSE <Выражение N>]

END

Поисковый синтаксис:

CASE

WHEN <Вариант 1> THEN <Выражение 1>

[[WHEN <Вариант 2> THEN <Выражение 2>] [...]]

[ELSE <Выражение N>]

END

Принципиальное различие между двумя видами синтаксиса в том, что в поисковом синтаксисе после ключевого слова WHEN мы можем строить логическое выражение, в котором допустимы операторы AND и OR. По данному логическому выражению и происходит поиск. При простом CASE – выражении проверяются только строго эквивалентные выражения.

Данный пример по аббревиатуре выводит полное название штата вместе с суммой заказов, размещенных в этом штате. Название штата выводится на русском языке.

SELECT штат =

CASE st

WHEN 'NY' THEN 'Нью-Йорк'

WHEN 'CA' THEN 'Калифорния'

WHEN 'FL' THEN 'Флорида'

WHEN 'OH' THEN 'Огайо'

ELSE 'Это не штат'

END,

Сумма = Sum(ord\_sum)

FROM usa\_order

GROUP BY st

### Блоки команд

Код между командами BEGIN и END называется блоком. Блоки используются с операторами управления и цикла, такими как IF... ELSE и WHILE. Если не используются BEGIN и END, выполняется только первая команда, которая следует сразу за IF... ELSE или WHILE.

Блоки BEGIN…END могут быть вложенными.

Синтаксис блока:

BEGIN

{<Команда> | <Блок команд>}

END

Простой пример транзакции:

BEGIN TRANSACTION

UPDATE account

SET sum\_ = sum\_ + 10000

WHERE key\_customer = 12

AND date\_write = '1/13/97'

COMMIT TRANSACTION

Блок IF... ELSE позволяет создавать код, который выполняется по условию. Команда T-SQL, следующая за ключевым словом IF, выполняется в том случае, если требования условия удовлетворяются. Опционное ключевое слово ELSE добавляет альтернативные команды, которые будут выполняться в том случае, если условие IF не удовлетворяется. При использования ключевых слов IF... ELSE необходимо принимать во внимание следующее: конструкция IF... ELSE может использоваться в пакетах, хранимых процедурах и дополнительных запросах, конструкция IF... ELSE может быть вложенной как по IF, так и по ELSE. Никаких ограничений на количество уровней вложенности нет.

Синтаксис команды IF... ELSE:

IF <Условие>

{<Команда> | <Блок команд>}

[ELSE [<Условие>]

{<Команда> | <Блок команд>}]

Конструкция WHILE задает условие, при котором будет повторяться команда T-SQL или блок BEGIN... END. Выполнение команд будет повторяться до тех пор, пока условие истинно. Поэтому существует опасность бесконечного цикла. Выполнение команд в цикле WHILE может контролироваться изнутри цикла с помощью ключевых слов BREAK и CONTINUE.

Синтаксис команды WHILE:

WHILE <Условие>

{<Команда> | <Блок команд>}

[BREAK]

{<Команда> | <Блок команд>}

[CONTINUE]

{<Команда> | <Блок команд>}

Команда EXECUTE выполняет системную, пользовательскую хранимую или расширенную хранимую процедуру. Она также поддерживает выполнение символьной строки внутри пакета SQL. Все части строки EXECUTE должны состоять из символьных данных; числовые данные должны быть конвертированы. Функции не могут использоваться для создания строки, которая должна выполняться, однако любые команды Transact-SQL, включая функции, можно передавать в строку. Разрешается создавать вложенные команды EXECUTE. Команда EXECUTE может включать: строку, локальную переменную, содержащую строку, конкатенацию строк и строковых локальных переменных.

Синтаксис команды EXEC[ute]:

EXEC[ute] (string-literal | @string-var)

Следующий ниже пример создает хранимую процедуру, которая, в свою очередь, использует команду EXECUTE. Хранимая процедура выбирает все пользовательские таблицы из базы данных и выполняет команду UPDATE STATISTICS над каждой из них.

CREATE PROC DBstats

AS

SET NOCOUNT ON

SELECT '\*\*\* Database: ' + db\_name() + ' \*\*\*'

DECLARE @next char(30)

SELECT @next = ' '

WHILE @next IS NOT NULL

BEGIN

SELECT @next *=* MIN (name)

FROM sysobjects

WHERE type = 'U'

AND name > @next

IF @next IS NOT NULL

BEGIN

EXEC ('UPDATE STATISTICS ' + @next)

SELECT 'Statistics Updated on: ' + @next

END

END

Таким образом, создавая строку, которая передается команде EXECUTE, мы избавляем себя от неоднократного исполнения команды UPDATE STATISTICS с каждой таблицей.

Пример хранимой процедуры, возвращающей значение:

CREATE PROC calc\_sum

AS

DECLARE Qallsum money

SELECT @allsum = sum (sum\_)

FROM account

WHERE date\_write = GETDATEO

RETURN @allsum

# 3 Задания для самостоятельной работы

1. Средствами MS SQL Server создайте пакет команд.
2. Определите одну транзакцию, содержащую несколько команд.
3. Создайте несколько транзакций, откомпилируйте их и установите для них различные уровни изоляции.
4. Организуйте выполнение основных типов блокировок.
5. Создайте хранимую процедуру.
6. Осуществите вывод сообщения об ошибке или любого пользовательского сообщения.
7. Создайте пакет с использованием выражения контроля условия и несколько блоков команд, включая контроль цикла.

#### 4 Контрольные вопросы

1. Дайте определение пакету данных.
2. Приведите примеры команд, которые можно использовать в пакете и которые нельзя.
3. Дайте определение транзакции.
4. Поясните синтаксис блокировки данных.
5. Какие виды блокировок данных вам известны? Приведите примеры.
6. Что такое уровень изоляции транзакции?
7. Какие команды допустимы в транзакциях?
8. Приведите пример команды управления циклом.
9. Какая команда позволяет выходить из процедуры принудительно?
10. Какая команда выводит строковое сообщение пользователю?
11. Дайте определение блоку.

# Лабораторная работа №10 - Администрирование SQL Server. Определение баз данных и журналов транзакций. Управление доступом к данным.

## 1 Цель и порядок работы

Освоить приемы управления устройствами, базами данных и доступом к данным.

## 2 Основные теоретические сведения

В данной лабораторной работе собраны основные методы работы с БД и таблицами в них.

### 

### 2.1 Управление базами данных. Основные понятия

### 2.1.1 Создание баз данных и журналов транзакций

Для выполнения каких-либо действий с объектами SQL Server предварительно необходимо соединиться с требуемым сервером. Запустите SQL Server и в окне диалога Register Server, укажите необходимые параметры для регистрации сервера. В дальнейшем соединение с этим сервером будет происходить автоматически.

Для создания новой БД вы можете выполнить одно из следующих действий:

1. выделить папку “*Databases*”;
2. выбрать в меню *Action* (или в контекстном меню) команду | *New Database*, откроется окно диалога *Database properties*.

С помощью этого окна вы можете указать имя создаваемой БД в поле “*Name*” (имя БД должно быть уникальным). БД может располагаться на различных устройствах в нескольких файлах, для этого в столбце “*file name*” указывается логическое имя файла; столбец “*Location*” указывает на его местоположение, на физических дисках, и необходимый начальный объем в МБ указывается в столбце “*Initial size*”. Можно так же установить автоматическое приращение файла в мегабайтах или в процентном соотношении от свободного пространства диска (группа *File growth*) и ограничить максимальный размер БД (группа *Maximum file size*).

При создании БД SQL Server выполняет следующие действия:

* выделяется пространство на жёстком диске или нескольких дисках;
* в системную таблицу ***sysdatabases*** БД ***master*** добавляется запись о создаваемой базе данных;
* выполнятся инициализация страниц базы данных;
* в новую базу данных копируется БД ***model***.

Итак, созданная БД содержит стандартные объекты SQL Server:

**Database Users** (Пользователи БД). Единственный пользователь, существующий в стандартной системе, – это *dbo* (DatabaseOwner).

**Database Roles** (роли БД). Определено 10 ролей.

**Tables** (таблицы). В стандартную модель включено семнадцать таблиц, на основе которых вы можете строить свои собственные таблицы.

**Views** (виды). Предусмотрено 20 видов, которые могут отображать информацию, извлекаемую из БД.

**Stored Procedures** (хранимые процедуры). В модели БД таковых нет.

**Rules** (правила). В модель БД правила не включены.

**Defaults** (стандартные установки). Если для создаваемых БД вы хотите определить стандартные установки, то вам придется добавить их самостоятельно в модель БД.

**User-Defined Datatypes** (типы данных, определяемые пользователем).

**Database Diagrams** (диаграммы БД). В модели БД таковых нет.

*Транзакция*– логическая единица работы, которую должен выполнить SQL Server. Она может включать один или несколько операторов языка SQL; при этом серверу должно быть понятно, какие операторы из пакета являются частью транзакции.

Пример выполнения нескольких операций в одной транзакции.

Create table TABLE\_A(/\* создание таблицы \*/

X smallint null /\* с полем Х \*/

Y smallint null /\* и с полем Y \*/

)

GO /\* выполнить \*/

Create table TABLE\_B( /\* создание таблицы В \*/

Z smallint null /\* с полем Z \*/

)

GO /\* выполнить \*/

Begin Tran /\* начало транзакции \*/

Update TABLE\_A /\* обновление таблицы А \*/

Set X := X+1 /\* увеличить значение поля на 1 \*/

Where Y = 100 /\* где в поле Y значение 100 \*/

Update TABLE\_B /\* обновление таблицы В \*/

Set Z := Z+1 /\* увеличить значение на 1 \*/

If @@rowcount = 0 or @@error !=0 /\* если строки не были изменены \*/

Begin

Rollback Tran /\* то производим откат транзакции \*/

Print ‘Ошибочка!’ /\* выдаем сообщение \*/

Return /\* возвращаемся из проц. \*/

End

Commit Tran /\* завершение транзакции \*/

*Журналы транзакций* – это рабочие области, которые SQL Server применяет для записи состояния системы «до и после» выполнения транзакции. Эта информация может использоваться для отмены выполнения транзакции и для восстановления БД, если возникнет такая необходимость.

Как и упоминалось в ранее, создание журнала транзакций аналогично созданию файлов БД, (вкладка Transaction Log диалогового окна “*Database properties*”).

Для создания БД можно также воспользоваться командой Transact-SQL. Напомним синтаксис этой команды:

CREATE DATABASE database\_name

[ON [PRIMARY] [<filespec> [,...n]]

[LOG ON {<filespec> [,...n]}]

[FOR LOAD | FOR ATTACH ]

<filespec> ::=

([ NAME = logical\_file\_name,]

FILENAME = 'os\_file\_name'

[, SIZE = size]

[, MAXSIZE = { max\_size | UNLIMITED } ]

[, FILEGROWTH = growth\_increment] )

Например:

CREATE DATABASE NewDatabase -- создание БД, которая появится в Enterprise Manager:

ON -- задает список файлов БД

(

NAME = NewDatabase\_dat, -- логическое имя файла БД

FILENAME = 'C:\NewDatabase\_dat.mdf', -- имя на физическом диске

SIZE = 4MB, -- начальный размер БД

FILEGROWTH = 1MB -- приращение БД

)

LOG ON -- задает список файлов журнала транзакций

(

NAME = NewDatabase\_log, -- логическое имя файла журнала транзакций

FILENAME = 'C:\NewDatabase\_log.ldf', -- имя на физическом диске

SIZE = 2MB, -- начальный размер файла

FILEGROWTH = 1MB -- приращение

)

### 2.1.2 Определение размера БД

Единственным способом уменьшения размера БД является его удаление и создание БД с другими параметрами. Можно лишь изменить приращение и ограничить максимальный размер файлов БД и ЖТ. Для увеличения можно добавить новое устройство, на котором будет располагаться БД.

### 2.1.3 Выбор размера БД

При выборе размера БД следует учитывать объём и непосредственно данных (размеры системных таблиц, таблиц с данными, индексов), и журнала транзакций. Не стоит задавать слишком большой размер БД, т.к. в SQL Server можно легко увеличить её размер в дальнейшем, но значительно труднее уменьшить уже установленную величину.

В журнале транзакций записываются все изменения, выполняемые в БД. Основными факторами, влияющими на объем журнала транзакций, являются: активность работы с данными, частота выполнения архивирования, объем данных, для которых проводится одновременное обновление. Сюда же можно отнести и стиль программирования. Пристрастие к «длинным транзакциям» требует большого журнала транзакций. Если не хотите доводить объем журнала транзакций до немыслимых пределов, чаще завершайте транзакции при обновлении данных. В общем случае для журнала транзакций следует выделить не менее 20% от объема БД.

При тщательном планировании системы обработки данных можно достаточно точно определить пространство, требуемое для БД, если руководствоваться следующими соображениями:

Пространство, занимаемое данными, определяется количеством таблиц, входящих в БД. Для каждой таблицы можно определить число столбцов и общий размер одной записи (зная ее тип данных). Далее необходимо определить, сколько записей поместится на одной странице данных и, таким образом, сколько страниц займет каждая таблица. На каждой странице для данных отводится 1084 байта. Размер одной записи не превышает объема одной страницы, т.к. запись не может переходить на другую страницу.

Пространство, занимаемое журналом транзакций, определяется менее точно, т.к. оно зависит от частоты изменения данных и создания архивных копий. Здесь лучше всего руководствоваться приведенными выше рекомендациями, основываясь на размере БД.

Пространство, занимаемое индексами, зависит от типа индексов (кластерные или некластерные) и фактора заполнения страниц памяти (fill factor). Фактор заполнения позволяет резервировать место на странице для помещения новых записей при перестройке индекса без создания новой страницы, что существенно повышает производительность системы, но требует большего пространства для БД. Для определения фактического объема индекса необходимо размер страницы (2016 байт) умножить на значение фактора заполнения. Так, если фактор заполнения равен 50%, то на одной странице поместится 1008 байт. При окончательном расчете прибавьте 10-20% на пространство, необходимое для перестройки индекса.

### 2.1.4 Владелец БД

Чаще созданием БД занимается системный администратор, а проблемы в дальнейшей модификации возлагаются на другого пользователя, отвечающего за работу конкретной системы. Для этого данному пользователю необходимо передать полномочия создателя БД, т.е. сделать его владельцем БД. Выполнить такую задачу непосредственно в SQL Server нельзя. Придется прибегнуть к стандартной хранимой процедуре Transact-SQL.

Синтаксис:

sp\_changedbowner [@loginame =] 'login'[,[@map =] remap\_alias\_flag]

Пример использования:

sp\_changedbowner SoftMan;

где параметр *SoftMan* должен быть правильным именем для входа в систему. Для выполнения процедуры нажмите кнопку с зеленой стрелкой в окне Query Analyzer.

### 2.1.5 Определение опций БД

Вернемся к окну диалога *Edit Database* и откроем вкладку *Options*. Вы можете установить для БД следующие параметры:

**Select into/Bulk Copy** – позволяет (быстро, без записи в журнал транзакций) выполнить такие действия с БД, как загрузка таблиц, выполнение команд WRITETEXT или UPDATETEXT.

**Columns Null by Default** – значением по умолчанию будет NULL.

**No Checkpoint on Recovery** – предотвращает добавление контрольной записи в БД после её восстановления из архива. Это позволяет выполнять последовательное восстановление данных из различных архивов.

