

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Н. Н. Пустовалова, Е. А. Блинова

БАЗЫ ДАННЫХ. MICROSOFT SQL SERVER
ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Минск 2022

ПРЕДИСЛОВИЕ

Практикум содержит задания для выполнения лабораторных работ для изучения базы данных Microsoft SQL Server. В каждой работе имеются краткие теоретические сведения по рассматриваемым вопросам.

Преподаватель определяет, какие лабораторные работы должны выполнять студенты и в каком объеме. Предполагается, что выполнение большинства лабораторных работ занимает у студентов два академических часа.

Задания, отмеченные звездочками, имеют повышенную сложность.

Задания для выполнения лабораторных работ содержат также кнопки, при нажатии на которые открываются тесты, предназначенные для контроля знаний студентов. Тестирование происходит по команде преподавателя и занимает от 1 до 3 минут. Для работы тестирующих программ предварительно в приложении Word надо разрешить использование макросов. Тексты ответов на формах располагаются каждый раз случайным образом, и ответить на вопросы можно только один раз, так как после нажатия на кнопку «Результаты» форма с вопросами и вариантами ответов исчезает.

Для оформления лабораторных работ рекомендуется использовать приложение Microsoft Word. Каждая работа должна содержать название работы, условия заданий, тексты разработанных sql-скриптов, результаты. Должны быть приведены решения для базы данных UNIVER (приведена в приложении) и X MyBase (база данных студента согласно индивидуальному заданию).

В верхнем колонтитуле следует записать фамилию студента и номер группы, в нижнем – номера страниц. Шрифт – 10, интервал – одинарный, поля по 1,5 см.

ОГЛАВЛЕНИЕ

Лабораторная работа № 1. [Основы работы в SQL Server Management Studio](#)

Лабораторная работа № 2. [Проектирование баз данных. Нормализация](#)

Лабораторная работа № 3. [T-SQL – язык реляционной базы данных](#)

Лабораторная работа № 4. [Многотабличные SELECT-запросы](#)

Лабораторная работа № 5. [Использование подзапросов](#)

Лабораторная работа № 6. [Группировка данных](#)

Лабораторная работа № 7. [Анализ данных и комбинирование результатов запросов](#)

Лабораторная работа № 8. [Использование представлений](#)

Лабораторная работа № 9. [Основы программирования на T-SQL](#)

Лабораторная работа № 10. [Создание и применение индексов](#)

Лабораторная работа № 11. [Обработка результатов запросов с помощью курсоров](#)

Лабораторная работа № 12. [Особенности использования транзакций](#)

Лабораторная работа № 13. [Разработка хранимых процедур](#)

Лабораторная работа № 14. [Разработка и использование функций](#)

Лабораторная работа № 15. [Применение DML-триггеров](#)

Лабораторная работа № 16. [Использование XML](#)

Приложение. [Примеры баз данных](#)

Лабораторная работа № 1. Знакомство с инструментами СУБД Microsoft SQL Server

Создавать базы данных можно с использованием утилиты **SQL Server Management Studio (SSMS)**, входящей в состав программного обеспечения **СУБД Microsoft SQL Server**, или с помощью языка **Transact-SQL**. В данной лабораторной работе рассматривается первый вариант.

Задание	Краткие теоретические сведения
<p>1. Создать базу данных с помощью утилиты SQL Server Management Studio. Имя базы – фамилия студента и слово ПРОДАЖИ.</p> <p>Изучить файлы, которые при этом создаются.</p>	<p>Для запуска утилиты SQL Server Management Studio (SSMS) надо выполнить: Пуск / Программы / Microsoft SQL Server / SQL Server Management Studio. Утилита используется для создания, администрирования и управления базами данных.</p> <p>Если панель Обозреватель объектов (Object Explorer) не видна, то надо выполнить Вид / Обозреватель объектов (View / Object Explorer). Для подключения к серверу следует выбрать Соединить / Компонент Database Engine / Соединить (Connect / Database Engine / Connect).</p> <p>Чтобы <u>создать базу данных</u> надо в контекстном меню пункта Базы данных (Databases) выполнить команду Создать базу данных (New Database). В диалоговом окне указать имя новой базы данных.</p>
<p>2. Создать таблицу ТОВАРЫ, содержащую поля:</p> <p>Наименование (nvarchar(20)), Цена (real), Количество (int).</p> <p>Сделать первый столбец первичным ключом.</p>	<p>Для <u>создания таблицы</u> следует раскрыть папку Базы данных (Databases) двойным щелчком, раскрыть нужную базу данных, в контекстном меню пункта Таблицы (Tables) выполнить команду Создать таблицу (New Table).</p> <p>Ввести имена столбцов (атрибутов) в таблицу и их свойства. Все типы данных отображаются, если щелкнуть мышью по знаку стрелки в столбце</p>

<p>3. Создать таблицу ЗАКАЗЧИКИ, содержащую поля: Наименование_фирмы (nvarchar(20)), Адрес (nvarchar(50)), Расчетный_счет (nvarchar(15)). Сделать первый столбец первичным ключом.</p> <p>4. Создать таблицу ЗАКАЗЫ, содержащую поля: Номер_заказа (nvarchar(10)), Наименование_товара (nvarchar(20)), Цена_продажи (real), Количество (int), Дата_поставки (date), Заказчик (nvarchar(20)). Сделать первый столбец первичным ключом.</p>	<p>Тип данных (DataType). Флажок в поле Разрешить (Allow Nulls) должен быть установлен, если можно вводить в этот столбец значение NULL. Для выбранного типа данных можно определять свойства в таблице Свойства столбца (Column Properties). Например, типу nvarchar (строковый тип переменной длины) требуется определить значение для строки Длина (Length), типу decimal требуется указать значения в строках Точность (Precision) и Масштаб (Scale). Типу int можно не задавать никаких значений в этих строках. Если для столбца существует значение по умолчанию, то его нужно ввести в строку Значение по умолчанию или привязка (Default Value or Binding) Чтобы сделать <u>столбец первичным ключом</u> надо щелкнуть по столбцу правой кнопкой мыши и выбрать пункт Задать первичный ключ (Set Primary Key). После создания таблицы следует закрыть окно с таблицей и в появившемся окне ввести имя таблицы для ее сохранения. Нажать кнопку Обновить (Refresh). С помощью контекстного меню можно просматривать свойства таблицы, переименовывать ее, удалять и т. п.</p>
<p>5. Установить связи между таблицами ЗАКАЗЧИКИ и ЗАКАЗЫ по полям Наименование_фирмы и Заказчик. Установить связи между таблицами ТОВАРЫ и ЗАКАЗЫ по полям Наименование и Наименование_товара.</p>	<p>Для <u>создания связей</u> надо в контекстном меню Диаграммы баз данных (Database Diagrams) выполнить команду Создать диаграмму базы данных (New Database Diagrams), в появившемся окне выделить таблицу ТОВАРЫ и нажать Добавить (Add), аналогично выбрать остальные таблицы. Сначала надо определить, какая таблица главная, и какая – подчиненная. Затем тянуть мышкой поле (Наименование) от главной таблицы (ТОВАРЫ) к нужному полю (Наименование_товара) в подчиненной таблице (ЗАКАЗЫ).</p>