**Truncate Log on Checkpoint** – при выполнении процедуры проверки автоматически стирает уже принятые транзакции из журнала транзакций. В этом случае выполнение команды DUMP TRANSACTION приведет к ошибке. Если этот флажок не установлен, то необходимо принудительно очищать журнал транзакций для предотвращения его переполнения.

**Single User** – включает монопольный режим доступа.

**DBO Use Only** – позволяет открыть БД только её владельцу.

**Read Only** – предотвращает выполнение любых модификаций БД.

### 2.1.6 Размещение сегментов и объектов БД

Если данные БД размещаются на различных физических устройствах, то важно понимать, где SQL Server будет располагать таблицы, индексы и другие объекты. Для журнала транзакций мы можем явно, при создании БД, указать место его расположения. По умолчанию SQL Server записывает объекты БД на свободное пространство дисков, не определяя специально их размещение. В то же время жесткая привязка объектов БД к сегментам позволяет:

* повысить производительность работы системы;
* контролировать расположение объектов и рост их объема.

Для повышения производительности рекомендуется располагать сегменты на физически разных дисках, а ещё лучше на устройствах с разными контроллерами. За счёт распараллеливания чтения/записи данных можно достичь повышения производительности. Параллельные операции чтения/записи эффективны при расположении на различных дисках таблиц и некластерных индексов, различных частей очень больших таблиц, текстовых данных и изображений.

Помимо сегмента для журнала транзакций (***logsegment***), при создании БД автоматически создаются ещё два сегмента: ***default*** – для таблиц и индексов и ***system*** – для системных таблиц, включающих, в частности, описание всех объектов базы данных. Эти два сегмента располагаются в области данных на всех отведенных для БД устройствах. Для расположения областей данных БД на разных сегментах необходимо использовать стандартные хранимые процедуры (таблица №8).

Таблица №8 - Стандартные хранимые процедуры для работы с сегментами

|  |  |
| --- | --- |
| *Процедура и синтаксис* | *Назначение* |
| sp\_addsegment <Имя сегмента> | Добавление нового сегмента в БД |
| sp\_extendsegment <Имя сегмента> <Имя объекта> | Расширение сегмента на другое устройство |
| sp\_dropsegment <Имя сегмента> <Имя объекта> | Удаление сегмента на указанном устройстве |
| sp\_helpsegment <Имя сегмента> | Получение информации обо всех сегментах (или указанном сегменте) |

Сегмент является принадлежностью конкретной БД, поэтому перед выполнением процедур её нужно открыть. В указанном сегменте не могут располагаться объекты других БД, пусть даже расположенных на том же устройстве. Таким образом, сегменты разных баз данных могут иметь одинаковые имена.

### 2.1.7 Удаление базы данных

При удалении БД вместе с ней удаляются и все объекты содержащиеся в ней и освобождается пространство, занимаемое БД на диске.

Для удаления с помощью SQL Server выполните следующее:

* выберете сервер, на котором находится БД, и выделите папку *Databases*.
* на правой панели появится список баз данных, выделите нужную БД;
* в контекстном меню выберете команду Delete.

Синтаксис команды удаления БД на языке Transact-SQL выглядит следующим образом:

DROP DATABASE database\_name, [database\_name,...]

Пример:

DROP DATABASE NewDatabase;

где *NewDatabase* имя удаляемой БД из примера создания БД.

### 2.2 Работа с таблицами

### 2.2.1 Создание таблицы

Выберите требуемую БД и откройте эту папку, а затем щелкните правой кнопкой мыши на объекте *Tables*. В контекстном меню выберите команду *New Table*. На экране появится окно диалога.

Здесь вы можете сформировать поля для хранения данных и установить для них необходимые атрибуты.

Для формирования полей необходимо заполнить таблицу, которая имеет следующие столбцы:

**Column Name** – имя столбца (поля);

**Data Type** – тип данных поля; выбирается из списка;

**Length** – размер, указывается только для тех типов данных, для которых он не определяется автоматически;

**Precision** – точность;

**Scale** – масштаб;

**Allow Nulls** - указывает на допустимость использования элементов NULL;

**Default Value** – значение по умолчанию;

**Identity** указывает, что данная колонка будет использоваться в качестве счётчика;

**Identity Seed** – базовое значение для счётчика;

**Identity Increment** – приращение базы;

В SQL Server можно так же создавать два типа временных таблиц – локальные и глобальные. Чтобы создать локальную временную таблицу, нужно на месте первого символа в её имени поставить знак фунта (#). Локальная таблица доступна только для того сеанса, в котором она была создана. По окончании сеанса таблица автоматически удаляется. В имени глобальной временной таблицы необходимо поставить два знака фунта (#). Такая таблица будет доступна для всех сеансов. Таблица автоматически удаляется по окончании последнего сеанса.

Создание таблиц с помощью Transact-SQL выглядит следующим образом:

CREATE TABLE

[

database\_name.[owner].| owner.

] table\_name

(

{<column\_definition>

| column\_name AS computed\_column\_expression

| <table\_constraint>

} [,...n]

)

[ ON {filegroup | DEFAULT} ]

[ TEXTIMAGE\_ON {filegroup | DEFAULT} ]

<column\_definition> ::= { column\_name data\_type }

[ [ DEFAULT constant\_expression ]

| [ IDENTITY [(seed, increment ) [NOT FOR REPLICATION] ] ]

]

[ ROWGUIDCOL ]

[ <column\_constraint>] [ ...n]

Например:

CREATE TABLE NewDatabase.dbo.tlbSupplers -- Создание таблицы в БД NewDatabase

(

SupID int, -- создать поле SupID типа int

FirstName char(35), -- создать поле FirstName типа char(35)

LastName char(35), -- создать поле LastName типа char(35)

Badge int -- создать поле Badge типа int

)

INSERT INTO NewDatabase.dbo.tlbSupplers -- добавить в таблицу tlbSupplers

-- значения: SuppID: 1; FirstName: alex; LastName: molcovich; Badge: 2000

VALUES (1,'alex','molcovich',2000)

SELECT \* FROM NewDatabase.dbo.tlbSupplers -- вывести все поля таблицы

### 2.2.2 Модификация таблицы

При необходимости внесения изменений в существующую таблицу выберите её в списке объектов окна SQL Server и нажмите правую кнопку мыши. В контекстном меню для выбора будут доступны следующие команды:

**New Table** – команда создания новой таблицы;

**Desine Table** – открывает окно в котором можно выполнить изменения в таблице (окно тоже, что и при создании таблицы);

**Open Table** – открытие таблицы;

**Full-Text Index Table** – полнотекстовая таблица индексов;

**Manage Indexes** – открывает окно диалога *Manage Indexes*, с помощью которого можно создавать или изменять параметры существующих индексов. Можно создать индекс немедленно или отложить эту операцию;

**Manage Triggers** - открывает окно диалога *Manage Triggers*, с помощью которого можно создавать или изменять параметры существующих триггеров;

**Manage Permissions** – открывает окно диалога *Object Permissions*, с помощью которого можно установить соответствующие разрешения для работы пользователей с данной таблицей.

**Import/Export Data** – импорт/экспорт данных;

**Create New Publication** – публикация, т.е. отправка информации в другие системы;

**Generate SQL Scripts** – генерация SQL запросов;

**Display Dependencies** – открывает окно диалога *Generate SQL Scripts*, которое позволяет установить параметры при создании программы для выполнения определенных действий типа построения таблицы, индексов, связей и т.п.

**Copy** – копирование таблицы;

**Delete** – удаляет таблицу;

**Rename** – переименовывает таблицу;

**Properties** – отображение свойств таблицы.

В окне диалога *Manage Indexes* существующие индексы перечисляются в раскрывающемся списке *Index*. Для создания нового индекса выберите в этом списке элемент *New Index* и укажите имя создаваемого индекса. После этого в списке *Available Columns in Table* выделите столбцы, которые необходимо включить в индекс, и нажмите кнопку *Add*. Они будут перенесены в список *Column in Index (Key)*.

В области *Index Attributes* можно определить следующие характеристики создаваемого индекса:

**Unique Keys** – уникальный индекс.

**Ignore Duplicate Keys** – позволяет при уникальном индексе при добавлении или изменении данных не допускать появления повторяющегося значения ключа. Такие данные в таблицу записаны не будут, и появится предупреждающее сообщение.

**Clustered** – кластерный индекс.

**Ignore Duplicate Rows** – позволяет для кластерного неуникального индекса не допускать добавления повторяющихся записей. При попытке добавить такую запись в таблицу появится сообщение об ошибке.

**Allow Duplicate Rows** позволяет для кластерного неуникального индекса иметь в таблице повторяющиеся записи.

**Sorted Data** – сортировка данных не выполняется, т.к. в этом случае подразумевается, что данные уже отсортированы в требуемом порядке. Используется только для индексов кластерного типа. Если данные не расположены в требуемом порядке, индекс создан не будет.

**Sorted Data Reorg** – при создании индекса будет выполнена сортировка данных.

**Fill Factor** – процент заполнения каждой страницы данными для резервирования места под будущие изменения.

Обратите внимание, что в остальных полях окна диалога *Manage Indexes* содержится большое количество полезных параметров, в основном касающихся используемого пространства.

### 2.3 Автоматизация процесса администрирования БД

В SQL Server для облегчения планирования различных действий, связанных с эксплуатацией БД, в SQL EM включен мастер планирования обслуживания БД – Database Maintenance Plan. Мастер планирования позволяет в диалоговом режиме задать подавляющее большинство действий, которые требуется выполнить в процессе эксплуатации БД: проверку размещения данных, соответствие данных и индексов системным данным, обновление данных для оптимизатора, реорганизацию страниц данных и страниц индексов, формирование архивных копий БД и журнала транзакций. Установленные действия могут выполняться автоматически через определенное количество дней или недель.

Мастер планирования выполнен в традиционном для Windows стиле и предусматривает десять шагов, при выполнении которых в диалоговом режиме формируется план поддержки эксплуатации БД.

Для запуска мастера откройте SQL EM и нажмите самую правую кнопку на его панели инструментов Database Maintenance Wizard.

**Шаг 1: Introduction**

На этом шаге необходимо указать базу данных, для которой вы хотите спланировать выполнение тех или иных действий. Для перехода к следующему шагу нажмите кнопку *Next*.

**Шаг 2: About the Data in Your Database**

На этом шаге необходимо определить информацию о выбранной БД. Составление плана требует, чтобы вы имели представление о том, как часто данные в БД пополняются или изменяются.

В области *Data Volatility* необходимо выбрать один из параметров, наиболее точно характеризующих интенсивность изменения данных в выбранной БД в течение дня:

***Data is never changed*** – данные никогда не изменяются;

***Less than 5%*** – менее чем на 5%;

***More than 5% but less than 20%*** – более 5%, но менее 20%;

***More than 20%*** – более 20%.

В области Data Growth необходимо выбрать один из параметров, наиболее точно характеризующих интенсивность добавления новых данных в выбранную БД в течение дня:

***No new data is ever added*** – данные никогда не добавляются;

***Less than 2%*** – менее чем на 2%;

***More than 2% but less than 10%*** – более 2%, но менее 10%;

***More than 10%*** – более 10%.

**Шаг 3: Data Verification**

На этом шаге проверяется корректность связей между данными, правильность расположения страниц и надёжность считывания данных с диска.

Вы можете предусмотреть выполнение следующих проверок:

***Check data linkage*** – страницы с данными проверяются на соблюдение требуемой последовательности их расположения и на корректность считывания и записи информации. В пользовательской программе этому пункту соответствует команда DBCC CHECKDB.

***Check index linkage*** – страницы с индексами проверяются на соблюдение требуемой последовательности их расположения. Несоблюдение этой последовательности может привести к ошибкам при выполнении запросов. Страницы с индексами также будут проверяться на корректность считывания и записи информации. В пользовательской программе этому пункту соответствует команда DBCC CHECKDB.

***Check data allocation*** – страницы с данными проверяются на правильность внутренней информации. Анализируется корректность доступа к используемым и неиспользуемым страницам, т.к. в противном случае возможно возникновение ошибок при выполнении обращений к страницам и потеря данных. В пользовательской программе этому пункту соответствует команда DBCC NEWALLOC.

***Check index allocation*** – страницы с индексами проверяются на правильности внутренней информации. Анализируется корректность доступа к используемым и неиспользуемым страницам индекса. В программе этому пункту соответствует команда DBCC NEWALLOC.

***Check text/image data allocation*** – проверяется правильность расположения и связей страниц, на которых размещены данные типа ***text*** или ***image***. В программе этому пункту соответствует команда DBCC TEXTALL.

***Check system data*** – системные таблицы проверяются на правильность информации и связей между ними. В программе этому пункту соответствует команда DBCC CHECKCATALOG.

***Add the data verification test to*** – определяет, к какому плану выполнения проверок относятся выбранные опции. Для ежедневных проверок выберите кнопку *daily maintenance plan*; для еженедельных – *weekly maintenance plan*.

**Шаг 4: Data Optimization**

На этом шаге определяются параметры оптимизации расположенных данных.

При обновлении, удалении или добавлении данных страницы вскоре становятся заполненными чрезвычайно неравномерно. Оптимизация сводится к переносу данных между страницами для их равномерного заполнения в соответствии со сделанными установками. Затем производится обновление метаданных SQL Server в соответствии с изменившимся расположением табличных данных. Оптимизация позволяет существенно повысить скорость работы SQL Server.

Для установки условий оптимизации вы можете использовать следующие опции:

***Update optimizer information*** – каждая страница с данными в выбранной БД будет проверяться на соответствие расположения ключей в данных и индексах. Впоследствии эта информация используется для оптимизации запросов. Эта опция соответствует команде UPDATE STATISTICS.

***Reorganize data and index pages*** – индексы всех таблиц будут перестроены. Это крайне необходимо, когда страницы с данными заполняются полностью и их необходимо делить для размещения новых данных. Процесс разделения страниц увеличивает размер таблиц, но самое неприятное это то, что увеличивается время выполнения запросов и обновления данных. Перестройка индексов позволяет восстановить необходимую степень заполнения страниц на основе значения, установленного ранее (для этого установите флажок *Maintain original free space*) или выбранного в данный момент в счетчике *Change free space to*. Величина свободного пространства определяется как 100 минус фактор заполнения. Эта опция соответствует команде DBCC DBREINDEX. Фактор заполнения программным путем можно определить с помощью опции FILLFACTOR команды CREATE INDEX.

***Add the data optimizations to*** – определяет необходимость добавления выбранных опций к ежедневному или еженедельному плану.

**Шаг 5: Data Protection**

На этом шаге планируется создание архивной копии данных. Вы можете выбрать ежедневное или еженедельное архивирование данных в зависимости от интенсивности их обновления, наличия дублирующей информации, её важности для поддержания производственного процесса и т.д.

Выберите одно из следующих действий:

***Don’t back up the database*** – не создавать архивную копию;

***Back up the database daily*** – создавать архивную копию ежедневно;

***Back up the database weekly*** – создавать архивную копию еженедельно.

Установите флажок *Only do the backup if*, если вы считаете необходимым создание архивной копии только в случае успешного завершения процесса верификации данных. Это гарантирует качество создаваемой копии, достаточное для восстановления данных в случае их потери.