Заполнить таблицы информацией (5-10 строк в каждой).	<p>Первыми <u>заполняются главные таблицы</u> (ТОВАРЫ и ЗАКАЗЧИКИ), затем подчиненные. Чтобы заполнить таблицу надо в ее контекстном меню выполнить команду Изменить первые 200 строк (Edit Top 200 Rows), в появившемся окне ввести данные.</p> <p>Чтобы увидеть созданные таблицы в списке, надо нажать Обновить.</p>
<p>6. Сформировать следующие запросы и проанализировать результаты:</p> <ul style="list-style-type: none"> – определить товары, поставки которых должны осуществиться после некоторой даты; – найти товары, цена которых находится в некоторых пределах; – определить названия фирм, заказавших конкретный товар; – найти заказы определенной фирмы по ее названию, отсортировать их по датам поставки. <p>Сохранить запросы в sql-скрипте.</p>	<p>Для <u>формирования запроса</u> надо щелкнуть по кнопке Создать запрос на панели инструментов. В контекстном меню появившегося окна следует выполнить команду Создать запрос в редакторе (Design Query in Editor), в окне Добавление таблицы (Add Table) выбрать нужные таблицы. В таблице или таблицах поставить галочки у полей, которые должны быть включены в запрос. Эти поля будут отображены в нижней части бланка запроса.</p> <p>В столбце Фильтр (Filter) бланка запроса следует указать условия отбора для нужных полей. При этом могут использоваться операторы сравнения (=, >, <, >=, <=, ><) и логические операторы AND, OR, NOT, а также оператор IN, который проверяет значение поля на равенство любому значению из списка (элементы списка отделяются друг от друга запятыми, список заключается в круглые скобки).</p> <p>Если условие налагается на несколько полей, и они связаны логическим оператором И, то условия вводятся в одном столбце Фильтр у нужных полей, если логическим оператором ИЛИ – то в разных столбцах Or.</p> <p>Чтобы отсортировать результат надо в столбце Порядок сортировки (Sort Type) для соответствующего поля выбрать тип сортировки (по возрастанию или по убыванию). После формирования запроса нажать Ок.</p>

Чтобы выполнить запрос надо щелкнуть по кнопке **Выполнить (Execute)** на панели инструментов. Результаты появятся на экране.

Для *сохранения* запроса можно использовать кнопку **Save** на панели инструментов.

Тест "SQL Server Management Studio"

[В начало практикума](#)

Лабораторная работа № 2. Проектирование баз данных. Нормализация

При проектировании реляционной базы данных необходимо исследовать предметную область с целью определения объектов, нормализовать данные и установить связи между ними. **Нормализация** данных – это процесс, в результате выполнения которого таблицы базы данных проверяются на наличие зависимостей между столбцами и, если необходимо, то исходная таблица разделяется на несколько таблиц.

Задание	Краткие теоретические сведения
1. Изучить способ приведения информации к <i>первой нормальной форме</i> , проанализировав пример в правой части.	<p>Пусть исходная информация о продажах имеет следующие поля:</p> <p>Наименование_товара, Цена, Количество_на_складе, Наименование_заказанного_товара, Цена_заказанного_товара, Количество_заказанного_товара, Общая_стоимость, Дата_поставки, Заказчик, Адрес_заказчика, Расчетный_счет_заказчика, Телефон_заказчика.</p> <p>Данные подлежат нормализации.</p> <p>Чтобы таблица соответствовала 1-й нормальной форме (1NF), необходимо, чтобы все значения ее полей были неделимыми и не вычисляемыми, а все записи – уникальными (не должно быть полностью совпадающих строк).</p> <p>Если много заказчиков купят один и тот же товар, то в таблице будут повторяться одни и те же исходные реквизиты товара. Для уменьшения избыточности информации, нужно преобразовать первоначальную таблицу к двум таблицам:</p> <p>Наименование_товара, Цена, Количество_на_складе</p> <p>и</p>

	<p>Наименование_заказанного_товара, Цена_заказанного_товара, Количество_заказанного_товара, Общая_стоимость, Дата_поставки, Заказчик, Адрес_заказчика, Расчетный_счет_заказчика, Телефон_заказчика.</p> <p>Первая таблица соответствует первой нормальной форме, а во второй таблице имеется избыточность. Если один заказчик купит много товаров, то в таблице будут повторяться одни и те же исходные реквизиты заказчика. К тому же поле Общая_стоимость может быть вычислено по полям Цена_заказанного_товара и Количество_заказанного_товара. Поэтому вторую таблицу надо разбить на две:</p> <p>Заказчик, Адрес_заказчика, Расчетный_счет_заказчика и Наименование_заказанного_товара, Цена_заказанного_товара, Количество_заказанного_товара, Дата_поставки, Заказчик, Телефон_заказчика.</p> <p>Поле Общая_стоимость, как вычисляемое, в структуру таблицы не включается.</p>
2. Изучить способ приведения ко <i>второй нормальной форме</i> , проанализировав пример в правой части.	<p>Чтобы таблица соответствовала 2-й нормальной форме (2NF), необходимо, чтобы она находилась в 1-й нормальной форме и все не ключевые поля полностью зависели от ключевого.</p> <p>Из полученных трех таблиц первые две соответствуют второй нормальной форме, а в третьей таблице нет ключевого поля. Если в качестве такового взять поле Наименование_заказанного_товара, то оно может принимать одно и то же значение для различных заказчиков. Поэтому требуется ввести новое поле, которое являлось бы первичным ключом для всех остальных.</p> <p>Таким ключом может стать поле Номер_заказа, которое надо добавить, т. е. третья таблица теперь должна содержать поля:</p>

	<p>Номер_заказа, Наименование_заказанного_товара, Цена_заказанного_товара, Количество_заказанного_товара, Дата_поставки, Заказчик, Телефон_заказчика.</p> <p>Определим имена таблиц:</p> <p>ТОВАРЫ (<u>Наименование товара</u>, Цена, Количество_на_складе); ЗАКАЗЧИКИ (<u>Заказчик</u>, Адрес_заказчика, Расчетный_счет_заказчика); ЗАКАЗЫ (<u>Номер заказа</u>, <u>Наименование_заказанного_товара</u>, <u>Цена_заказанного_товара</u>, <u>Количество_заказанного_товара</u>, <u>Дата_поставки</u>, <u>Заказчик</u>, <u>Телефон_заказчика</u>).</p> <p>Ключевые поля подчеркнуты.</p>
3. Ознакомиться с третьей нормальной формой.	<p>Для перехода к 3-й нормальной форме (3NF), необходимо обеспечить, чтобы все таблицы находились во 2-й нормальной форме и все не ключевые поля в таблицах не зависели взаимно друг от друга.</p> <p>В таблице ЗАКАЗЫ поля Заказчик и Телефон_заказчика взаимно зависимы. Чтобы привести таблицу к 3 нормальной форме надо поле Телефон_заказчика переместить в таблицу ЗАКАЗЧИКИ.</p> <p>Теперь полученные таблицы соответствуют всем требованиям.</p> <p>ТОВАРЫ (<u>Наименование товара</u>, Цена, Количество_на_складе); ЗАКАЗЧИКИ (<u>Заказчик</u>, Адрес_заказчика, Расчетный_счет_заказчика, Телефон_заказчика); ЗАКАЗЫ (<u>Номер заказа</u>, <u>Наименование_заказанного_товара</u>, <u>Цена_заказанного_товара</u>, <u>Количество_заказанного_товара</u>, <u>Дата_поставки</u>, <u>Заказчик</u>).</p>

4. Определить *типы данных* для полей нормализованных таблиц базы данных **ПРОДАЖИ**.

MSS поддерживает следующие типы данных: числовые, символьные, для хранения даты и времени, денежные, двоичные и специальные.

Тип данных	Описание	Тип данных	Описание
integer (int)	Целочисленные значения, занимают 4 байта.	date	Дата, занимает 3 байта
smallint	Целочисленные значения, занимают 2 байта	time[(p)] $0 \leq p \leq 7$	Время, занимает от 3 до 5 байтов. p – колич. знаков после точки в секундах
tinyint	Неотрицательные целочисленные значения, занимают 1 байт	smalldatetime	Дата и время, занимает 2 байта
bigint	Целочисленные значения, занимают 8 байт	datetime	Дата и время, занимает 4 байта
real	Вещественные числа с плавающей точкой	datetime2	Дата и время, занимает 8 байтов
decimal (p[,s]) (dec) или numeric (p[,s])	Вещественные значения с фикс. точкой, p – общее количество цифр, s – количество цифр после точки. Занимает от 5 до 17 байт.	char[(n)]	Строки фиксированной длины из однобайтовых символов, n – количество символов
float[(p)]	Вещественные числа с плавающей точкой. Если p < 25 – одинарная точность (4 байта), при p > 25 – двойная точность (8 байтов)	nchar[(n)]	Строки фиксированной длины символов Unicode. Каждый символ занимает 2 байта.
money	Денежные значения, занимают 8 байтов	varchar[(n)]	Строки переменной длины из однобайтовых символов
smallmoney	Денежные значения, занимают 4 байта	nvarchar[(n)]	Строки переменной длины символов Unicode – 2 байта
binary(n)	Задает битовую строку, длиной ровно n байтов	varbinary(n)	Задает битовую строку, длиной не более n байтов

5. В соответствии со своим вариантом, номер которого определяет преподаватель, провести нормализацию исходной информации из таблицы, представленной ниже, создав как минимум три таблицы. При необходимости использовать дополнительные поля.