**Шаг 6: Backup Destination**

На этом шаге выбирается место расположения архивной копии. Она может быть создана на диске (кнопка *Disk*) или магнитной ленте (кнопка *Type*). Для диска с помощью кнопки справа можно выбрать папку размещения архивной копии. Магнитную ленту можно выбрать с помощью раскрывающегося списка.

Установите флажок *Delete files older than* для удаления архивной копии, с момента создания которой прошло более указанного в счётчике количества недель.

Имя файла архивной копии будет сгенерировано автоматически в соответствии со следующим шаблоном:

<ИМЯ БД>\_DB\_DUMP.YYYYMMDD, где YYYY – год, MM – месяц, DD – день.

Например:

TEST\_DB\_DUMP.19970701

На магнитную ленту непосредственно записываются страницы с данными, и имя файла не задается.

**Шаг 7: When to Run, Who to Notify**

На этом шаге вы определяете план создания архивной копии и указываете на необходимость генерации отчета и посылки оповещения по электронной почте после создания архивной копии.

**Шаг 8: Wizard Complete!**

На этом шаге требуется подтвердить сделанный ранее выбор. Перечень установленных действий вы можете посмотреть в списках *Daily Plan Summary* (для ежедневного выполнения) и *Weekly Plan Summary* (для еженедельного выполнения).

После нажатия кнопки *Finish* появится окно диалога, в котором вы можете распечатать созданный план.

После принятия созданного плана SQL Server выполняет различные установки, которые относятся к разным аспектам управления сервера баз данных. Поэтому не так просто разобраться, относятся ли запланированные действия SQL Server к составленному плану или установлены самостоятельно. В связи с этим рекомендуется напечатать подготовленный план.

В следующем окне диалога вы можете выполнить выбранные действия немедленно, нажав кнопку *Run Plan Now*. Если вы щелкните на кнопке *Done*, действия будут выполняться в соответствии с планом. Немедленное исполнение плана не отменяет запланированных действий.

### 2.4 Управление доступом к данным

Управление доступом к данным является важной составной частью обеспечения высокой надёжности и сохранности данных. Как и упоминалось ранее, SQL Server 2000 имеет тесную интеграцию с семейством платформ Windows NT – систему администрирования работы пользователей.

### 

### 2.4.1 Система защиты SQL Server

Система защиты данных SQL Server предусматривает использование атрибутов перечисленных в ниже приведенной таблице.

Для доступа к данным SQL Server пользователь должен преодолеть несколько уровней защиты.

Таблица №9 **-** Некоторые термины, применяемые при задания доступа

|  |  |
| --- | --- |
| *Атрибут* | *Описание* |
| **Login** | Идентификатор пользователя SQL Server – имя пользователя для регистрации. Это имя должно быть уникально. |
| **User** | Имя пользователя БД, привязанное к используемому для входа в SQL Server имени (login) и позволяющее осуществить доступ к данным БД. |
| **Alias** | Имя пользователя БД (псевдоним), являющееся общим для нескольких идентификаторов пользователей SQL Server (login). Общие имена используются для предоставления нескольким пользователям одинаковых разрешений на действия с БД. |
| **Group** | Набор имён пользователей БД. Включение пользователя в группу автоматически предоставляет ему имеющиеся для данной группы разрешения. Это существенно облегчает администрирование большого числа пользователей. |

Об остальных уровнях защиты можно найти в приложении.

### 2.4.2 Планирование доступа к данным

Определение прав доступа на уровне БД является следующим после SQL Server этапом на пути к данным. Этот уровень позволяет установить связь между именами пользователей для регистрации и именами пользователей БД, которые определяются на уровне БД.

### 2.4.3 Группы пользователей БД

Начните с составления списка задач, которые должны выполняться с БД SQL Server. Выделите задачи, связанные с обновлением или добавлением данных, и задачи, связанные только с просмотром данных. Определите пользователей, которые будут выполнять эти задачи.

Пользователей, которые будут выполнять общие действия, объедините в группы. Теперь каждой задаче будет соответствовать группа пользователей, что позволит существенно ускорить планирование доступа к данным SQL Server. В некоторых случаях на перечень групп может влиять организационная структура фирмы.

Каждой группе присвойте имя, которое поможет понять её назначение.

### 2.4.4 Обеспечение доступа к данным

Создайте группы для каждой БД. Для этого в SQL EM в иерархическом списке *Server Manager* вызовите в папке *Groups/Users* контекстное меню и выполните команду *New Group*. На экране появится окно диалога, в котором наберите название новой группы.

Для создания или удаления группы необходимо иметь права **sa** или **dbo**. Естественно, что для каждой БД может быть создана группа только с уникальным именем.

Существующих пользователей можно добавить в группу, выбрав их имена в списке *Users* и переместив их в список *Users in Group*. Для добавления группы в БД нажмите кнопку *Add*.

Для работы с группами из программы можно использовать следующие хранимые процедуры.

Таблица №10 - Некоторые хранимые процедуры для работы с группами

|  |  |
| --- | --- |
| *Процедура и синтаксис* | *Назначение* |
| sp\_addgroup <Имя группы> | Добавление группы |
| sp\_dropgroup <Имя группы> | Удаление группы |
| sp\_changegroup <Имя группы> <Имя пользователя> | Перемещение пользователя в другую группу |
| sp\_helpgroup [<Имя группы>] | Просмотр информации о всех созданных группах или о конкретной группе |

Установите для каждого пользователя имя регистрации. Помните, что имена регистрации (login) устанавливается на уровне сервера. Для этого используется окно диалога SQL *Server Logins Properties*, которое вызывается кнопкой на панели инструментов SQL EM или в папке *Security | Logins*.

Пользователи определяются на уровне БД. Для добавления в БД нового пользователя в контекстном меню соответствующей папки БД, выделите объект *Users*, выберите команду *New Database User*. На экране появится окно диалога *Users Properties*.

В поле *User Name* наберите имя пользователя. Имя регистрации вы можете выбрать *Login*. В списке *Group* выберите группу, к которой будет принадлежать пользователь.

Весьма важным моментом является определение для каждого имени регистрации базы данных по умолчанию. В окне SQL *Server Logins Properties* в области *Database Access* в столбце *Default* щелкните мышью в строке, отвечающей той БД, соединение с которой будет устанавливаться автоматически после регистрации пользователя. Очевидно, что эта БД должна иметь у себя разрешение на доступ соответствующего пользователя. В качестве БД по умолчанию лучше назначить базу данных, к которой пользователь обращается чаще всего.

SQL Server имеет два специальных имени пользователя:

*Администратор БД* (**dbo**) является владельцем БД. Это имя установлено в системной БД ***model*** и, таким образом, автоматически определяется для каждой создаваемой БД. Этот пользователь не может быть удален из БД и всегда имеет идентификатор пользователя, равный 1.

*Пользователь*-*гость* (**guest**) получает право, для которого достаточен доступ к SQL Server. По умолчанию такой пользователь не присутствует в БД при её создании. Не рекомендуется добавлять этого пользователя в БД ***model***.

Каждый пользователь по умолчанию входит в группу ***Public***. Если БД имеет пользователя **Guest** (гость), то любой пользователь, получивший доступ к серверу, получит доступ к данной БД и будет пользоваться правами, предоставленными этой группе.

Для того чтобы избежать утомительной процедуры добавления пользователей к каждой БД, SQL Server может «копировать» пользователей. Вы можете определить дополнительного пользователя, который будет иметь такие же права, используя общие имена (*aliases*). Они устанавливаются в окне диалога *Users Properties*. Теперь вам будет доступна область *Aliases*, в которой вы можете задать общие имена.

Определение общих имен выполняется с помощью связей, которые устанавливаются между таблицами ***sysusers***, ***syslogins*** и ***sysalternates*** БД ***master***.

Эти же таблицы используются для проверки прав пользователя при доступе к БД. Если пользователь получил доступ к серверу на основании данных таблицы ***syslogins***, проверяется наличие записи о пользователе в таблице ***sysusers***. Если такой записи нет, ищется отношение, установленное в таблице ***sysalternates***, к другому пользователю. Если оно не найдено, проверяется наличие пользователя с именем guest в таблице ***sysusers***. Любой положительный ответ разрешает пользователю доступ к данным. Отрицательный окончательный ответ запрещает доступ.

При удалении имени пользователя следует учесть следующее:

***Администратор БД*** может удалить любого пользователя любой БД.

***Владелец БД*** может удалить пользователя только своей БД.

***Пользователь*** не может быть удален из БД, если он является владельцем какого-либо объекта в БД. Для проверки владения используйте следующий запрос:

SELECT name, type FROM sysobjects

WHERE uid = user\_id (<Имя пользователя>)

Изменить владельца какого-либо объекта в БД невозможно. Для этого надо удалить объект и создать его заново.

Для удаления пользователя из БД в иерархическом списке Server Manager SQL EM откройте папку *Groups/Users* для требуемой БД, выберите имя пользователя и нажмите клавишу *Del*.

При удалении имени регистрации следует учесть следующее:

* Только администратор БД (**sa**) может удалить имя регистрации.
* Нельзя удалить имя регистрации пользователя, который является владельцем объекта в текущей БД.

Для удаления имени регистрации в иерархическом списке SQL EM откройте папку *Security*, выберите пункт *Logins*. Выберите требуемое имя и нажмите клавишу *Del*. При этом SQL EM автоматически удалит пользователей, которые связаны с этим именем, если только они не являются владельцами каких-либо объектов.

Для администрирования доступа пользователей из программы можно использовать хранимые процедуры, список которых приведен в таблице №11.

Таблица №11 - Хранимые процедуры для администрирования доступа пользователей

|  |  |
| --- | --- |
| *Процедура и синтаксис* | *Назначение* |
| sp\_addlogin <имя регистрации> [,<пароль> [,<БД по умолчанию> [,<язык по умолчанию>]]] | Добавляет имя регистрации |
| sp\_adduser <имя регистрации>  [,<имя пользователя> [,<имя группы>]] | Добавляет пользователя в БД |
| sp\_addalias <имя регистрации>,  <имя пользователя> | Добавляет общее имя пользователя (в виде записи в таблице ***sysalternates***) |
| sp\_droplogin <имя регистрации> | Удаляет имя регистрации |
| sp\_dropuser <имя пользователя> | Удаляет пользователя |
| sp\_defaultdb <имя регистрации>, <имя БД> | Изменяет БД по умолчанию для указанного имени регистрации |
| sp\_password <старый пароль>, <новый пароль> [,<имя регистрации>] | Изменяет пароль пользователя |
| sp\_helpuser <имя пользователя> | Позволяет получить информацию о пользователе БД |

### 2.5 Приложение

### 2.5.1 Уровень операционной системы клиента

Уровень операционной системы клиента определяет права пользователя на:

* запуск приложения для работы с SQL Server;
* доступ к ресурсам локального компьютера;
* доступ к сетевым ресурсам;
* использование клиентских утилит SQL Server.

Надежность защиты на этом уровне определяется особенностями ОС. Очевидно, использование на клиентском компьютере Windows 95 не может полностью перекрыть дорогу пользователю, хотя и имеет другие неоспоримые преимущества. Windows NT Workstation обеспечит на этом уровне защиту, не уступающую серверу. Поэтому минимально допустимой клиентской платформой назовём именно эту систему.

### 2.5.2 Уровень SQL Server

Уровень SQL Server обеспечивает защиту данных на основе регистрации пользователя, которая включает указание личного имени пользователя и пароля. SQL Server может использовать стандартный доступ. В этом случае имя пользователя и пароль назначаются только для SQL Server и не имеют отношения к имени пользователя и паролю, используемых Windows NT. Интегрированный доступ позволяет применять для доступа к SQL Server имя пользователя и пароль, используемые в Windows NT. Это позволяет более эффективно организовать систему доступа, т.к. на основе одной учетной записи пользователя можно организовать доступ к различным серверам семейства BackOffice.

### 2.5.3 Уровень базы данных SQL Server

Доступ пользователя к той или иной БД может быть обеспечен одним из следующих способов:

* добавлением пользователя БД;
* распространением на пользователя прав другого пользователя, имеющего доступ к БД (общие имена – *aliases*);
* создание пользователя с именем **Guest** (гость);
* назначение пользователя владельцем БД.

После получения доступа к БД пользователь может просматривать описания объектов, т.к. они хранятся в системных таблицах. Для доступа к каждому объекту БД пользователь должен иметь соответствующие права.

### 2.5.4 Уровень объектов SQL Server

При создании объектов БД SQL Server автоматически предоставляет все права владельца тому пользователю, который создал данный объект. Владелец объекта может выполнять с ним любые действия. По умолчанию только владелец объекта может производить выборку данных из таблицы.

Несмотря на то, что доступ к объектам БД является последним уровнем защиты данных, организовать защиту на этом рубеже можно достаточно гибко за счет выдачи разрешений. Эти разрешения, например, могут позволять пользователю выбирать данные из представления, но не из таблиц, на которых основано представление. Таким образом, администратор может предоставить пользователю доступ к данным, содержащимся в таблицах, но не к самим таблицам.

### 2.5.5 Стандартный режим безопасности

Стандартный доступ к данным SQL Server предусматривает использование идентификатора пользователя (login) и пароля (password). По умолчанию SQL Server имеет встроенные имена для входа в систему, которые перечислены в ниже приведенной таблице№12.

Таблица №12 - Встроенные имена SQL Server

|  |  |
| --- | --- |
| *Login* | *Описание* |
| **sa** | Имя администратора SQL Server. Он является владельцем сервера, системных БД и пользовательских БД, независимо от создателя этих БД. |
| **probe** | Используется некоторыми приложениями (например, SQL Performance Monitor) для соединения с SQL Server. Не применяйте это имя для пользователей. |
| **repl\_publisher**  **repl\_subscriber** | Используются для управления соединением между серверами при репликации данных. Не рекомендуется модифицировать эти имена и предоставлять их пользователям. |

Информация о пользователях хранится в таблице ***syslogins*** БД ***master***. Вы можете получить подробные сведения о пользователях, выполнив запрос к этой таблице. Однако наиболее важная информация записана в поле *suid* (*suid* – это некое целое число), т.к. именно этот идентификатор выводится сервером в сообщениях об ошибках, связанных с доступом к данным. Когда пользователь пытается войти в SQL Server, производится поиск указанного имени пользователя в поле *name*. Если имя найдено и пароль введен правильно, ключевой идентификатор пользователя запоминается для вновь созданного соединения.

Если в окне Query SQL EM установить текущую БД ***master*** и набирать код:

SELECT suid, name FROM syslogins,

то на вкладке Results вы увидите список идентификаторов и имен пользователей.

Для получения имени пользователя по его идентификатору или идентификатора по имени можно использовать системные функции SUSER\_NAME (<Идентификатор>) или SUSERID (<Имя>). Без аргумента эти функции вернут соответственно имя или идентификатор для текущего пользователя.

## 3 Задания для самостоятельной работы

1. Создать устройство.
2. Создать базу данных на созданном устройстве.
3. Измените параметры устройства, размер БД.
4. Создайте таблицу. В этой таблице сформируйте поля, в соответствии с заданием выданным преподавателем.
5. Автоматизируйте процесс администрирования БД.
6. Установите стандартный (интегрированный, смешанный) режим безопасности.
7. Создайте группы для каждой БД. Установите для каждого пользователя имя регистрации.

## 4 Контрольные вопросы

1. Как создать (удалить) БД?
2. Какие действия выполняет SQL Server при создании БД?
3. Какими соображениями необходимо руководствоваться, чтобы достаточно точно определить пространство, требуемое для БД?
4. Что такое журнал транзакций?
5. Для чего используется процедура sp\_changedbowner?
6. Как можно получить информацию о БД, о журнале транзакций?
7. Какие параметры можно установить для БД?
8. Какие стандартные процедуры используются для работы с сегментами?
9. Как создать таблицу?
10. Как создать и изменить индекс в SQL EM?
11. При установке какого параметра включается монопольный режим доступа?
12. Какие характеристики создаваемого индекса можно определить в области Index Attributes?
13. Перечислите, какие шаги необходимо выполнить для автоматизации процесса администрирования БД?
14. Назовите уровни защиты SQL Server?
15. Перечислите режимы безопасности?
16. Что предусматривает стандартный режим безопасности?
17. Какие параметры можно установить для SQL Server?
18. Какие действия необходимо выполнить, чтобы создать группу для каждой БД?
19. Назовите хранимые процедуры, использующиеся для администрирования групп?
20. Как установить для каждого пользователя имя регистрации?
21. Что необходимо учитывать при удалении имени?
22. Назовите хранимые процедуры, использующиеся для администрирования доступа пользователей?

# Лабораторная работа №11 - Администрирование SQL Server. Предоставление разрешений. Архивирование данных.

## 1 Цель работы

Освоить систему предоставления разрешений, MS SQL Server, а также получить навыки архивирования данных.

## 2 Основные теоретические сведения

### 2.1 Предоставление разрешений

### 2.1.1 Типы разрешений

SQL Server использует разрешения для управления доступом в пределах БД. Используя разрешения, вы можете определять доступ пользователя к следующим объектам БД:

* таблицам;
* значениям по умолчанию;
* правилам;
* индексам;
* представлениям;
* триггерам;
* хранимым процедурам.

Не имея соответствующих разрешений, пользователь может получить доступ к SQL Server, но не к работе с данными.

Каждая БД имеет независимую систему разрешений. Таким образом, пользователь, имеющий разрешения для работы с одной БД, не сможет выполнять тех же действий с другой БД.

Разрешение выдается (***grant***) или отнимается (***revoke***). В дальнейшем, употребляя слово «разрешение», мы будем подразумевать возможность выполнения и того и другого действия.

Разрешения могут предоставляться как отдельным пользователям, так и группам. Если вы предоставляете разрешение для какого-то пользователя, то физически это подразумевает добавление записи в системную таблицу ***sysprotects***, которую имеет каждая БД. Таким образом, разрешение не может быть предоставлено просто по имени регистрации.

Разрешения, которые получает пользователь, зависят от объектов и команд, с которыми данный пользователь должен работать.

### 2.1.2 Объектные разрешения

Разрешения, которые относятся к объектам (*объектные разрешения*), позволяют управлять доступом к: таблицам; представлениям; хранимым процедурам.

При этом для каждого объекта могут предоставляться различные типы разрешений (таблица №13).

Таблица №13 - Типы разрешений

|  |  |
| --- | --- |
| *Команда* | *Объекты* |
| SELECT | Таблица (включая уровень столбцов), предоставление |
| UPDATE | Таблица (включая уровень столбцов), представление |
| INSERT | Таблица, представление |
| DELETE | Таблица, представление |
| REFERENCE | Таблица |
| EXECUTE | Хранимая процедура |

Типы разрешений SELECT, UPDATE, INSERT и DELETE используются достаточно часто. Они определяют возможность пользователя выполнять команды, в опции FROM которых перечислены такие объекты, как таблицы или представления, к которым пользователь должен иметь доступ. Тип разрешения REFERENCE управляет доступом пользователя к поддержке целостности данных. На практике это необходимо только в том редком случае, когда требуется установить отношение между двумя объектами, имеющими разных владельцев. Тип разрешения EXECUTE позволяет пользователю выполнять хранимые процедуры. Обратите внимание, что таким образом пользователь может выполнять какие-либо действия, определяемые хранимыми процедурами, но не имеет доступа непосредственно к данным таблиц или представлений.

Разрешать или запрещать доступ к объектам БД может владелец объекта. При этом в разрешениях нуждаются только другие пользователи, но не сам владелец.

Для системного администратора разрешения для доступа к объектам не проверяются. Следовательно, ему не требуется давать разрешение на работу с тем или иным объектом.

### 2.1.3 Командные разрешения

Разрешения, которые относятся к командам (командные разрешения), позволяют управлять доступом к выполнению различных действий с БД. Эти действия не привязаны к конкретным объектам и включают возможность выполнения следующих команд:

CREATE DATABASE

CREATE DEFAULT

CREATE PROCEDURE

CREATE RULE

CREATE TABLE

CREATE VIEW

DUMP DATABASE

DUMP TRANSACTION

Командные разрешения могут предоставляться системным администратором или владельцем БД. Как правило, командные разрешения требуются пользователям только на этапе разработки. Не предоставляйте пользователям таких разрешений на этапе эксплуатации приложения.

### 2.1.4 Иерархия разрешений

При обсуждении различных типов разрешений несложно было заметить, что с точки зрения системы доступа пользователей к объектам БД SQL Server насчитывает четыре типа пользователей:

* системный администратор (**sa**);
* владелец БД (**dbo**);
* владелец объекта (**dboo**);
* пользователь БД.

Системный администратор находится на вершине иерархии, т.к. имеет все разрешения для доступа к объектам и выполнения команд для любой БД. Пользователь, не являющийся владельцем объекта, по умолчанию не имеет никаких разрешений. Даже при получении доступа к SQL Server он не сможет выполнить никаких действий с данными. Однако любой пользователь может считывать данные из системных таблиц, т.к. для этого не требуется специальных разрешений.

### 2.1.5 Права системного администратора

Системный администратор находится вне системы разрешений, это позволяет ему управлять обработкой данных и выполнять восстановление поврежденных данных независимо от их принадлежности. Системный администратор выполняет общие для СУБД задачи, независимые от пользовательского приложения для обработки данных. Следующие задачи лучше оставить для выполнения системному администратору и не предоставлять разрешений на них другим пользователям:

* остановка процесса (KILL);
* выключение SQL Server (SHUTDOWN);
* переконфигурация SQL Server (RECONFIGURE);
* выполнение операций, связанных с проверкой целостности системы (DBCC);
* добавление и удаление хранимых процедур.

Только системный администратор может создавать БД (CREATE DATABASE) и давать на это разрешение другим пользователям. Лучше зарезервировать контроль над выполнением этой операции за системным администратором, т.к. именно он определяет размер и местоположение каждой БД. Задача модернизации БД (ALTER DATABASE) по умолчанию закрепляется за владельцем БД, но только в том случае, если он имеет разрешение на создание БД.

### 2.1.6 Права владельца базы данных

Владелец БД имеет все права на выполнение любых действий с объектами своей БД. В ниже приведенной таблице 14, перечислены разрешения, которые автоматически предоставляются владельцу БД и не могут быть переданы другому пользователю.

Таблица №14 - Разрешения, автоматически предоставляемые владельцу БД

|  |  |
| --- | --- |
| *Задача* | *Команды* |
| Создание объекта | CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TABLE, CREATE VIEW |
| Создание архива БД или журнала транзакций | DUMP DATABASE, DUMP TRANSACTION |
| Использование объектов | SELECT, UPDATE, INSERT, DELETE |

Разрешения наследуются при создании объекта. Например, если пользователь получил разрешение на создание представления, он, создав представление, становится его владельцем и может предоставлять или отменять разрешения на доступ к нему другим пользователям БД.

В таблице №15 перечислены разрешения, которые владелец БД не может предоставить другому пользователю.

Таблица №15 - Разрешения, которые владелец не может предоставить другому пользователю

|  |  |
| --- | --- |
| *Задача* | *Команды* |
| Восстановление данных БД и журнала транзакций из архива | LOAD DATABASE, LOAD TRANSACTION |
| Удаление БД | DROP DATABASE |
| Выполнение операций, связанных с проверкой целостности данных | DBCC |
| Выполнение роли другого пользователя | SETUSER |
| Установка контрольных точек | CHECKPOINT |
| Предоставление и отмена командных разрешений | GRANT, REVOKE |
| Модификация БД | ALTER DATABASE |

### 2.1.7 Права владельцев объектов БД

Пользователь, который создал объект БД, является его владельцем и автоматически получает все разрешения по работе с ним. Другие пользователи, включая владельца БД, не имеют никаких разрешений для работы с этим объектом, пока его владелец не предоставит их явно. Естественно, это не касается системного администратора.

Объектные разрешения могут быть получены для выполнения команд: SELECT, UPDATE, INSERT, DELETE, REFERENCE и EXECUTE. Командные разрешения, включая GRANT и REVOKE, не могут быть переданы другому пользователю.

В таблице 16 перечислены разрешения, которые по молчанию получает владелец таблицы.

Таблица №16 - Разрешения, получаемые владельцем таблицы по умолчанию

|  |  |
| --- | --- |
| *Задача* | *Команды* |
| **1** | **2** |
| Модернизация таблицы путем добавления столбца | Alter Table |

Продолжение таблицы №16

|  |  |
| --- | --- |
| **1** | **2** |
| Создание или удаление индекса | Create Index, Drop Index |
| Создание или удаление триггера | Create Trigger, Drop Trigger |
| Удаление таблицы | Drop Table |
| Удаление данных из таблицы | Truncate Table |
| Обновление данных об индексах | Update Statistics |

Владельцы других объектов по умолчанию получают разрешение на удаление своих объектов (команды DROP VIEW, DROP PROCEDURE, DROP RULE, DROP DEFAULT).

### 2.1.8 Установка разрешений

Доступ к таблицам, представлениям и хранимым процедурам должен быть установлен на этапе проектирования системы обработки данных. Параллельно с выяснением необходимости создания того или иного объекта необходимо документировать, какие пользователи должны получить доступ к этому объекту.

**Установка разрешений для групп пользователей.**

Наиболее простой путь предоставления разрешений заключается в предоставлении или запрещении тех или иных действий группам пользователей. В этом случае включение пользователя в группу автоматически предоставляет ему возможность выполнения действий, разрешенных для данной группы.

При этом весьма важно учитывать, что любой пользователь БД, даже если вы явно указываете его принадлежность какой-то группе, по умолчанию входит в группу *public*. Пользователь **guest** будет, естественно, также входить в группу *public*. Таким образом, любой пользователь получивший доступ к БД, получит разрешения, предоставленные группе *public*. Можно использовать это для предоставления определенных разрешений любому пользователю БД. При этом особые требования к доступу могут быть реализованы путем создания других групп пользователей.

Рассмотрим простой пример. Если в вашу БД включено 50 таблиц и у вас создано 10 групп пользователей, то для того, чтобы предоставить им возможность выборки данных из всех таблиц, вы должны предоставить разрешение SELECT каждой из десяти групп для каждой таблицы. Гораздо проще предоставить такое разрешение группе *public*, т.к. это потребует только одного разрешения для каждой таблицы. Объем работы будет сокращен в десять раз. Дополнительные разрешения, например для удаления или добавления данных, могут быть предоставлены какой-то конкретной группе.

Проблема с пользователем **guest** может быть разрешена путем предоставления разрешений группе *public*, а затем запрещения выполнения каких-либо действий пользователю **guest**. Таким образом, любой пользователь, получивший доступ к серверу, будет ограничен в доступе к данным на уровне конкретной БД.

При предоставлении разрешений рекомендуется использовать следующую последовательность:

1. Предоставьте соответствующие разрешения группе *public*. Эти разрешения будут иметь приоритет над разрешениями, ранее предоставленными пользователям, входящим в группу *public* или в другие группы (то есть фактически всем пользователям).
2. Предоставьте соответствующие разрешения другим группам, которые находятся в иерархии ниже системной группы public. Разрешения, предоставленные группе, будут иметь приоритет над разрешениями пользователей этой группы.
3. Предоставьте соответствующие разрешения отдельным пользователям. Эти разрешения будут действовать до тех пор, пока вы не измените разрешения для группы public или группы, в которую входит данный пользователь.

Если не следовать данной последовательности действий, то вы можете столкнуться с неожиданными несовпадениями между вашими планами и действительностью, которые чаще всего выражаются в следующем:

запрещение разрешения, установленное для пользователя, вдруг не действует, т.к. после этого вы предоставили такое разрешение для группы, в которую входит данный пользователь;

предоставление разрешения пользователю не позволяет ему выполнить требуемое действие, т.к. соответствующее разрешение было запрещено для группы, в которую входит данный пользователь.

SQL Server для проверки разрешений использует таблицу. При предоставлении или запрещении разрешения SQL Server действует следующим образом:

* В таблице ***sysprotects*** ищется запись, в которой были бы такие же значения в столбцах ***id*** (код объекта), ***uid*** (код пользователя) и ***action*** (код действия) и противоположное значение в столбце ***protecttype*** (код предоставления разрешения, если оно запрещается, и код запрещения, если разрешение предоставляется). Если такая запись не найдена, в таблицу добавляется новая запись.
* Если код в столбце ***uid*** представляет собой код группы пользователей, то SQL Server ищет записи с такими же значениями в столбцах ***id***, ***action*** и ***uid*** для пользователей этой группы и удаляет записи, т.к. разрешения, предоставленные для группы пользователей, имеют приоритет над разрешениями, предоставленными пользователю.
* Если код в столбце ***uid*** представляет собой код группы public, то SQL Server ищет записи с такими же значениями в столбцах ***id***, ***action*** и ***uid*** для пользователей этой группы и удаляет эти записи. Найденные записи удаляются как для пользователей, так и для групп (кроме записи для группы public).

### 2.1.9 Установка разрешений для работы с БД

Для предоставления или запрещения разрешений на работу с БД запустите SQL EM. В окне *Server* EM выберите требуемую БД, в контекстном меню выберете команду *Properties*, а затем вкладку *Permissions*.

Выберите группу или пользователя, а затем соответствующий столбец с разрешением.

Возможные разрешения перечислены в ниже приведенной таблице 17.

Щелчком мыши вы можете предоставить соответствующее разрешение (появляется зеленая галочка) или запретить его (появляется перечеркнутый красный круг). Синяя галочка обозначает, что данное разрешение предоставлено и действует в настоящее время, а красная – что данное разрешение в настоящее время запрещено. Установленные разрешения начинают действовать после того, как вы нажмете кнопку «*ОК»*.

Таблица №17 - Возможные разрешения

|  |  |
| --- | --- |
| *Разрешение* | *Позволяет пользователю создать* |
| CREATE DB | Базу данных. Может быть предоставлено только системным администратором и только для пользователя, включенного в БД ***master*** |
| CREATE TABLE | Таблицу |
| CREATE VIEW | Представление |
| CREATE SP | Хранимую процедуру для БД |
| CREATE DEFAULT | Значение по умолчанию, которое SQL Server будет записывать в столбец, если ничего не введено пользователем |
| CREATE RULE | Правила ввода данных |

### 2.1.10 Определение объектных разрешений

Для предоставления или запрещения объектных разрешений запустите SQL EM и выполните следующие действия. В окне *Server* EM выберите требуемый сервер, выделите требуемый объект *Databases | NewDatabase | Tables*. В контекстном меню *All Tasks | Manage Permissions*.

В появившемся окне диалога, вы можете установить пункты, приведенные в таблице №18.