Создать базу данных (имя базы – **X_MyBase**, где **X** – первые буквы своей фамилии) с помощью команд **Server Management Studio**, определить структуру таблиц, установить связи и заполнить таблицы информацией.

№ варианта	Исходная информация
1	Реализация изделий из стекла. Компания торгует изделиями из стекла. Информационные поля: Наименование товара, Цена, Описание, Количество на складе, Покупатель, Телефон, Адрес, Дата сделки, Количество заказанного товара.
2	Курсы повышения квалификации. В учебном заведении имеются курсы повышения квалификации. Информационные поля: Номер группы, Специальность, Отделение, Количество студентов, Код преподавателя, Фамилия преподавателя, Имя, Отчество, Телефон, Стаж, Количество часов, Предмет, Тип занятия, Оплата.
3	Фирма по продаже запчастей. Организация занимается продажей запасных частей для автомобилей. Некоторые из поставщиков могут поставлять одинаковые детали (один и тот же артикул). Информационные поля: Код поставщика, Название, Адрес, Телефон, Название детали, Количество деталей на складе, Артикул, Цена, Примечание, Количество заказанных деталей, Дата заказа.
4	Контроль доставки товаров. Организация продаёт и доставляет мебель. Информационные поля: Наименование фирмы-заказчика, Адрес, Телефон, Контактное лицо, Наименование товара, Цена, Описание, Количество заказанного товара, Дата поставки, Вид доставки.

№ варианта	Исходная информация
5	Организация факультативов для студентов. В высшем учебном заведении организованы факультативы. Существует минимальный объем часов предметов, которые должен прослушать каждый студент. Информационные поля: Фамилия студента, Имя, Отчество, Адрес, Телефон, Название предмета, Объем лекций, Объем практических занятий, Объем лабораторных работ, Оценка.
6	Учет внутриофисных расходов. Сотрудники частной фирмы имеют возможность осуществлять мелкие покупки в пределах некоторой суммы для нужд фирмы, предоставляя в бухгалтерию товарный чек. Информационные поля: Название отдела, Количество сотрудников, Название товара, Описание, Предельная норма расхода, Потраченная сумма, Дата.
7	Выдача банком кредитов. Кредит может получить юридическое лицо. Информационные поля: Название вида кредита, Ставка, Название фирмы-клиента, Вид собственности, Адрес, Телефон, Контактное лицо, Сумма, Дата выдачи, Дата возврата.
8	Грузовые перевозки. Компания осуществляет перевозки по различным маршрутам. Информационные поля: Название маршрута, Дальность, Количество дней в пути, Оплата, Фамилия водителя, Имя, Отчество, Стаж, Дата отправки, Дата возвращения.
9	Анализ показателей финансовой отчетности предприятий. В структуру холдинга входят несколько предприятий. Работа предприятия оценивается показателями (прибыль, себестоимость и пр.) Информационные поля: Название показателя, Важность показателя, Название предприятия, Банковские реквизиты, Телефон, Контактное лицо, Дата, Значение показателя.

№ варианта	Исходная информация
10	Склад. Компания предоставляет услугу хранения товара на складе. Информационные поля: Наименование товара, Цена, Количество, Описание, Место хранения, Покупатель, Телефон, Адрес, Дата сделки, Количество ячеек.
11	Учет стоимости рекламы. Заказчики помещают рекламу в телеэфире в определенной передаче в определенный день. Информационные поля: Название передачи, Рейтинг, Стоимость минуты, Название фирмы-заказчика, Банковские реквизиты, Телефон, Контактное лицо, Вид рекламы, Дата, Длительность в минутах.
12	Учет выполненной работы. Цех выпускает детали, которые производятся путем выполнения рабочими нескольких операций. Ежедневно фиксируется выполненная работа. Информационные поля: Наименование операции, Признак сложности, Фамилия работника, Имя, Отчество, Адрес, Телефон, Стаж, Количество деталей, Дата.
13	Интернет-магазин. На сайте компании выставлены на продажу товары. В случае приобретения товаров на сумму выше некоторого значения клиент получает в дальнейшем скидку на каждую покупку в размере 2%. Информационные поля: Название товара, Количество на складе, Цена, Единица измерения, Фамилия клиента, Имя, Отчество, Адрес, Телефон, E-mail, Признак скидки, Количество заказанного товара, Дата продажи.
14	Списание оборудования. Информационные поля: Название оборудования, Тип оборудования, Дата поступления, Количество, Подразделение установки, Причина списания, Дата списания, Фамилия ответственного, Имя, Отчество, Должность, Дата приема на работу.

№ варианта	Исходная информация
15	Отдел кадров - Назначение на должность. По информации базы данных издаются приказы на зачисление и перевод сотрудников. Информационные поля: Фамилия сотрудника, Имя, Отчество, Дата рождения, Пол, Отдел, Дата назначения, Название должности, Льготы по должности, Требования к квалификации, Срок_контракта.
16	Отдел кадров - Отпуска. По информации базы данных издаются приказы на отпуска сотрудников. Информационные поля: Фамилия сотрудника, Имя, Отчество, Семейное положение, Дата рождения, Образование, Стаж, Отдел, Должность, Город, Улица, Номер дома, Квартира, Период отпуска, Вид отпуска, Количество дней отпуска.

Тест "Нормализация данных и создание базы данных"

[В начало практикума](#)

Лабораторная работа № 3. T-SQL – язык реляционной базы данных

Язык реляционной базы данных в **SQL Server** называется **Transact-SQL (T-SQL)**. Операторы языка делятся на несколько групп, основными из которых является язык определения данных (**Data Definition Language, DDL**) и язык манипулирования данными (**Data Manipulation Language, DML**).

Язык **DDL** содержит три обобщенных оператора: **CREATE объект** (создание объекта базы данных), **ALTER объект** (изменение характеристик объекта) и **DROP объект** (удаление существующего объекта). Эти операторы создают, изменяют и удаляют объекты базы данных, такие как сама база данных, таблицы, столбцы и индексы.

Язык **DML** содержит операторы, которые манипулируют данными, осуществляя выборку информации (**SELECT**), добавление (**INSERT**), удаление (**DELETE**) и изменение (**UPDATE**). При записи операторов можно использовать на клавиатуре любой регистр.

Задание	Краткие теоретические сведения
1. Удалить базу данных X_MyBASE , созданную с помощью команд Server Management Studio и вновь создать с помощью языка T-SQL.	<p>В этой и других лабораторных работах рассматриваются запросы на примере базы данных ПРОДАЖИ из лабораторной работы № 1 и базы данных UNIVER, описание которой приведено в Приложении.</p> <p>Чтобы создать скрипт на языке SQL надо в окне Редактор запросов, которое открывается после нажатия кнопки Создать запрос на панели инструментов, набрать операторы языка и нажать кнопку Выполнить или клавишу <F5>.</p> <p>Оператор CREATE предназначен для создания объектов БД. Например, создать базу данных с именем ПРОДАЖИ можно с помощью операторов:</p> <p style="padding-left: 40px;">USE master; CREATE database ПРОДАЖИ;</p> <p>Затем надо нажать кнопку Обновить (Refresh).</p>

2. Разработать сценарии для создания в базе данных X_MyBASE нужных таблиц.

Использовать ограничения целостности.

Установить связи между полями.

Просмотреть структуры таблиц с помощью команды **Проект (Design)** в контекстном меню таблиц.