Таблица №18 - Допустимые пункты

|  |  |
| --- | --- |
| Разрешение | Пользователь или группа может |
| SELECT | Считывать данные из таблицы, представления или столбца |
| INSERT | Добавлять новые данные в таблицу или представление |
| UPDATE | Изменять данные в таблице, представлении или столбце |
| DELETE | Удалять данные из таблицы или столбца |
| EXECUTE | Выполнять хранимую процедуру |
| DRI | Ссылаться на таблицу без наличия разрешения SELECT для этой таблицы |

Для определения разрешений SELECT или UPDATE для какого-либо столбца (таблицы или представления) установите флажок *List All Users*. Для предоставления всех разрешений всем пользователям или группам для всех объектов. Для принятия выполненных установок щелкните на кнопке *«Применить»*.

При предоставлении разрешений следует учитывать взаимозависимости между объектами БД. Такие зависимые объекты БД, как представления, хранимые процедуры и триггеры дают администраторам следующие дополнительные возможности обеспечения безопасности данных:

1. представления могут закрыть доступ к некоторым записям таблиц;
2. хранимые процедуры могут запретить (или выполнить) дополнительную проверку всех изменяемых данных;
3. триггеры позволяют обновлять данные в связанных записях.

Если представление и таблица, на которой оно основано, имеют разных владельцев, то пользователь, который хочет считать данные представления, должен получить (от владельца таблицы) разрешение и на считывание данных из таблицы. Пользователь получает доступ к таблице, и дополнительный уровень безопасности использоваться не может.

Не рекомендуется без особых причин допускать такую ситуацию. Следует придерживаться следующих правил:

1. владелец БД должен самостоятельно создавать все ее объекты и, следовательно, быть их настоящим владельцем;
2. при работе с большой БД, когда при создании её объектов не обойтись без нескольких специалистов, установите общее имя владельца БД для имен регистрации пользователей, создающих объекты в этой БД.

### 2.2 Архивирование данных

### 2.2.1 Методы предотвращения потерь данных

Архивирование данных является одним из важнейших мероприятий, направленных на обеспечение сохранности данных. Архивирование позволяет быстро восстановить данные после аппаратных или программных сбоев. Оно должно выполняться совместно с другими мероприятиями, направленными на поддержание сохранности данных.

Для предотвращения потерь данных могут использоваться возможности как Windows NT, так и SQL Server.

### 2.2.2 Архивные устройства

При архивировании вы должны определить место, где будет создаваться архивная копия данных. Ассоциировать логическое имя с физическим устройством, на котором будет создаваться архивная копия, можно путем указания архивного устройства (backup device).

Архивное устройство может использоваться исключительно при выполнении команд DUMP или LOAD и, как правило, создается либо на жестком диске, либо на магнитной ленте.

Магнитная лента может обеспечить повышенную безопасность хранения данных (т.к. она изымается из компьютера в нерабочее время и помещается в более неприступное место типа банковского сейфа). В этом случае на сохранность архивной копии не повлияет даже такая всесокрушающая катастрофа, как пожар. Технология производства стримеров и магнитных лент стремительно совершенствуется. Повышается скорость записи и емкость кассет. В общем случае использование магнитных лент предпочтительнее для очень больших объемов данных.

Архивное устройство на жестком диске представляет собой файл. Архивирование на жесткий диск выполняется быстрее, чем на магнитную ленту. Его использование позволит сократить время создания архивной копии. Естественно, данные БД и архивная копия должны, по возможности, располагаться не только на физически разных дисках, но и на дисках, управляемых разными контроллерами.

Помимо этих двух типов носителей, SQL Server позволяет создавать архивную копию на таких съемных носителях данных, как дискета или магнитооптический диск. С ростом скорости записи и объема таких дисков их использование, по всей видимости, будет постоянно расширяться.

В качестве архивного устройства может использоваться и внешний носитель данных. В этом случае необходимо задействовать протокол Named Pipe для правильного определения имени устройства, указываемого в соответствии со стандартом Universal Naming Convention Windows NT.

Независимо от типа архивного устройства SQL Server добавляет новую копию к существующей. Таким образом, если вы не будете предпринимать каких-либо действий, размер архивной копии будет постоянно возрастать.

Создание архивной копии может выполняться как системным администратором, так и владельцем БД. Эта привилегия может быть предоставлена и другому пользователю БД, но не стоит оставлять ее без контроля.

Для создания архивного устройства запустите SQL EM, в окне Server EM раскройте папку Management, щелкните правой кнопкой мыши на пункте Backup и в появившемся контекстном меню выберите команду New Backup Device. Появится окно диалога, в котором необходимо указать имя и местоположение архивного устройства. Для создания устройства нажмите кнопку Ok.

При присвоении имени архивному устройству явно обозначьте его тип. Например, для магнитной ленты можно использовать имена Tape 1, Tape 2 и т.д. Для жесткого диска в имени устройства используйте имя БД и тип архивной копии. Например, test\_backup, test\_tran и т.д.

### 2.2.3 Архивирование данных

Выберите нужную вам БД и в контекстном меню команду *All Tasks | Backup Database*. Появится окно диалога.

В области Backup To выберите архивное устройство, на котором хотите создать копию. SQL Server позволяет записать копию как на одно, так и на несколько устройств. При необходимости, не выходя из этого окна диалога, можно создать новое архивное устройство, нажав кнопку New.

Создание копии на нескольких устройствах (striped backup) позволяет использовать до 32 устройств, на которые будут записаны примерно равные порции данных. Это существенно ускоряет процесс копирования за счёт распараллеливания записи. Для выбора из списка нескольких устройств удерживайте нажатой клавишу Ctrl.

Для создания архивной копии немедленно нажмите кнопку Ok. SQL Server начнет копирование, и вы не сможете продолжить работу с SQL EM, пока копия не будет создана.

Для создания архивной копии в фоновом режиме нажмите кнопку Schedule. На экране появится окно диалога *Backup Volume Label*, в котором вы можете указать номер создаваемой архивной копии, например «SS0001», после чего появитсяокно диалога *Schedule Backup*, где вы определите необходимые параметры создания архива:

1. **Immediately** – создание архива может начаться сразу после нажатия кнопки *«ОК»*.
2. **One Time** – создание архива начнется в указанный в поле *On Date* день и указанный в поле *At Time* час.
3. **Recurring** – создание архива будет выполняться периодически. Для изменения времени копирования в архив нажмите кнопку *«Change»*. Появится окно диалога *Task Schedule*, в котором можно задать самые разнообразные условия периодического архивирования данных.

Или можно также непосредственно в поле SQL *Command* отредактировать команду DUMP, которая будет выполняться для создания архивной копии.

### 2.2.4 Журнал транзакций

Для понимания механизма работы пошагового архивирования необходимо знать, как SQL Server записывает транзакции.

В журнал транзакций записываются все изменения, выполняемые в таблицах командами INSERT, UPDATE или DELETE. Записываются также действия, связанные с созданием новых объектов, определением привилегий и т.п., то есть изменяющие содержание системных таблиц. Запись изменений ведется путем добавления данных в таблицу ***syslogs***.

Журнал транзакций SQL Server обладает следующими отличительными свойствами:

1. Ведет учет изменений, связанных со всеми объектами данной БД.
2. Является общим для всех пользователей БД.
3. Изменения происходят в дисковом буфере SQL Server и записываются на диск только в момент завершения.
4. Изменения записываются сначала в журнал транзакций, а только затем в БД (write-ahead log).

Журнал транзакций не предназначен для чтения или выполнения каких-либо изменений с записанными в нем данными. При изменении данных SQL Server выполняет следующие действия:

1. Добавляет запись в журнал транзакций (сначала – в буфер) о начале транзакции.
2. Записывает (сначала – в буфер) все изменения в журнал транзакций.
3. Формирует изменения в данных БД (сначала – в буфере).
4. Добавляет запись (сначала – в буфер) в журнал транзакций о завершении транзакции.
5. Записывает все измененные страницы с данными на диск.

Если подается команда не завершения транзакции, а ее отмены, то запись на диск не выполняется. Для подтверждения записи данных из буфера SQL Server на диск используется механизм контрольных точек (checkpoints). Они бывают двух типов:

1. контрольные точки, которые автоматически выполняются SQL Server;
2. контрольные точки, выполнение которых инициируется владельцем БД или системным администратором с помощью команды CHECKPOINT.

Автоматический запуск процесса выполнения контрольных точек зависит от заполнения дискового буфера и установленного интервала восстановления SQL Server (recovery interval). Первая контрольная точка свидетельствует о записи изменений из буфера в журнал транзакций, и только вторая – записи измененных данных в БД.

Контроль за журналом транзакций – является одной из важнейших функций системного администратора. При исчерпании свободного пространства, отведенного для журнала транзакций, пользователи не смогут изменять данные, содержащиеся в БД. Для облегчения управления журналом транзакций следуйте следующим рекомендациям:

1. Размещайте журнал транзакций на отдельном устройстве.
2. Наблюдайте за журналом транзакций, используя графические возможности SQL EM или системные процедуры sp\_spaceused и sp\_helpdb.
3. Регулярно выполняйте архивное копирование журнала транзакций.

### 2.2.5 Планирование архивирования

Неоднократные упоминания о важности тщательного планирования процесса архивирования при разработке пользовательского приложения, заставляют нас более подробно остановиться на моментах, которые должны быть обязательно учтены.

### 2.2.6 Частота архивирования

Частота архивирования и тип используемого архива в общем случае зависят от двух факторов:

1. допустимый объем данных, который может быть потерян при сбое устройства хранения данных;
2. объем транзакций, выполняемый SQL Server.

Практика показала, что в большинстве случаев достаточно делать архивную копию БД еженедельно, а журнала транзакций – ежедневно. В то же время для систем с очень высокой интенсивностью обновления данных рекомендуется ежедневное копирование БД и ежечасное – журнала транзакций.

### 2.2.7 Динамическое или статическое архивирование

SQL Server позволяет выполнить *динамическое* архивирование данных (online), которое происходит параллельно с работой пользователей. При этом SQL Server многократно просматривает страницы данных, сверяясь с журналом транзакций, при необходимости записывает изменения на диск и создает наиболее «свежую» копию данных. Такой способ создания архивной копии неизбежно вызывает большую нагрузку на систему и приводит к снижению производительности. В то же время для круглосуточно работающих систем это единственно возможный вариант, и необходимо только подобрать для выполнения копии время, когда изменение данных в системе происходит наименее интенсивно.

Для систем, которые не эксплуатируются круглосуточно, лучше использовать *статический* вариант создания архивной копии (offline). В этом случае работа пользователей по обновлению данных пересекается по времени с архивированием.

### 2.2.8 Объем архивной копии

Объём архивной копии зависит от объема и количества архивируемых баз. Выше уже говорилось об этих аспектах и тех предпочтениях, которые требуется определить, если вы стремитесь получить максимально компактную копию данных. Остается только напомнить, что стремление минимизировать объем архивных данных понижает надежность восстановления и увеличивает его время.

Не забывайте включить в свои планы регулярное копирование системных баз данных, в которых отслеживается состояние системы. Например, БД ***master*** обновляется всякий раз при выполнении достаточно большого количества операций, перечень которых приведен в таблице №19.

Таблица №19 - Операции, при которых обновляется БД *master*

|  |  |
| --- | --- |
| *Операции* | *Команды и системные процедуры* |
| **1** | **2** |
| Создание, модернизация и удаление БД | CREATE DATABASE, ALTER DATABASE, DROP DATABASE |

Продолжение таблицы №19

|  |  |
| --- | --- |
| **1** | **2** |
| Создание, модернизация и удаление устройств | DISK INIT, DISK RESIZE, sp\_addumpdevice, sp\_dropdevice |
| Модернизация журнала транзакций | sp\_logdevice |
| Включение или отключение зеркальных дисков | DISK MIRROR, DISK UNMIRROR, DISK REMIRROR |
| Добавление или удаление сервера | sp\_addserver, sp\_dropserver |
| Добавление или удаление имени регистрации | sp\_dropremotelogin, sp\_addlogin, sp\_droplogin, sp\_addremotelogin |
| Изменение конфигурации | Sp\_configure |

**3 Задание для самостоятельной работы**

1. Установить разрешения для работы с БД: создание БД, создание таблицы, создание представления, создание правила ввода данных.
2. Установить объектные разрешения: считывание данных из таблицы, изменение данных в таблице, удаление данных из таблицы, выполнение хранимых процедур.
3. Выполнить зеркальную запись данных.
4. Создать архивную копию данной БД, которая будет действовать в течение 10 дней.

## 4 Контрольные вопросы

1. К каким объектам БД можно определить доступ пользователя, используя разрешения?
2. Какие типы разрешений вы знаете?
3. Какие типы разрешений могут предоставляться для каждого объекта?
4. Какой тип разрешения управляет доступом пользователя к поддержке целостности данных?
5. Какие разрешения называются командными?
6. Что собой представляет иерархия разрешений?
7. Какие права имеет системный администратор? Какие задачи лучше оставить для выполнения системному администратору?
8. Какие права имеет владелец БД? Какие разрешения владельцу БД предоставляются автоматически?
9. Какие разрешения владелец БД не может предоставить другому пользователю?
10. Какие права имеют владельцы объектов БД? Какие разрешения, по умолчанию, получает владелец таблицы?
11. Что использует SQL Server для проверки разрешений? Как действует SQL Server при предоставлении или запрещении разрешения?
12. Как установить разрешения для работы с БД? Перечислите возможные разрешения?
13. Какие действия необходимо выполнить для предоставления или запрещения объектных разрешений?
14. Какие дополнительные возможности дают зависимые объекты БД такие, как представления, хранимые процедуры и триггеры?
15. Для чего используется архивирование данных в SQL Server?
16. Какие конфигурации зеркальной записи данных поддерживает SQL Server?
17. В чем заключается архивирование? Какие типы архивирования вы знаете?
18. На чем, как правило, создается архивное устройство?
19. Какие действия необходимо выполнить для архивирования данных?
20. Какие установки можно сделать при архивировании?
21. Какими отличительными свойствами обладает журнал транзакций?
22. Какие действия выполняет SQL Server при изменении данных?
23. Что такое динамическое (статическое) архивирование?
24. При выполнении каких операций БД ***master*** обновляется?

# Лабораторная работа №12 - Восстановление данных. Обмен данными.

## 1 Цель и порядок работы

Овладеть основными методами восстановления данных и объектов SQL Server. Изучить методы копирования данных, овладеть навыками по созданию тиражных копий БД.

## 2 Основные теоретические сведения

### 2.1 Восстановление данных

В процессе функционирования SQL Server в силу различных причин может произойти сбой в работе сервера, что может привести к повреждению баз данных. Вы должны быть готовы воспользоваться имеющимися резервными копиями, чтобы восстановить работоспособность сервера в минимальный срок. Поскольку очень важно как можно быстрее предоставить пользователям возможность нормально работать с базами данных, следует иметь четкий и проверенный план действий по восстановлению.

SQL Server обладает возможностями автоматического восстановления при сбое системы. Каждый раз, когда происходит открытие базы данных, например, при запуске системы, SQL Server проверяет целостность баз данных, анализируя соответственные журналы транзакций. Для анализа используется участок журнала, содержащий сведения обо всех транзакциях, произошедших, начиная с момента выполнения последней операции CHECKPOINT, и заканчивается остановкой SQL Server. Система обнаруживает транзакции, которые не завершены, и производит их откат (roll back). Одновременно с этим происходит поиск завершенных транзакций, которые не были сохранены на диске, и осуществляется их фиксация (forward back). Это гарантирует, что в случае неудачного завершения работы сервера при его следующем запуске будут сохранены целостность и постоянство данных.

Вы можете инициировать процесс восстановления базы данных в любой момент, когда сочтете это нужным. Однако в SQL Server встроены внутренние механизмы, которые препятствуют случайному стиранию базы данных вследствие невнимательности. Процесс восстановления будет прерван в случае, если вы пытаетесь восстановить базу данных, которая уже имеется на сервере, и при этом база данных, которую вы пытаетесь восстановить, отличается от базы данных, содержащейся в резервной копии, либо имена этих двух баз данных не совпадают. С другой стороны, вы можете отключить вышеуказанные меры предосторожности.

Можно прибегнуть к восстановлению данных с резервных и с том случае, когда необходимо создать на другом сервере абсолютно идентичную копию существующей базы данных.

Восстановление данных подразумевает загрузку последней архивной копии данных и всех архивных копий журнала транзакций, выполненных с момента архивирования данных. Таким образом, данные восстанавливаются по состоянию на момент создания последней архивной копии журнала транзакций. Если повреждено устройство, на котором размещена БД, и это повлекло за собой потерю данных, данные могут быть восстановлены путем загрузки последних архивных копий данных и тех копий журнала транзакций, которые удалось восстановить.

При восстановлении данных и журнала транзакций необходимо учитывать следующее:

* + Когда БД восстанавливается на основе архивной копии, она должна быть закрыта, так как данные в этой БД перезаписываются на загружаемые данные.
  + Если данные восстанавливаются из-за повреждения физического устройства, в первую очередь необходимо стереть поврежденную БД. Это можно выполнить с помощью команды DROP DATABASE или системной процедуры **sp\_dbremove**. Только после этого можно пересоздать БД и восстановить данные из архивной копии.
  + Загрузка архивной копии журнала транзакций выполняется путем повторного выполнения записанных в нем изменений и отмены транзакций, которые не были завершены на момент создания архивной копии.
  + Архивные копии журнала транзакций должны загружаться в последовательности, в которой были созданы. SQL Server проверяет дату и время создания архивных копий БД и журнала транзакций для контроля правильной последовательности использования копий. После завершения загрузки всех архивных копий журнала транзакций содержание БД восстанавливается в том виде, в котором оно существовало на момент создания последней копии журнала транзакций (за исключением незавершенных транзакций). Такой процесс позволяет восстановить максимально возможный объем данных, потерянных из-за отказа системы.
  + Если БД создана с опцией FOR LOAD, а затем в нее загружена архивная копия данных, то эта БД получает статус «**dbo use only**» — «только для владельца БД». Таким образом, перед ее использованием другими пользователями вы должны отменить этот статус.

### 2.1.1 Основные методы восстановления данных

В SQL Server можно выделить два метода восстановления данных: автоматическое и вынужденное, выполняемое системным администратором при повреждении данных.

**Автоматическое восстановление** выполняется при загрузке SQL Server. Оно позволяет гарантировать, что все завершенные транзакции физически записаны " БД и тем самым риск наличия незавершенных транзакций при потере работоспособности системы минимален. Потеря работоспособности системы, как правило, возникает из-за перебоев в электроснабжении, и при этом подразумевается, что носитель данных не вышел из строя. Если действует режим автоматического восстановления, то при следующем запуске SQL Server автоматически восстанавливает данные для каждой БД. При этом выполняются следующие действия:

* + Отменяются транзакции, незавершенные на момент выключения SQL Server или потери работоспособности системы.
  + На основании журнала проверяются транзакции, которые были завершены в промежуток времени между выполнением последней контрольной точки и потерей работоспособности системы. При наличии таких транзакций их выполнение повторяется и записываются изменения, произведенные в данных. Такое действие получило название ***откат вперед*** (**rolling forward**).

Автоматическое восстановление начинается с БД ***master,*** затем следует***model*** после чего стирается содержание БД***tempdb*** и последовательно восстанавливаются БД***msdb, pubs* и *distribution*** (если сервер сконфигурирован для публикаций). После этого восстанавливаются БД пользователей. В процессе автоматического восстановления данных пользователи могут регистрироваться на сервере, но доступ к БД будет открыт только после полного завершения процесса.

Восстановление данных «по необходимости» происходит в случае выхода из строя носителя БД (жесткого диска или его контроллера). В случае неработоспособности носителя данных выполните следующие действия:

1. Найдите необходимую информацию о потерянных устройствах и БД.
2. Удалите испорченные устройства и БД, которые имеют отношение к потерянным данным и располагаются на неповрежденных устройствах.
3. Создайте заново потерянные устройства.
4. Пересоздайте БД с испорченными или потерянными данными. Наиболее простой путь для пересоздания БД — это запуск программы с командой CREATE DATABASE, с помощью которой была создана БД, и последовательный запуск программ с командой ALTER DATABASE, которые выполнялись для модернизации данной БД. Если такие программы не были сохранены, то необходимую информацию о БД можно найти в системных таблицах.
5. Выполните восстановление данных на основе архивной копии.
6. Проверьте и при необходимости отредактируйте данные, которые изменялись уже после создания последней архивной копии.

### 2.1.2 Восстановление объектов SQL Server

**Восстановление БД.**

После восстановления устройства выполните следующие действия для воссоздания поврежденной БД:

1. Выполните приведенный ниже запрос для проверки использовавшихся поврежденной базой данных устройств:

#### SELECT segmap, ‘size’ = size/512 FROM sysusages

#### WHERE dbid =

#### (SELECT dbid FROM sysdatabases

WHERE name = ‘testDB’)

Таким образом, мы определили размеры блоков для БД ***testDB.*** Вы можете и( пользовать те же параметры при пересоздании БД.

2. Создайте заново БД, используя команду CREATE DATABASE, дублирующую все записи старой таблицы ***sysusages,*** начиная с первой и до первого устройства, на котором располагался журнал транзакций:

CREATE DATABASE testDB ON datadevl = 20, datadev2 *=* 10

LOG ON logdevl *=* 10

3. При необходимости используйте команду ALTER DATABASE для расширения пространства на отдельных устройствах. Например, для увеличения размера, занимаемого БД на устройстве *datadev**l,* выполните следующую команду:

###### ALTER DATABASE testDB ON datadevl = 2

Для выделения места для БД на другом устройстве можно выполнить команду:

###### ALTER DATABASE testDB ON datadev3 = 2

При выполнении команды ALTER DATABASE не забудьте, что при выделении места на устройстве, которое уже частично занято данной БД, оно автоматически будет использоваться так же, как и уже имеющееся пространство (если были записаны данные, то и будут записываться данные, если журнал транзакций — то журнал транзакций). Когда вы выделяете место на устройстве, которое до этого не использовалось данной БД, оно по умолчанию будет применяться для хранения данных. Для размещения на таком устройстве журнала транзакций выполните кoмaнду ALTER DATABASE с системной процедурой sp\_logdevice. Например:

ALTER DATABASE testDB ON logdev2 = 4

EXEC sp\_logdevice testDB, logdev2

4. Перезагрузите БД, используя команду LOAD DATABASE, и затем загрузите архив журнала транзакций и последнюю архивную копию текущего журнала транзакций, используя команду LOAD TRANSACTION.

Для загрузки БД применяется следующий шаблон:

LOAD DATABASE *<**Иия БД>*

FROM *<Архивное устройство 1>* [, ... ]

[WITH *<Опции>]*

Для загрузки журнала транзакций используйте следующий шаблон:

LOAD TRANSACTION *<Имя БД>*

FROM *<Архивное устройство* 1*>* [, ... ]

[WITH *<0пции*>]

Для загрузки заголовка примените следующий шаблон:

LOAD HEADERONLY

FROM *<Архивное устройство>*

**Восстановление БД *master*.**

Повреждения БД *master* не могут остаться незамеченными, так как они вызовут невозможность запуска SQL Server или приведут к ошибкам чтения/записи данных. Наиболее распространенным случаем повреждения БД *master* является выход из строя жесткого диска во время записи данных.

Процедура восстановления поврежденной БД *master* отличается от восстановления БД пользователей. Если БД *master* недоступна из-за повреждения данных, ее придется пересоздать, используя SQL Server Setup, и затем восстановить на основе последней архивной копии. Естественно, изменения, выполненные после создания последней архивной копии, будут потеряны и их придется повторить еще раз.

Для восстановления БД *master* выполните следующие действия:

1. Запустите программу SQL Setup для пересоздания БД *master.* Эта БД должна быть создана с теми же кодовой страницей и порядком сортировки, которые использовались на момент создания последней архивной копии. Также необходимо убедиться, что указано такое же расположение, путь, имя и размер устройства.

2. Добавьте к устройствам сервера архивное устройство, если копия *BQ.master* находится не на дискетах.

3. Запустите SQL Server в однопользовательском режиме, как описано выше.

4. Восстановите БД, используя ее последнюю архивную копию.

5. Выполните изменения, которые не были отражены в архивной копии.

6. Восстановите БД *msdb.*

**Восстановление БД *master* без архивной копии**

При восстановлении БД *master* бывают и более тяжелые ситуации, когда:

* + повреждено устройство, на котором расположена БД master;
  + повреждена копия журнала транзакций или её не существует. При этом остались в целости данные для БД пользователей.

В случае повреждения копии журнала транзакций для восстановления БД *master* выполните следующие действия:

1. Запустите программу SQL Setup.

2. Создайте БД *master c* той же кодовой страницей и порядком сортировки, которые использовались на момент записи последней архивной копии. Также необходимо убедиться в правильности расположения, пути, имени и размера устройства *master.*

3. Запустите SQL Server в однопользовательском режиме.

4. Выполните команду DISK REINIT для инициализации устройств, имена которых не сохранились в таблице ***sysdevices***.

DISK REINIT

NAME = <Логическое имя>,

PHYSNAME = <Физическое имя>,

VDEVNO = <Номер устройства>,

SIZE = <Число страниц>

Чрезвычайно важно указать ранее использовавшиеся значения для параметров VDEVNO и SIZE. Если вы испытываете проблемы с определением правильных значений, попробуйте получить информацию о них, запустив системную процедуру sp\_helpdevice или просмотрев таблицы *sysdevices,* *sys-databases* и *sysusages* в БД *master.* Команда DISK REINIT восстанавливает ссылки на устройство в системных таблицах, когда физически устройство не потеряно (файл существует), а только отсутствует запись о нем в таблице *sysdevices.* Эту команду необходимо выполнить для каждого такого устройства. Подобная ситуация возникает, когда копия, на основе которой восстанавливается БД *master,* но соответствует последнему состоянию системы.

5. Выполните команду DISK REFIT для перестройки таблиц *sysusages* и *sysdatabases.* Эта команда позволяет восстановить записи в указанных таблицах для всех команд CREATE DATABASE и ALTER DATABASE, а также удалить записи, потерявшие актуальность.

Команду DISK REFIT необходимо выполнить сразу после команды DISK REINIT.

**Восстановление БД *msdb*.**

При восстановлении БД *master* программа Setup стирает и создает заново БД *msdb,* в результате чего теряется вся информация о планировании задач. После восстановления БД *master* выполните следующие действия для восстановления Д *msdb:*

1. Увеличьте, при необходимости, объем устройства *msdbdata* и выделите пространство для размещения *БД msdb.* Оно не должно быть меньше пространства, выделенного до восстановления *БД master.*

2. Восстановите (обычным образом) БД *msdb* из наиболее поздней архивной копии.

3. Восстановите журнал транзакций по архивным копиям.

4. Повторите установку задач, которые должны выполняться в определенное время и которые не удалось восстановить на основании архивной копии.

**Восстановление отдельной таблицы.**

Ecли для восстановления таблицы вы используете команду LOAD TABLE, то следует помнить, что ее опции совершенно идентичны опциям команды LOAD DATABASE за исключением SOURCE и APPEND. Опция SOURCE определяет имя таблицы, на основании которой восстанавливается данная таблица. Эта опция используется только тогда, когда имена восстанавливаемой таблицы и таблицы в архивной копии различны. Опция APPEND служит для указания необходимости добавить данные из архивной копии в таблицу, которая уже имеет данные.

Несмотря на простоту и некоторые удобства, которые предоставляет восстановление отдельных таблиц, необходимо отметить следующие важные ограничения:

* + Нельзя восстанавливать таблицы, имеющие столбцы типа text или image.
  + Индексы таблиц не восстанавливаются.
  + Архивная копия таблицы должна быть выполнена на сервере с тем же типом процессора, что и тот, на котором происходит ее восстановление (впрочем, это относится и к восстановлению других объектов SQL Server).
  + Восстанавливаемая таблица должна быть обязательно пользовательского типа, тогда как архивная копия таблицы может быть и пользовательской и системной.

### 2.2 Обмен данными

Управление данными является одной из важнейших задач системного администратора. К процессу управления данными относятся:

* + передача данных в БД из разных источников;
  + преобразование данных из БД SQL Server в другой формат;
  + распространение данных на другие сервера по сети или с помощью съемных носителей данных;
  + загрузка архивной копии БД.

### 2.2.1 Копирование данных

Для копирования данных в SQL Server 7.0 могут использоваться SQL Enterprise Manager и специальная командная утилита Bulk Copy Program (bcp).

**Копирование с помощью SQL Enterprise Manager.**

С помощью SQL ЕМ вы можете выполнять следующие действия:

* + копировать объекты между SQL Server, работающими в разных ОС;
  + копировать объекты между SQL Server, на которых установлены разные порядки сортировки данных;
  + отбирать при копировании только объекты определенного типа или некоторые объекты указанного типа;
  + выполнять передачу только структуры, только данных или и структуры, и данных;
  + дополнять или заменять существующие данные;
  + удалять существующие объекты перед копированием новой структуры.
  + включать объекты, связанные с передаваемым объектом;
  + выбрать программу, которая используется для копирования объектов и обеспечения безопасности;
  + немедленно выполнить передачу данных или спланировать это действие в виде единичной или регулярной акции.

Для передачи объектов между БД запустите SQL ЕМ, в окне Server Manager выберите требуемый сервер и находящуюся на нем БД. В меню Tools выполните команду Data Transformation Services\Import или Data Transformation Services\Export. На экране появится одноименное окно диалога. Далее, запускается DTS Import Wizard или DTS Export Wizard (в зависимости от выбора) которые организуют простой и доступный даже самому начинающему пользователю способ переноса данных.

**Копирование с помощью программы Bulk Copy (bcp).**

Более широкие возможности для передачи данных различных форматов предоставляет программа Bulk Copy (bcp). Она не имеет графического интерфейса. Утилита bcp предназначена главным образом для передачи данных между SQL Server и СУБД других типов. Она позволяет копировать данные SQL Server в файлы разных форматов.

Используйте программу bcp для передачи данных в следующих случаях:

* копирование таблиц при несовпадении структур исходной и целевой таблицы. В этом случае данные из таблицы копируются в файл, а затем из файла в целевую таблицу;
* передача больших объемов данных в (или из) SQL Server;
* передача столбцов или данных в другие программы, в том числе в табличные процессоры;
* передача данных, которые хранятся в текстовом формате (потребуется указать символы-разделители для столбцов и строк, например табуляция, запятая и т. д.);
* передача данных между мэйнфреймами и SQL Server.