Для сохранения скрипта используется **Файл / Сохранить как (File / Save As)**.
При создании таблиц используются различные ограничения. Ограничения, накладываемые на столбцы таблиц баз данных, предотвращают появление данных, не соответствующих предварительно заданным свойствам таблиц. Эти ограничения называются ограничениями целостности.

Ограничение	Действие ограничения целостности
data type тип данных	Предотвращает появление в столбце значений, не соответствующих типу данных
not null запрет значений null	Предотвращает появление в столбце значений null
default знач. по умолчанию	Устанавливает значение в столбце по умолчанию при выполнении операции INSERT
primary key первичный ключ	Предотвращает появление в столбце повторяющихся значений и пустого значения
foreign key внешний ключ	Устанавливает связь между таблицей со столбцом, имеющим свойство foreign key и таблицей, имеющей столбец со свойством primary key;
unique уникальное значение	Не допускает пустые и повторяющиеся значения, не может быть использовано для связи с полем другой таблицы
check проверка значений	Предотвращает появление в столбце значения, не удовлетворяющего логическому условию

Ниже записаны скрипты для создания таблиц с ограничениями целостности в базе данных **ПРОДАЖИ** и установления связей между полями таблиц. Здесь связи определяются при создании таблицы **Заказы**.

```
use ПРОДАЖИ
CREATE table ТОВАРЫ
( Наименование nvarchar(50) primary key,
  Цена real unique not null,
  Количество int
);
```

Создание таблицы Заказчики:

```
CREATE TABLE Заказчики
( Наименование_фирмы nvarchar(20) primary key,
  Адрес nvarchar(50),
  Расчетный_счет nvarchar(20)
);
```

Создание таблицы Заказы:

```
CREATE TABLE Заказы
( Номер_заказа int primary key
  Наименование_товара nvarchar(20) foreign key references
    Товары (Наименование),
  Цена_продажи real,
  Количество int,
  Дата_поставки date,
  Заказчик nvarchar(20) foreign key references
    Заказчики(Наименование_фирмы)
)
```

3. Опробовать процедуру внесения изменения в структуру одной из таблиц с помощью оператора **ALTER** добавив столбец.

С использованием **ALTER** добавить некоторые ограничения целостности.

Просмотреть новую структуру и удалить добавленный столбец.

Для модификации существующих объектов базы данных или сервера применяется оператор **ALTER**.

Добавление столбца в таблицу:

ALTER Table Товары ADD Дата_поступления date;

Этот оператор можно записать в тот же скрипт, но выполнить следует, не повторяя предыдущих операторов, для чего оператор надо выделить и нажать **Выполнить (Execute)**.

Можно использовать клавишу **F5**.

Обновить базу данных с помощью кнопки **Обновить (Refresh)**.

Если значение атрибута равно по умолчанию некоторому сочетанию символов, то используется ограничение **Default** и через пробел в одинарных кавычках нужные символы.

Для атрибута может быть введено ограничение **check**. Например, ограничения добавляются с помощью **ALTER**:

ALTER Table STUDENT ADD POL nchar(1) default 'м' check (POL in ('м', 'ж'));

Атрибут **POL** может принимать только значения **м** или **ж**.

Для удаления объектов сервера или БД предназначен оператор **DROP**.

Удаление столбца:

ALTER Table Товары DROP Column Дата_поступления;

Удаление таблицы **ТОВАРЫ**: **DROP table ТОВАРЫ**

<p>4. С помощью оператора INSERT заполнить все таблицы информацией.</p>	<p>Оператор INSERT используется для ввода информации в таблицу.</p> <pre>INSERT into Товары (Наименование, Цена, Количество) Values ('Стол', 25.5, 4), ('Стул', 15, 4); INSERT into Товары (Наименование, Цена, Количество) values ('Полка', 20.0, 10) INSERT into Заказы (Номер_заказа, Наименование_товара, Цена_продажи, Количество, Дата_поставки, Заказчик) values (10,'Полка', 22.5, 1, '2014-5-7', 'Белвест'), (11,'Диван', 152, 3, '2014-5-8', 'Zte') INSERT into Заказы values (12,'Диван', 252, 1, '2014-6-7', 'Луч')</pre>
<p>5. Вывести все строки и столбцы одной из таблиц. Написать оператор SELECT, выбирающий все строки для двух столбцов таблицы. Подсчитать количество строк в таблице. Опробовать процедуру внесения изменения в содержимое одной из таблиц с помощью оператора UPDATE. Проверить результат, используя Select.</p>	<p>SELECT – основной оператор для поиска информации в базе данных. Чтобы вывести всю информацию из таблицы Товары, можно записать:</p> <pre>SELECT * From Товары;</pre> <p>Для выбора содержимого двух столбцов (Наименование, Цена) служит оператор:</p> <pre>SELECT Наименование, Цена From Товары;</pre> <p>Чтобы подсчитать количество строк в таблице можно использовать следующую форму оператора: SELECT count(*) From Товары;</p> <p>Вывод наименований товаров, цена которых меньше 200, при этом столбец результатов озаглавлен Дешевые товары:</p>

**SELECT Наименование [Дешевые товары] FROM Товары
Where Цена < 200**

Если наименование поля содержит символ пробела, то оно заключается в квадратные скобки.

Для изменения строк таблицы используется оператор **UPDATE**.

Например, в примере, приведенном ниже, содержимое столбца **Количество** заменяется на число 1:

UPDATE ТОВАРЫ set Количество = 1;

Содержимое столбца **Цена** увеличивается на 1 для товара, наименование которого – **Стол**:

UPDATE ТОВАРЫ set Цена = Цена+1 Where Наименование = 'Стол';

Чтобы удалить одну или несколько строк используется оператор **DELETE**. В примере, приведенном ниже, удаляется строка из таблицы **Товары**, в которой значение столбца **Наименование** равно **Стул**.

DELETE from ТОВАРЫ Where Наименование = 'Стул';

6. Внести изменения в сценарий создания базы данных **X_MyBASE** с тем, чтобы файлы размещались в определенных местах памяти.

БД представляет собой набор файлов операционной системы трех типов: первичный файл (расширение **mdf**), вторичные файлы (**ndf**) и файлы журнала транзакций (**log**). Все файлы БД, кроме файлов журнала транзакций, распределены по файловым группам. Файловые группы – это поименованный набор файлов БД.

```
use master
go
create database UNIVER
on primary
( name = N'UNIVER_mdf', filename = N'D:\BD\UNIVER_mdf.mdf',
  size = 10240Kb, maxsize=UNLIMITED, filegrowth=1024Kb)
log on
( name = N'UNIVER_log', filename=N'D:\BD\UNIVER_log.ldf',
  size=10240Kb, maxsize=2048Gb, filegrowth=10%)
go
```

Файловая группа, называемая первичной, является обязательной. Для ее обозначения используются ключевые слова **ON PRIMARY**.

В примере создается база **UNIVER**, файл **UNIVER_mdf.mdf** с первоначальным размером 10240Kb, максимальным размером неограниченным, приращением 1024Kb. Файл располагается на диске **D** в папке **BD** в первичной файловой группе.

Журнал транзакций в операторе **CREATE DATABASE** описывается отдельно в секции, обозначенной ключевыми словами **LOG ON**.

Пример с использованием вторичной файловой группы:

```
use master
create database UNIVER on primary
( name = N'UNIVER_mdf', filename = N'D:\BD\UNIVER_mdf.mdf',
```

	<pre> size = 10240Kb, maxsize=UNLIMITED, filegrowth=1024Kb), (name = N'UNIVER_ndf', filename = N'D:\BD\UNIVER_ndf.ndf', size = 10240KB, maxsize=1Gb, filegrowth=25%), filegroup FG1 (name = N'UNIVER_fg1_1', filename = N'D:\BD\UNIVER_fgq-1.ndf', size = 10240Kb, maxsize=1Gb, filegrowth=25%), (name = N'UNIVER_fg1_2', filename = N'D:\BD\UNIVER_fgq-2.ndf', size = 10240Kb, maxsize=1Gb, filegrowth=25%) log on (name = N'UNIVER_log', filename=N'D:\BD\UNIVER_log.ldf', size=10240Kb, maxsize=2048Gb, filegrowth=10%) </pre>
7. Разместить таблицы базы данных X_MyBASE в файловых группах.	<p>Пример создания таблицы AUDITORIUM, предназначенной для хранения данных об аудиторном фонде вуза, в файловой группе FG1:</p> <pre> CREATE TABLE AUDITORIUM (AUDITORIUM char(20) primary key, AUDITORIUM_TYPE char(10) foreign key references AUDITORIUM_TYPE(AUDITORIUM_TYPE), AUDITORIUM_CAPACITY int default 1 check (AUDITORIUM_CAPACITY between 1 and 300), AUDITORIUM_NAME varchar(50)) on FG1; </pre>

8. Запустить утилиту **SQLCMD** с параметром, позволяющим вывести в окно консоли краткую инструкцию о применении утилиты.