COBET— Программа bcp не может быть запушена с помощью графических средств SQL ЕМ или ISQL/w. Для ее запуска используйте командную строку или командный файл.

Чаще всего для работы с программой bcp создают командный файл (ВАТ-файл), который позволяет задать сразу несколько опций, определяющих условия ее работы.

Рекомендуется использовать следующие расширения для файлов, создаваемых программой bcp:

* + BCP — собственный формат;
  + ТХТ — символьный формат;
  + ERR — файл ошибок;
  + FMT — форматный файл.

Для выполнения программы bcp необходимо иметь следующие разрешения для работы с таблицами:

* + Для исходной таблицы необходимо иметь разрешение па выполнение команды SELECT. Такое же разрешение вы должны получить для таблиц *sysobjects,* *syscolumns* и *sysindexes,* хотя для любого пользователя, входящего в группу public, такое разрешение устанавливается автоматически по умолчанию.
  + Для целевой таблицы необходимо иметь разрешение INSERT.

При копировании данных их запись выполняется динамически. Другие пользователи вполне могут применять целевые БД для записи, однако это может привести к соперничеству блокировок.

Считывание данных, напротив, требует монопольного использования БД, так как выполняется проверка целостности. Для предотвращения доступа к БД других пользователей достаточно с помощью SQL ЕМ в окне диалога Edit Database установить флажок, разрешающий доступ к БД только ее владельцу. Если по каким-либо причинам делать это нежелательно, в первой сессии начните транзакцию и заблокируйте возможность изменения данных другими пользователями:

BEGIN TRANSACTION

SELECT COUNT (\*)

FROM <Имя таблицы> HOLDLOCK

Опция HOLDLOCK оставляет другим пользователям возможность считывания данных.

Запустите вторую сессию, сформируйте команду для запуска программы bcp, вернитесь к первой сессии и завершите транзакцию:

COMMIT TRANSACTION

Для задания различных условий выполнения программы bcp можно использовать опции, перечисленные в таблице №20.

Таблица №20 - Опции программы bcp

|  |  |
| --- | --- |
| *Параметры* | *Описание* |
| **1** | **2** |
| **Обязательные** | |
| Database\_name.owner.  Table\_name | Имя БД, при необходимости с указанием её владельца и имени таблицы или представления |
| Datafile | Имя файла и полный путь к нему. Можно копировать данные с нескольких дискет (указав имя накопителя на гибких магнитных дисках) |
| In | Out | Направление перемещения данных (in – копирование данных из файла в таблицу SQL Server, out – копирование данных в файл) |
| */*U login\_d | Имя регистрации пользователя |
| */*S servername | Имя сервера, с которым необходимо установить соединение |
| /P password | Пароль |
| **Необязательные** | |
| /m maxerrors | Максимально допустимое количество ошибок при передаче данных |
| /f formatfile | Создание форматного файла для будущего использования. В форматном файле вы можете сохранить установки для типов данных, копируемых или пропускаемых столбцов в таблицах и т. д. Для пропуска столбцов задайте для них нулевую длину |
| /e errfile | Создание файла ошибок, куда записываются строки, которые не удалось передать |
| /F firstrow /L lastrow | Определяют первую и/или последнюю запись при передаче части таблицы или представления |
| /b batchsize | Число записей, которое передается в одном пакетном файле. По умолчанию все копируемые записи передаются одним пакетом |

Продолжение таблицы №20

|  |  |
| --- | --- |
| **1** | **2** |
| /n native | Копирует данные, используя принятый для них тип данных. Применяется при передаче от одного SQL Server к другому. Файл с данными будет иметь двоичный формат |
| /c character | Копирует данные, используя для всех столбцов символьный тип данных. По умолчанию поля отделяются символом табуляции, а каждая новая запись начинается с новой строки. Применяется при импорте и экспорте данных SQL Server. Файл с данными будет иметь текстовый формат |
| /E | При копировании данных в таблицу SQL Server указывает на наличие в данных уникального идентификатора записей |
| /t field\_term | Символ, используемый для разделения полей |
| /r row\_terminator | Символ используемый для разделения записей |
| /i inputfile /o outputfile | Определяет имя исходного (целевого) файла |
| /v | Отображает информацию о текущей версии DB-Library |
| /a packet\_size | Определяет размер (в байтах) сетевого пакета, с помощью которого осуществляется обмен данными с сервером. Значение по умолчанию равно 4096. Оно может изменяться в пределах от 512 до 65 535. При устойчивой работе сети увеличение размера пакета должно привести к повышению скорости передачи данных. Однако исследования, выполненные Microsoft, показали, что максимальная скорость достигается при размере пакета от 4096 до 8192 |

В следующем примере данные из таблицы *model,* расположенной в БД *auto,* копируются в текстовый файл MODEL. BCP:

BCP auto. .model in D:\AUTO\MODEL.BCP /с /t","/r \n /e

D:\AUT.O\MODEL.ERR /mlOO /b500 /a8192 /uAndreyG /sqlserver

Опции, использованные в этой команде, определяют следующие особенности передачи:

* + данные будут записаны в символьном (текстовом) виде;
  + для отделения полей в строке будет использована запятая;
  + нескопированные по каким-то причинам записи будут сохранены в файле MODEL.ERR в папке D:\AUTO;
  + каждая запись начинается с новой строки;
  + максимальное число ошибок, допускаемое в процессе копирования, — 100;
  + данные будут передаваться по 500 записей;
  + размер сетевого пакета 8192 байта;
  + имя регистрации «AndreyG»;
  + имя сервера «sqlserver».

Если какие-то из обязательных параметров не указаны, то программа bcp запросит их. В вышеприведенном примере не указан пароль. Он будет запрошен программой сразу после ее запуска.

В целом, последовательность действий при копировании данных с помощью программы Ьср и необходимые для этого права исполнителей приведены и таблице №21.

Таблица №21 -Действия при использовании программы **bcp**

|  |  |
| --- | --- |
| *Выполняемое действие* | *Пользователь* |
| Установка опции *select into/bulk copy* в значение истина | Системный администратор или владелец БД |
| Удаление индексов | Владелец таблицы |
| Выполнение копирования | Пользователь, имеющий разрешение INSERT |
| Установка опции *select into/bulk copy в* значение ложь | Системный администратор или владелец БД |
| Выполнение CHECKPOINT | Системный администратор или владелец БД |
| Создание архивной копии БД (при копировании очень важной информации) | Системный администратор или владелец БД |
| Пересоздание индексов | Владелец таблицы |
| Проверка правил ввода и целостности данных | Системный администратор или владелец БД |
| Создание архивной копии БД | Системный администратор или владелец БД |

**Копирование с помощью команд DUMP и LOAD.**

Команды DUMP и LOAD обычно используются для создания архивных копий и восстановления данных. Однако их можно использовать и для копирования данных между серверами на уровне БД. Тем самым удается организовать передачу целых БД с помощью выполнения заранее написанных программ, в то время как использование bср позволяет скопировать лишь отдельные таблицы. Команды DUMP и LOAD позволяют, при необходимости, вручную синхронизировать данные между различными серверами.

Если данные копируются на сервер, где ранее они не хранились, перед загрузкой архивной копии необходимо создать устройство и соответствующую базу данных.

### 2.2.2 Тиражирование данных на сменных носителях

**Создание БД.**

База данных, предназначенная для дальнейшего распространения, должна быть создана на новом устройстве. Оно не должно использоваться никакими другими БД (чтобы гарантировать непрерывность пространства, занимаемого данными). Необходимо использовать, по крайней мере, три устройства: одно для системных таблиц, одно для журнала транзакций и одно (как минимум) для данных.

Вы можете существенно облегчить процесс тиражирования, используя системную процедуру **sp\_create\_removable**:

sp create removable <Имя БД>, <Логическое имя устройства>,

<Физическое имя устройства>, < Размер устройства>,

<Логическое имя журнала транзакций>, «Физическое имя журнала транзакций>, <Размер журнала транзакций>, <Логическое имя устройства для данных\_1>, <Физическое имя устройства для данных\_1>, <Размер устройства для данных\_1> [... , <Логическое имя устройства для данных\_16>, <Физическое имя устройства для данных\_16>, <Размер устройства для данных\_16>]

Для запуска этой процедуры необходимо иметь права системного администратора.

**Подготовка БД.**

Перед записью БД на сменный носитель необходимо убедиться в том, что:

* + в БД отсутствуют пользовательские объекты;
  + запрещены разрешения для пользователей;
  + системный администратор является владельцем БД и всех ее объектов;
  + все фрагменты устройства непрерывно размещаются на диске.

Для выполнения такой проверки используйте системную процедуру:

sp certify\_removabie *<*Имя БД> [, AUTO]

Эта процедура также очищает журнал транзакций, удаляет устройство для него и переводит БД в режим, недоступный для пользователей.

Процедура возвращает информацию, которая записывается в текстовый файл, имя которого формируется в соответствии со следующим шаблоном:

CertifyR\_<Имя БД>.txt

**Размещение БД.**

После проверки БД создайте её мастер-копию и скопируйте на сменный носитель. Мастер-копия должна содержать устройство, на котором размещены системные таблицы, и устройства, на которых размещены данные. Проверьте мастер-копию на соответствие оригиналу, используя данные, возвращаемые системной процедурой **sp\_certify\_removable**.

## 3 Контрольные вопросы

1. Выполнить восстановление данных, копирование данных и создать тиражную копию БД.

**4 Контрольные вопросы**

1. Что подразумевает под собой восстановление данных, для чего применяется?
2. Что необходимо учитывать при восстановлении данных из журнала транзакций?
3. Перечислить основные методы восстановления данных, пояснить каждый из них.
4. Какие действия совершаются при автоматическом восстановлении?
5. Какие действия совершаются при вынужденном восстановлении?
6. Какие действия необходимо совершить для восстановления БД?
7. Пояснить назначение команд CREATE DATABASE, ALTER DATABASE, LOAD DATABASE, LOAD TRANSACTION?
8. Что необходимо учитывать при восстановлении отдельной таблицы?
9. Какими средствами можно осуществлять копирование данных в SQL Server?
10. Перечислите основные возможности SQL по управлению данными?
11. В каких случаях удобнее использовать Bulk Copy (bcp)?
12. Какие команды можно применять для копирования данных?
13. При помощи какой процедуры можно осуществляется процесс тиражирования данных? Описать синтаксис этой команды.

# Лабораторная работа №13 - Подключение базы данных SQL к проекту VISUAL STUDIO.

# 1 Цель работы

Ознакомиться с основными сведениями о проектировании и создании приложений с БД.

**2 Основные теоретические сведения**

Рассмотрим последовательно все этапы подключения базы данных к среде Microsoft Visual Studio.

1. Добавляем на форму компонент DataGridView (рисунок 54).

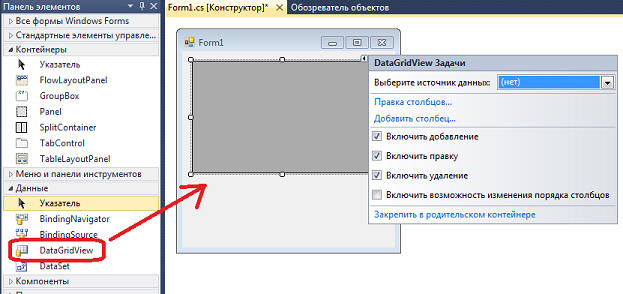


Рисунок 54 – Добавление на форму компонента DataGridView

1. Выбираем источник данных или создаем новый источник данных (рисунок 55).

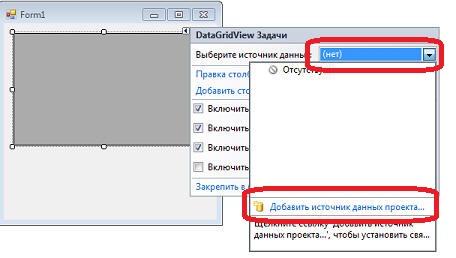


Рисунок 55 – Выбор источника данных

1. Откроется мастер настройки источника данных. Выбираем «База данных» и нажимам кнопку «Далее» (рисунок 56).

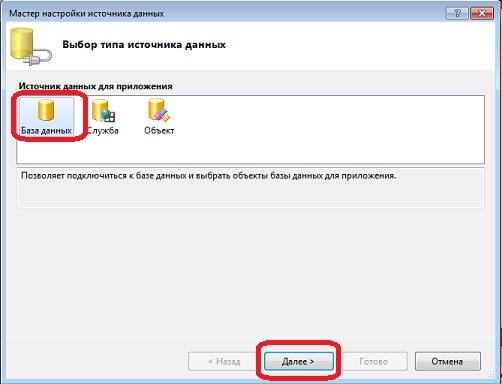


Рисунок 56 – Мастер настройки источника данных

1. Выбираем «Набор данных» и нажимаем кнопку «Далее» (рисунок 57).

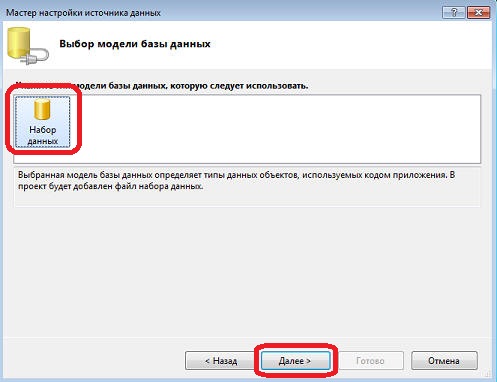


Рисунок 57 – Мастер настройки источника данных

1. Создаем подключение (рисунок 58).

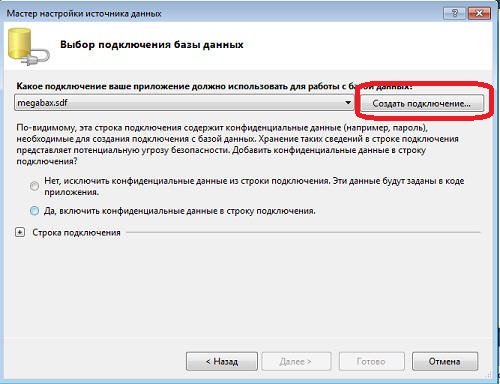


Рисунок 58 – Работа с мастером источника данных

1. В диалоге создания подключения, выбираем источник данных «Microsoft SQL Server DataBase File (SqlClient)», указаваем путь к созданной базе данных и пароль к ней (рисунок 59).

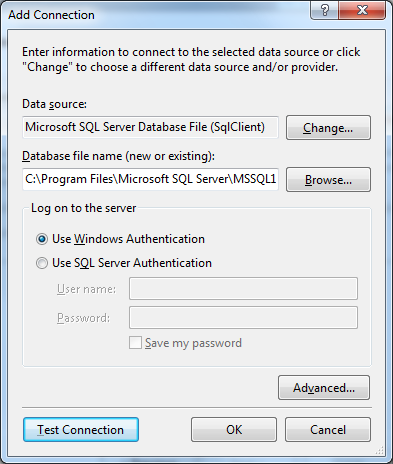


Рисунок 59 – Диалог Add Connection

1. Нажав на кнопку Test Connection, можно проверить сможет ли программа получить доступ к базе данных. В свойствах Дополнительно(Advanced) задаются расширенные настройки подключения, такие как строка подключения, относительное месторасположение базы и т.д. На следующем этапе нажимаем кнопку «Далее».