Команды **Transact-SQL** можно вводить и выполнять с помощью утилиты **SQLCMD**. Для этого надо выбрать **Пуск / Выполнить (Командная строка)**.

В поле ввода появившегося окна следует ввести **cmd**, затем команду **sqlcmd** и можно вводить операторы языка. Выход из данного режима – команда **exit**.

9. С помощью **SSMS** просмотреть диаграмму базы данных **X_MyBASE** и убедиться, что все требуемые внешние ключи (**foreign key**) отражены на диаграмме.

Опробовать различные операторы языка T-SQL на примере базы данных **X_MyBASE**.

Тест "Ознакомление с языком T-SQL"

[В начало практикума](#)

Лабораторная работа № 4. Многотабличные SELECT-запросы

Задание	Краткие теоретические сведения
<p>1. Ознакомиться с приложением. Изучить сценарии на языке T-SQL, содержащие операторы для создания и заполнения таблиц базы данных UNIVER.</p> <p>На основе таблиц AUDITORIUM_TYPE и AUDITORIUM сформировать перечень кодов аудиторий и соответствующих им наименований типов аудиторий.</p> <p>Использовать соединение таблиц INNER JOIN.</p>	<p>Соединение таблиц INNER JOIN (внутреннее соединение) наиболее часто используемый вид соединения реляционных таблиц.</p> <p>На основании таблиц Товары и Заказы сформировать перечень товаров с ценой исходной и ценой продажи:</p> <pre>SELECT Товары.Наименование, Товары.Цена, Заказы.Цена_продажи From Заказы Inner Join Товары On Заказы.[Наименование товара]= Товары.Наименование</pre>
<p>2. На основе таблиц AUDITORIUM_TYPE и AUDITORIUM сформировать перечень кодов аудиторий и соответствующих им наименований типов аудиторий, выбрав только те аудитории, в наименовании которых присутствует подстрока компьютер. Использовать соединение таблиц INNER JOIN и предикат</p>	<p>Результирующий набор создается следующим образом: выполняется декартово произведение (каждая строка одной таблицы соединяется с каждой строкой другой) для таблиц Товары и Заказы; из полученного результата выбираются строки, удовлетворяющие указанному условию; из всех столбцов результирующего набора выбираются столбцы, указанные в списке SELECT.</p> <p>На основании таблиц Товары и Заказы сформировать перечень товаров с ценой исходной и ценой продажи, при этом выбрать товары, которые содержат буквосочетание ‘ст’:</p> <pre>SELECT Заказы.[Наименование товара], Товары.Цена, Заказы.Цена_продажи From Заказы Inner Join Товары On Заказы.Наименование_товара = Товары.Наименование And Заказы.Наименование_товара Like '%ст%'</pre>

LIKE.

Перечень товаров с ценой исходной и ценой продажи на основании таблиц **Товары** и **Заказы** без применения INNER JOIN можно получить с помощью запроса:

```
SELECT Товары.Наименование, Товары.Цена, Заказы.Цена_продажи  
From Заказы, Товары  
Where Заказы.Наименование_товара = Товары.Наименование
```

В таком запросе можно использовать псевдонимы (T1 и T2):

```
SELECT T1.Наименование, T1.Цена, T2.Цена_продажи  
From Заказы As T2, Товары AS T1  
Where T2.Наименование_товара = T1.Наименование
```

3. На основе таблиц **PRORGESS**, **STUDENT**, **GROUPS**, **SUBJECT**, **PULPIT** и **FACULTY** сформировать перечень студентов, получивших экзаменационные оценки от 6 до 8. Результирующий набор должен содержать столбцы: **Факультет**, **Кафедра**, **Специальность**, **Дисциплина**, **Имя Студента**, **Оценка**. В столбце **Оценка** должны быть записаны экзаменационные оценки прописью: **шесть, семь, восемь**.

Результат отсортировать в порядке убывания по столбцу **PROGRESS.NOTE**.

Использовать соединение INNER JOIN,

На основании таблиц **Товары** и **Заказы** получить перечень товаров, дату поставки и сформировать столбец **Пределы цен** можно с помощью запроса, записанного ниже.

Здесь в выражении CASE каждое предложение WHEN содержит логическое выражение. Эти выражения проверяются на истинность сверху вниз, и при первом успешном сравнении формируется результирующее значение, указанное за ключевым словом THEN. В том случае, если ни одно из логических WHEN-выражений не принимает истинного значения, в качестве результата CASE формируется значение, указанное в предложении ELSE.

Результирующий набор содержит три столбца, наименования которых: **Наименование**, **Дата_поставки**, **Пределы цен**.

В последнем столбце выводятся тексты из выражения CASE. Реализована сортировка по полю **Заказы.Наименование_товара** в порядке возрастания.

предикат BETWEEN и выражение CASE.

```
SELECT Товары.Наименование, Заказы.Дата_поставки,  
Case  
when (Заказы.Цена_продажи between 1 and 50) then 'цена<50'  
when (Заказы.Цена_продажи between 50 and 100) then 'цена от 50 до 100'  
else 'цена больше 100'  
end [Пределы цен]  
FROM Товары INNER JOIN Заказы  
ON Товары.Наименование = Заказы.Наименование_товара  
ORDER BY Заказы.Наименование_товара
```

4. На основе таблиц **PULPIT** и **TEACHER** получить полный перечень кафедр и преподавателей на этих кафедрах.

Результирующий набор должен содержать два столбца: **Кафедра** и **Преподаватель**. Если на кафедре нет преподавателей, то в столбце **Преподаватель** должна быть выведена строка ***.

Примечание: использовать соединение таблиц LEFT OUTER JOIN и функцию **isnull**.

Левое внешнее соединение LEFT OUTER JOIN включает в набор несоединенные строки таблицы, имя которой записано слева от ключевых слов LEFT OUTER JOIN, а правое внешнее соединение RIGHT OUTER JOIN – несоединенные строки таблицы, имя которой записано справа.

Пусть требуется получить список заказанных товаров с указанием их количества на складе. Если наименование товара в таблице **Заказы** пропущено (в поле находится NULL), то в соответствующем поле столбца «Товар» вывести ***.

```
SELECT isnull(Заказы.Наименование_товара, '***') [Товар],  
Товары.Количество  
FROM Товары Left Outer JOIN Заказы  
ON Товары.Наименование = Заказы.Наименование_товара
```