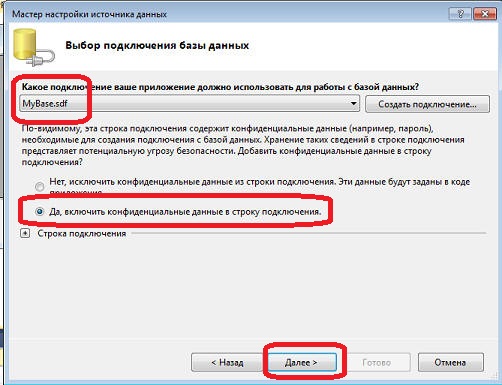


Рисунок 60 – Работа с мастером источника данных

1. Программа предложит скопировать базу данных в папку с проектом. Если нажать «Да», то нужная база данных будет скопирована в папку с проектом и вся работа будет вестись с ней. В противном случае вся работа будет осуществляться с базой, по указанному месторасположению при настройке подключения (рисунок 61).

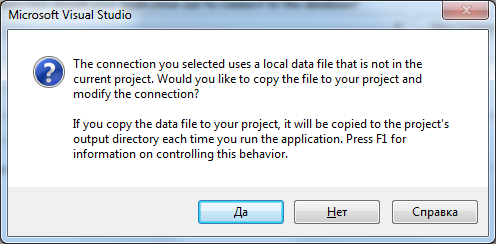


Рисунок 61 – Копирование базы данных

1. Сохранение подключения в файле конфигурации приложения.

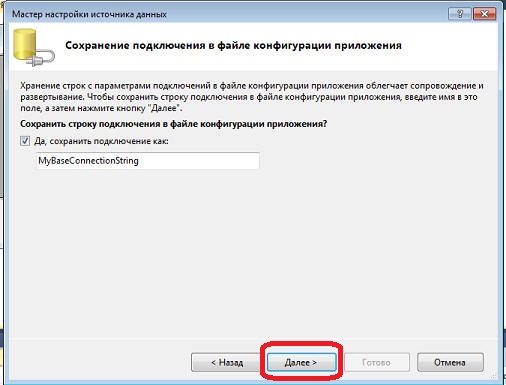


Рисунок 62 - Сохранение подключения в файле конфигурации приложения

1. В окне диалога появляется список данных, доступных для выбора (рисунок 63).

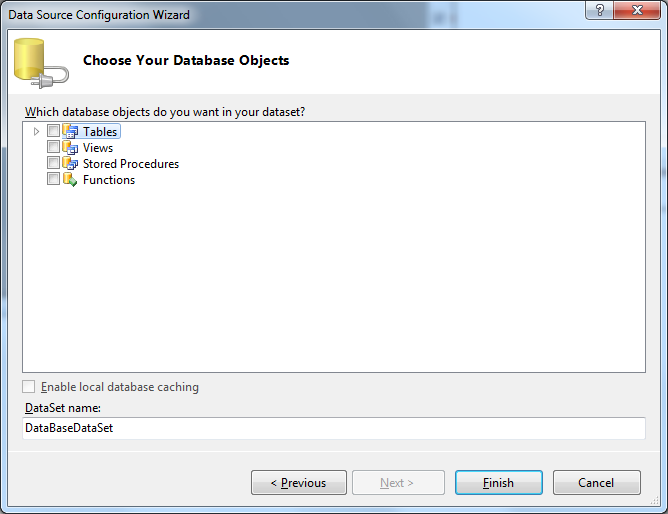


Рисунок 63 – Список данных, доступных для выбора

1. Выбираем объекты базы данных (рисунок 64).

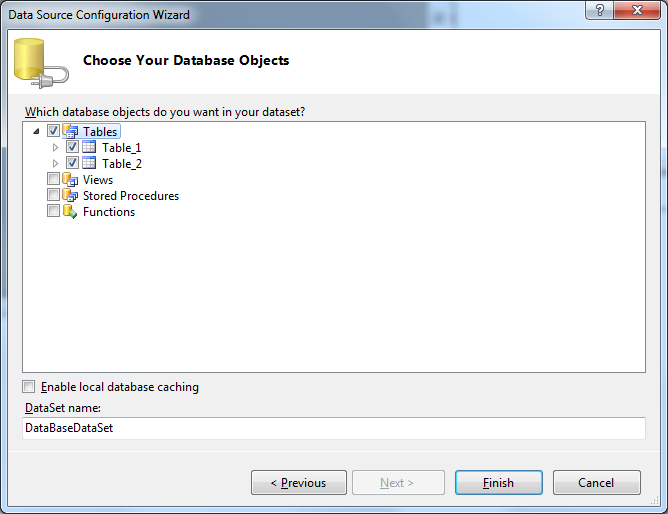


Рисунок 64 – Выбор объектов базы данных

1. Выделяем элемент DataGridView, нажимаем на треугольник в правом верхнем углу элемента и выбираем одну из таблиц для отображения, как указано на рисунке 65.

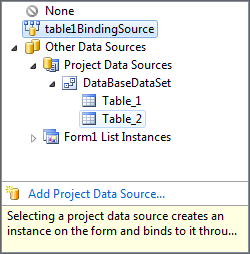


Рисунок 65 – Выбор таблицы для отображения

Выбранная таблица отображается в элементе (рисунок 66).

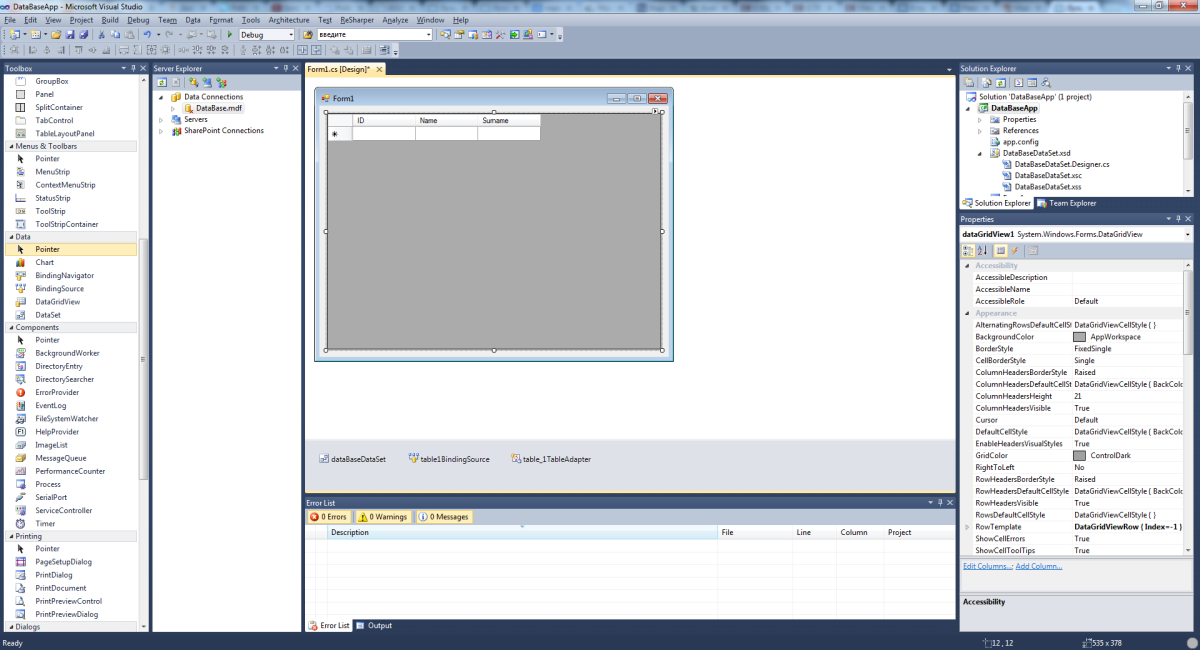


Рисунок 66 – Отображение таблицы в элементе DataGridView

Подключение базы данных выполнено. В заключение приведем пример готового приложения, разработанного в среде Visual Studio. Представленная программа предназначена для редактирования баз данных, созданных при использовании SQL сервера любого типа. Основная цель программы – упрощение для пользователя процесса внесения любых изменений в базу данных. Программа выполняет выбор сервера, подключение к серверу, выбор базы данных для редактирования, выбор таблицы из базы данных, внесение изменений в таблицу и сохранение результатов. Главное окно программы приведено на рисунке 67.

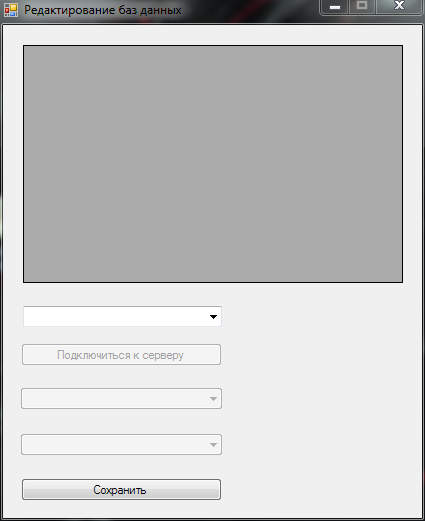


Рисунок 67 – Программа редактирования баз данных

**Код программы с комментариями**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Data.SqlClient; //Подключение пространства имен для работы с SQL сервером

namespace WindowsFormsApplication4

{ public partial class Form1 : Form

{ SqlConnection myConnection; //Создание нового подключения

SqlDataAdapter adapter; // Создание адаптера для работы с БД

DataTable DT; //Обьявление DataTable для работы с таблицей

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

myConnection = new SqlConnection(@"Data Source=" + comboBox1.Text + ";Integrated Security=True"); //Инициализируем соединение

/\*Пытаемся его открыть\*/

try

{

myConnection.Open();

}

catch (Exception)

{

MessageBox.Show("Не удалось открыть подключение. Проверьте имя сервера.", "Ошибка подключения", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

DataTable schemaTable = myConnection.GetSchema("Databases"); //Используем myConnection.GetSchema для отображения всех баз данного сервера

comboBox2.Items.Clear();

foreach (DataRow r in schemaTable.Rows)

{

comboBox2.Items.Add(r[0]);

}

myConnection.Close(); //Закрываем соединение

comboBox2.SelectedIndex = -1;

comboBox2.Enabled = true;

comboBox3.Enabled = false;

}

private void comboBox1\_DropDown(object sender, EventArgs e)

{

this.Cursor = System.Windows.Forms.Cursors.WaitCursor;

/\*Отображаем список всех доступных серверов\*/

System.Data.Sql.SqlDataSourceEnumerator NI = System.Data.Sql.SqlDataSourceEnumerator.Instance;

comboBox1.DataSource = NI.GetDataSources();

this.Cursor = System.Windows.Forms.Cursors.Default;

}

private void comboBox2\_TextChanged(object sender, EventArgs e)

{

if ((sender as ComboBox).Text == "")

{

comboBox3.Enabled = false;

}

else { comboBox3.Enabled = true; }

//Обновляем соединения т. к. теперь нам уже известна база

myConnection = new SqlConnection(@"Data Source=" + comboBox1.Text + ";Initial Catalog=" + comboBox2.Text + ";Integrated Security=True");

try

{

myConnection.Open();

}

catch (Exception)

{

MessageBox.Show("Не удалось открыть подключение. Проверьте имя сервера.", "Ошибка подключения", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

/\*Вывод на экран всех таблиц\*/

DataTable schemaTable = myConnection.GetSchema("Tables");

comboBox3.Items.Clear();

foreach (DataRow r in schemaTable.Rows)

{ comboBox3.Items.Add(r[2]);

}

comboBox3.Enabled = true;

comboBox3.SelectedIndex = -1;

myConnection.Close();

}

private void comboBox1\_TextChanged(object sender, EventArgs e)

{

if ((sender as ComboBox).Text == "")

{

button1.Enabled = false;

}

else { button1.Enabled = true; }

}

private void comboBox3\_TextChanged(object sender, EventArgs e)

{ string myquery1 = "SELECT \* FROM " + comboBox3.Text; //Создаем запрос для отображения БД

SqlCommand cmd = new SqlCommand(myquery1, myConnection); //Инициализируем команду

adapter = new SqlDataAdapter(cmd); //Инициализируем адаптер

DT = new DataTable(); //Инициализируем DataTable

SqlCommandBuilder commandBuilder = new SqlCommandBuilder(adapter); //Получаем список команд из адептера и...

adapter.Fill(DT); // ...инициализируем ими DataTable

dataGridView1.DataSource = DT; // Привязываем dataGridView1 к DataTable

/\*Поправляем размер dataGridView1\*/

int FullSize = 0;

for (int i = 0; i < dataGridView1.ColumnCount; i++)

{ FullSize += dataGridView1.Columns[i].Width;

}

FullSize += dataGridView1.RowHeadersWidth + 21;

dataGridView1.Width = FullSize;

this.Width = dataGridView1.Width + 50;

}

private void button2\_Click(object sender, EventArgs e)

{

/\*Пытаемся сохранить базу на сервере\*/

try

{

adapter.Update((DataTable)dataGridView1.DataSource);

MessageBox.Show("Таблица сохранена.", "Сохранение", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{ MessageBox.Show("Не удалось сохранить таблицу. Текст ошибки : \" " + ex.Message + ".\"", "Ошибка записи", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

**3 Задание для самостоятельной работы**

Разработать приложение к БД средствами выбранной среды программирования. Обеспечить ведение базы данных и организацию системы запросов к базе данных. Предусмотреть возможность печати результатов выборки.

# Список рекомендуемой литературы

1. И.А. Кумскова. Базы данных. – М.: КносРус, 2011. – 488 с.
2. [Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской](http://www.ozon.ru/context/detail/id/7346276/#tab_person). Базы данных. Теория и практика. – М.: Юрайт, 2012. - 464 с.
3. [Билл Карвин](http://www.ozon.ru/context/detail/id/7598812/#tab_person). Программирование баз данных SQL. Типичные ошибки и их устранение. – М.: Рид Групп, 2012. – 366 с.
4. [В. Дунаев](http://www.ozon.ru/context/detail/id/18569532/#tab_person). Базы данных. Язык SQL для студента. – Санкт – Петербург: [БХВ-Петербург](http://www.ozon.ru/context/detail/id/1098685/), 2012. – 320 с.
5. Голицына О.Л., Максимов H.B., Попов И.И. Базы данных: Учебное пособие. - М.: ФОРУМ: ИНФРА – М, 2005.
6. Кренке Д. Теория и практика построения баз данных: [пер.с англ] / Д. Кренке. - 9 - е изд. - СПб.: Питер, 2005. - 858 с.
7. Диго СМ. Базы данных - М.: Финансы и статистика, 2005.
8. Марков А.С., Лисовский К.Ю. Базы данных. Введение в теорию и методологию: Учебник. - М.: Финансы и статистика, 2008.
9. Когаловский М.Р. Энциклопедия технологий баз данных. М.: Фи­нансы и статистика, 2006.
10. Базы данных. Учебник для высших заведений /Под редакцией проф. А.Д. Хомоненко - СПб: Корона принт, 2010, 416 с.

Базы данных

Методические указания к лабораторным работам

Внутри кафедральное издание

Составитель Янаева Марина Викторовна

Компьютерная верстка М.В. Янаева