Встроенная функция **isnull** принимает два параметра и проверяет их значения

	<p>на NULL слева направо. Функция возвращает первое значение, не равное NULL</p>
<p>5. Создав две таблицы показать на примере, что соединение FULL OUTER JOIN двух таблиц является коммутативной операцией.</p> <p>Создать три новых запроса:</p> <ul style="list-style-type: none"> – запрос, результат которого содержит данные левой (в операции FULL OUTER JOIN) таблицы и не содержит данные правой; – запрос, результат которого содержит данные правой таблицы и не содержащие данные левой; – запрос, результат которого содержит данные правой таблицы и левой таблиц; <p>Использовать в запросах выражение IS NULL и IS NOT NULL.</p>	<p>Операция является <i>коммутативной</i>, если формируемый результирующий набор не зависит от порядка, в котором указаны исходные таблицы.</p> <p>Рассмотрим два запроса для таблиц Товары и Заказы. В первом запросе выводятся значения, полученные в результате полного внешнего объединения таблиц:</p> <pre>SELECT * FROM Товары at FULL OUTER JOIN Заказы aa ON aa.Наименование_товара = at.Наименование ORDER BY aa.Наименование_товара, at.Наименование</pre> <p>Во втором запросе выводится количество значений, которые не смогли соединиться:</p> <pre>SELECT COUNT(*) from Товары at FULL OUTER JOIN Заказы aa ON aa.Наименование_товара = at.Наименование Where Номер_заказа is NULL</pre>
<p>6. Разработать SELECT-запрос на основе CROSS JOIN-соединения таблиц AUDITORIUM_TYPE и AUDITORIUM, формирующего результат, аналогичный результату запроса в задании 1.</p>	<p>При использовании соединения CROSS JOIN каждая строка одной таблицы соединяется с каждой строкой другой таблицы.</p> <p>Например, на основании таблиц Товары и Заказы сформировать перечень товаров с ценой исходной и ценой продажи можно с помощью запроса:</p>

```
SELECT Товары.Наименование, Товары.Цена, Заказы.Цена_продажи  
From Заказы Cross Join Товары  
Where Заказы.Наименование_товара = Товары.Наименование
```

7. Разработать и выполнить аналогичные запросы для базы данных X_MyBASE.

8*. Создать таблицу **TIMETABLE** (Группа, аудитория, предмет, преподаватель, день недели, пара), установить связи с другими таблицами, заполнить данными. Написать запросы на наличие свободных аудиторий на определенную пару, на определенный день недели, наличие «окон» у преподавателей и в группах.

Тест "Многотабличные SELECT-запросы"

[В начало практикума](#)

Лабораторная работа № 5. Использование подзапросов

Подзапрос – это SELECT-запрос, который выполняется в рамках другого запроса. Подзапросы могут применяться в секции WHERE. Подзапросы бывают двух видов: коррелируемые и независимые.

Коррелируемый подзапрос зависит от внешнего запроса и выполняется для каждой строки результирующего набора.

Независимый подзапрос не зависит от внешнего запроса и выполняется только один раз, но результат его выполнения подставляется в каждую строку результирующего набора.

Задание	Краткие теоретические сведения
<p>1. На основе таблиц FACULTY, PUL-PIT и PROFESSION сформировать список наименований кафедр, которые находятся на факультете, обеспечивающем подготовку по специальности, в наименовании которой содержится слово технология или технологии.</p> <p>Использовать в секции WHERE предикат IN с некоррелированным подзапросом к таблице PROFESSION.</p>	<p>Операция IN формирует логическое значение «истина» в том случае, если значение, указанное слева от ключевого слова IN, равно хотя бы одному из значений списка, указанного справа.</p> <p>Пусть требуется определить список, даты поставки и исходные цены тех товаров, в адресе доставки которых есть слово Минск.</p> <p>Запросы ниже дают один и тот же результат:</p> <pre>SELECT Заказы.Наименование_товара, Заказы.Дата_поставки, Товары.Цена FROM Заказы, Товары Where Заказы.Наименование_товара = Товары.Наименование and Заказчик In (Select Наименование_фирмы FROM Заказчики Where (Адрес Like 'Минск%'))</pre>

2. Переписать запрос пункта 1 таким образом, чтобы тот же подзапрос был записан в конструкции INNER JOIN секции FROM внешнего запроса. При этом результат выполнения запроса должен быть аналогичным результату исходного запроса.

3. Переписать запрос, реализующий 1 пункт без использования подзапроса. Примечание: использовать соединение INNER JOIN трех таблиц.

4. На основе таблицы **AUDITORIUM** сформировать список аудиторий самых больших вместимостей для каждого типа аудитории. При этом результат следует отсортировать в порядке убывания вместимости. Примечание: использовать коррелируемый подзапрос с секциями TOP и ORDER BY.

```
SELECT Заказы.Наименование_товара, Заказы.Дата_поставки,  
Товары.Цена  
FROM Заказы Inner JOIN Товары  
ON Заказы.Наименование_товара = Товары.Наименование  
Where Заказчик In (Select Наименование_фирмы from Заказчики  
Where (Адрес Like 'Минск%'))  
  
SELECT Заказы.Наименование_товара, Заказы.Дата_поставки,  
Товары.Цена  
FROM Заказы Inner join Товары  
ON Заказы.Наименование_товара = Товары.Наименование  
Inner join Заказчики  
ON Заказчики.Наименование_фирмы = Заказы.Заказчик  
Where (Адрес Like 'Минск%')
```

На основании таблицы **Заказы** сформировать перечень товаров, для которых их цены продажи являются максимальными. Здесь одной и той же таблице присваиваются разные псевдонимы:

```
SELECT Наименование_товара, Цена_продажи  
FROM Заказы а  
where Заказчик = (select top(1) Заказчик from Заказы аа  
                 where аа.Наименование_товара = а.Наименование_товара  
                 order by Цена_продажи desc)
```

Операция EXISTS формирует значение «истина», если результирующий набор подзапроса содержит хотя бы одну строку, в противоположном случае – значение «ложь».

5. На основе таблиц **FACULTY** и **PULPIT** сформировать список наименований факультетов на котором нет ни одной кафедры (таблица **PULPIT**).

Использовать предикат **EXISTS** и коррелированный подзапрос.

6. На основе таблицы **PROGRESS** сформировать строку, содержащую средние значения оценок (столбец **NOTE**) по дисциплинам, имеющим следующие коды: **ОАиП**, **БД** и **СУБД**. Примечание: использовать три некоррелированных подзапроса в списке **SELECT**; в подзапросах применить агрегатные функции **AVG**.

7. Разработать **SELECT**-запрос, демонстрирующий способ применения **ALL** совместно с подзапросом.

На основании таблиц **Товары** и **Заказы** сформировать перечень товаров из таблицы **Товары**, которые не заказаны покупателями:

```
SELECT Наименование from Товары  
Where not exists (select * from Заказы  
Where Заказы.Наименование товара = Товары.Наименование)
```

Определить среднее значение цены продажи столов и стульев из таблицы **Заказы**:

```
SELECT top 1  
(select avg(Цена_продажи) from Заказы  
where Наименование_товара like 'стол' ) [Стол],  
(select avg(Цена_продажи) from Заказы  
where Наименование_товара like 'стул' ) [Стул]  
From Заказы
```

Операция **>= ALL** формирует истинное значение в том случае, когда значение стоящее слева больше или равно каждому значению в списке, указанном справа.

Пусть надо определить наименования товаров, цены продажи которых превышают или равны значениям цен товаров, наименования которых начинаются на букву ‘с’. Запрос может иметь вид:

```
SELECT Наименование_товара, Цена_продажи from Заказы  
Where Цена_продажи >=all (select Цена_продажи from Заказы  
where Наименование_товара like 'с%')
```

8. Разработать SELECT-запрос, демонстрирующий принцип применения ANY совместно с подзапросом.

Операция \geq ANY формирует истинное значение в том случае, если значение стоящее слева, больше или равно хотя бы одному значению в списке, указанном справа.

Определить наименования товаров, цены продажи которых превышают хотя бы одно значение цены продажи товаров, наименования которых начинаются на букву 'с':

```
SELECT Наименование_товара, Цена_продажи from Заказы  
Where Цена_продажи  $\geq$  any (select Цена_продажи from Заказы  
                 where Наименование_товара like 'с%')
```

9. Разработать и выполнить аналогичные запросы для базы данных X_MyBASE.

10*. Найти в таблице STUDENT студентов, у которых день рождения в один и тот же день. Объяснить решение.

Тест "Использование подзапросов"

[В начало практикума](#)

Лабораторная работа № 6. Группировка данных

Основное назначение **группировки** с помощью секции GROUP BY – разбиение множества строк, сформированных секциями FROM и WHERE, на группы в соответствии со значениями в заданных столбцах, а также выполнение вычислений над группами строк с помощью наиболее часто используемых функций: **AVG** (вычисление среднего значения), **COUNT** (вычисление количества строк), **MAX** (вычисление максимального значения), **MIN** (вычисление минимального значения), **SUM** (вычисление суммы значений).

При использовании секции **GROUP BY** в SELECT-списке допускается указывать **только** те столбцы, по которым осуществляется группировка.

Задание	Краткие теоретические сведения
<p>1. На основе таблиц AUDITORIUM и AUDITORIUM_TYPE разработать запрос, вычисляющий для каждого типа аудиторий максимальную, минимальную, среднюю вместимость аудиторий, суммарную вместимость всех аудиторий и общее количество аудиторий данного типа.</p> <p>Результирующий набор должен содержать столбец с наименованием типа аудиторий и столбцы с вычисленными величинами.</p> <p>Использовать внутреннее соединение таблиц, секцию GROUP BY и агрегатные функции.</p>	<p>Определить наименования и максимальные цены товаров при продаже, количество заказанных товаров для тех товаров, количество которых на складе больше 5:</p> <pre>SELECT Наименование_товара, max(Цена_продажи) [Максимальная цена], count(*) [Количество заказанных товаров] From Заказы Inner Join Товары On Заказы.Наименование_товара = Товары.Наименование And Товары.Количество >5 Group by Наименование_товара</pre>

2. Разработать запрос на основе таблицы **PROGRESS**, который будет содержать значения экзаменационных оценок и их количество в заданном интервале.

Сортировка строк должна осуществляться в порядке, обратном величине оценки.

Использовать подзапрос в секции FROM, в подзапросе применить GROUP BY и CASE.

Определить пределы изменения цен и соответствующее количество товаров при продаже:

```
SELECT *
FROM (select Case when Цена_продажи between 1 and 50 then 'цена<50'
      when Цена_продажи between 50 and 100 then 'цена от 50 до 100'
      else 'цена больше 100'
    end [Пределы цен], COUNT (*) [Количество]
  FROM Заказы Group by Case
      when Цена_продажи between 1 and 50 then 'цена<50'
      when Цена_продажи between 50 and 100 then 'цена от 50 до 100'
      else 'цена больше 100'
    end ) as T
  ORDER BY Case [Пределы цен]
  when 'цена<50' then 3
  when 'цена от 50 до 100' then 2
  when 'цена больше 100' then 1
  else 0
end
```

3. Разработать SELECT-запроса на основе таблиц **FACULTY**, **GROUPS**, **STUDENT** и **PROGRESS**, который содержит среднюю экзаменационную оценку для каждого курса каждой специальности и факультета.

Строки отсортировать в порядке убывания средней оценки.

Пусть требуется определить наименования товаров, адреса фирм, заказавших эти товары, исходные цены и средние значения цен данных товаров при продаже. При этом исходная цена должна быть больше 50.

Функция **CAST** используется для преобразования типов, функция **ROUND** обеспечивает расчет значений с точностью до двух знаков после запятой.

Средняя оценка должна рассчитываться с точностью до двух знаков после запятой.

Использовать внутреннее соединение таблиц, агрегатную функцию AVG и встроенные функции CAST и ROUND.

4. Переписать SELECT-запрос, разработанный в задании 3, так чтобы в расчете среднего значения оценок использовались оценки только по дисциплинам с кодами **БД** и **ОАиП**. Использовать WHERE.

5. На основе таблиц **FACULTY**, **GROUPS**, **STUDENT** и **PROGRESS** разработать запрос, в котором выводятся специальность, дисциплины и средние оценки при сдаче экзаменов на факультете ТОВ.

6. На основе таблицы **PROGRESS** определить для каждой дисциплины количество студентов, получивших оценки

```
SELECT g.Наименование_товара,  
       s.Наименование_фирмы,  
       f.Цена,  
       round(avg(cast(g.Цена_продажи as float(4))),2)  
From Товары f inner join Заказы g  
      on f.Наименование = g.Наименование_товара  
      inner join Заказчики s  
      on g.Заказчик = s.Наименование_фирмы  
WHERE f.Цена >50  
GROUP BY g.Наименование_товара,  
           s.Наименование_фирмы,  
           f.Цена
```

Пусть требуется определить цены продаж и количество столов и стульев:

```
SELECT Наименование_товара, Цена_продажи, SUM(Количество)  
Количество  
FROM Заказы  
WHERE Наименование_товара IN ('стол', 'стул')  
GROUP BY Наименование_товара, Цена_продажи;
```

Логическое выражение, указанное в секции HAVING, вычисляется для каждой строки результирующего набора, сформированного секцией GROUP BY. Как и в случае с секцией WHERE строка отбирается в результирующий набор, если

8 и 9.

Использовать группировку, секцию HAVING, сортировку.

логическое выражение принимает значение «истина».

Наименования заказанных товаров, цена которых меньше 50 или больше 100, и их количество (таблица Заказы):

```
SELECT p1.Наименование_товара, p1.Цена_продажи,
(select COUNT(*) from Заказы p2
 WHERE p2.Наименование_товара = p1.Наименование_товара
 and p2.Цена_продажи = p1.Цена_продажи) [Количество]
FROM Заказы p1
GROUP BY p1.Наименование_товара, p1.Цена_продажи
HAVING Цена_продажи <50 or Цена_продажи >100
```

7. Разработать и выполнить аналогичные запросы для базы данных X_MyBASE.

Контрольная работа¹ 1

[В начало практикума](#)

Лабораторная работа № 7. Анализ данных и комбинирование результатов запросов

Аналитическими запросами к базе данных принято называть SELECT-запросы, сводные (агрегатные) результаты которых вычисляются над данными, хранящимися в таблицах базы данных. К ним относятся запросы, содержащие конструкции **ROLLUP** и **CUBE**. Они используются в секции GROUP BY и служат для вычисления значений агрегатных функций для подмножеств строк.

Для комбинирования результатов Select-запросов используются операторы UNION, UNION ALL, INTERSECT и EXCEPT.

Задание	Краткие теоретические сведения
<p>1. На основе таблиц FACULTY, GROUPS, STUDENT и PROGRESS разработать SELECT-запрос, в котором выводятся специальность, дисциплины и средние оценки при сдаче экзаменов на факультете ТОВ.</p> <p>Использовать группировку по полям FACULTY, PROFESSION, SUBJECT.</p> <p>Добавить в запрос конструкцию ROLLUP и проанализировать результат.</p>	<p>Пусть требуется определить цены продаж и количество продаваемых столов и стульев:</p> <pre>SELECT Наименование_товара, Цена_продажи, SUM(Количество) Количество FROM Заказы WHERE Наименование_товара IN ('стол', 'стул') GROUP BY Наименование_товара, Цена_продажи;</pre> <p>ROLLUP возвращает комбинацию групп и итоговых строк, которая определена в порядке, в котором заданы группируемые столбцы:</p> <pre>SELECT Наименование_товара, Цена_продажи, SUM(Количество) Количество FROM Заказы WHERE Наименование_товара IN ('стол', 'стул') GROUP BY ROLLUP (Наименование_товара, Цена_продажи);</pre>

	<p>Правило формирования результирующего набора SELECT-запроса, применяющего конструкцию ROLLUP:</p> <ul style="list-style-type: none"> – результирующий набор содержит $n + 1$ групп строк, где n – количество выражений для группировки столбцов, указанных за словом ROLLUP; – первая группа строк является результатом выполнения группировки по всем n выражениям; – вторая группа строк является результатом группировки первой группы строк по $n - 1$ первым выражениям. Столбцы, по которым не выполнялась группировка, заполняются значениями NULL; – группа строк k является группировкой группы строк, полученной на предыдущем этапе по $n - k + 1$ первым выражениям; – последняя ($n + 1$)-я группа содержит одну строку. <p>Конструкция CUBE также используется в секции GROUP BY. Возвращает любую возможную комбинацию групп и итоговых строк.</p> <p>Правило формирования результирующего набора SELECT-запроса, применяющего конструкцию CUBE:</p> <ul style="list-style-type: none"> – формируется множество всех подмножеств выражений, указанных в CUBE-списке; – для каждого непустого подмножества выполняется группировка; – если количество элементов подмножества меньше количества элементов CUBE-списка, то соответствующие значения заполняются значением NULL; – для пустого подмножества выполняется группировка, аналогичная той, что выполнялась для ROLLUP.
--	---

3. На основе таблиц **GROUPS**, **STUDENT** и **PROGRESS** разработать SELECT-запрос, в котором определяются результаты сдачи экзаменов.

В запросе должны отражаться специальности, дисциплины, средние оценки студентов на факультете ТОВ.

Отдельно разработать запрос, в котором определяются результаты сдачи экзаменов на факультете ХТИТ.

Объединить результаты двух запросов с использованием операторов UNION и UNION ALL. Объяснить результаты.

4. Получить пересечение двух множеств строк, созданных в результате выполнения запросов пункта 3. Объяснить результат. Использовать оператор INTERSECT.

Оператор **UNION** выполняет теоретико-множественную операцию объединения результирующих наборов SELECT-запросов, в котором строки не могут повторяться.

Если требуется механическое объединение строк, можно применить оператор **UNION ALL**.

Получить информацию о заказанных товарах двух фирм и их количестве можно с помощью запроса:

```
SELECT Наименование_товара, sum(Количество) Количество
  FROM Заказы WHERE Заказчик='Луч'
 Group BY Наименование_товара
      UNION
SELECT Наименование_товара, sum(Количество) Количество
  FROM Заказы WHERE Заказчик='Белвест'
 Group BY Наименование_товара
```

Результатом оператора **INTERSECT** является набор строк, являющийся пересечением двух исходных результирующих наборов SELECT-запросов.

5. Получить разницу между множеством строк, созданных в результате запросов пункта 3. Объяснить результат. Использовать оператор EXCEPT.

Результатом оператора **EXCEPT** является набор строк, являющийся разностью двух исходных результирующих наборов SELECT-запросов (т.е. в результат включаются те строки, которые есть в первом запросе, но отсутствуют во втором).

6. Разработать и выполнить аналогичные запросы для базы данных **X_MyBASE**.

7*. Подсчитать количество студентов в каждой группе, на каждом факультете и всего в университете одним запросом. Подсчитать количество аудиторий по типам и суммарной вместимости в корпусах и всего одним запросом.

Тест "Группировка данных"

[В начало практикума](#)

Лабораторная работа № 8. Использование представлений

Представление (View) – это объект базы данных, представляющий собой *поименованный* SELECT-запрос, который хранится в базе данных. Представление создается с помощью оператора CREATE, удаляется с помощью оператора DROP и изменяется с помощью ALTER.

Задание	Краткие теоретические сведения
1. Разработать представление с именем Преподаватель . Представление должно быть построено на основе SELECT-запроса к таблице TEACHER и содержать следующие столбцы: код, имя преподавателя, пол, код кафедры .	<p>При создании представления к SELECT-запросу предъявляются следующие требования: секцию ORDER BY можно использовать только совместно с опцией TOP; не допускается применение секции INTO, COMPUTE и COMPUTE BY; все столбцы результирующего набора должны быть поименованы.</p> <p>Пусть требуется определить наименования заказанных товаров, цены их продаж и даты поставки:</p> <p>SELECT Наименование_товара [Товар], Цена_продажи [Цена продажи], Дата_поставки [Дата] from Заказы;</p> <p>Можно создать представление с именем Заказанные товары:</p> <p>CREATE VIEW [Заказанные товары] as select Наименование_товара [Товар], Цена_продажи [Цена продажи], Дата_поставки [Дата] from Заказы;</p> <p>Тогда тот же самый первоначальный запрос записывается короче:</p> <p>SELECT * from [Заказанные товары]</p>

	<p>Это представление можно использовать и в других запросах, например:</p> <pre>SELECT * from [Заказанные товары] order by [Дата];</pre> <p>Чтобы изменить представление, надо использовать оператор ALTER:</p> <pre>ALTER VIEW [Заказанные товары] as select Наименование_товара [Товар], Цена_продажи [Цена продажи], Дата_поставки [Дата], Количество [Количество] FROM Заказы;</pre> <p>Удаляется представление с помощью оператора DROP:</p> <pre>DROP VIEW [Заказанные товары];</pre>
<p>2. Разработать и создать представление с именем Количество кафедр. Представление должно быть построено на основе SELECT-запроса к таблицам FACULTY и PULPIT.</p> <p>Представление должно содержать следующие столбцы: факультет, количество кафедр (вычисляется на основе строк таблицы PULPIT).</p>	<p>Представление Сравнение цен выводит информацию об исходных ценах и ценах продажи тех товаров, которые заказаны клиентами, т. е. содержатся в таблице Заказы:</p> <pre>CREATE VIEW [Сравнение цен] as SELECT zk.Наименование_товара [Товар], tv.Цена [Исходная цена], zk.Цена_продажи [Цена продажи] FROM Заказы zk join Товары tv ON zk.Наименование_товара = tv.Наименование;</pre> <p>При создании представлений, позволяющих выполнять операции INSERT,</p>

	<p>DELETE и UPDATE, базовый SELECT-запрос должен удовлетворять правилам:</p> <ul style="list-style-type: none"> – запрос не должен содержать секцию группировки GROUP BY; – запрос не должен применять агрегатные функции, опции DISTINCT и TOP, операторы UNION, INTERSECT и EXCEPT; – в SELECT-списке запроса не должно быть вычисляемых значений; – в секции FROM запроса должна указываться только одна таблица. <p>В приведенном выше примере представление не удовлетворяет одному из этих правил.</p>
<p>3. Разработать и создать представление с именем Аудитории. Представление должно быть построено на основе таблицы AUDITORIUM и содержать столбцы: код, наименование аудитории.</p> <p>Представление должно отображать только лекционные аудитории (в столбце AUDITORIUM_TYPE строка, начинающаяся с символа ЛК) и допускать выполнение оператора INSERT, UPDATE и DELETE.</p>	<p>Имя представления может содержать параметры, которые заключаются в скобки и отображаются в первой строке результирующего набора.</p> <pre>CREATE VIEW Дорогие_товары (Товар, Цена, Количество) as select Наименование, Цена, Количество from Товары where Цена>200; go SELECT * from Дорогие_товары</pre> <p>Операторы INSERT осуществляют вставку строк с новой информацией:</p> <pre>INSERT Дорогие_товары values('Диван', 300, 3) INSERT Дорогие_товары values('Шкаф', 150, 7)</pre>

4. Разработать и создать представление с именем **Лекционные_аудитории**.

Представление должно быть построено на основе SELECT-запроса к таблице **AUDITORIUM** и содержать следующие столбцы: **код, наименование аудитории.**

Представление должно отображать только лекционные аудитории (в столбце **AUDITORIUM_TYPE** строка, начинающаяся с символов ЛК).

Чтобы операция вставки не могла осуществляться в том случае, когда информация не удовлетворяет условию, записанному в секции Where, то следует создавать представление с опцией WITH CHECK OPTION. Выполнение INSERT и UPDATE допускается, но с учетом ограничения, задаваемого опцией WITH CHECK OPTION.

Например, можно изменить представление **Дорогие_товары**:

```
ALTER VIEW Дорогие_товары (Товар, Цена, Количество)
as select Наименование, Цена, Количество from Товары
      where Цена > 200 WITH CHECK OPTION;
go
```

Тогда оператор INSERT не выполнится, поскольку цена не удовлетворяет нужному условию секции Where.

```
INSERT Дорогие_товары values('Стол', 80, 9)
```

5. Разработать представление с именем **Дисциплины**. Представление должно быть построено на основе SELECT-запроса к таблице **SUBJECT**, отображать все дисциплины в алфавитном порядке и содержать следующие столбцы: **код, наименование дисциплины и код кафедры.**

Использовать TOP и ORDER BY.

Поскольку секцию ORDER BY можно использовать только совместно с опцией TOP, то представление может выглядеть следующим образом:

```
CREATE VIEW Дорогие_товары (Товар, Цена, Количество)
as select TOP 150 Наименование, Цена, Количество FROM Товары
      ORDER BY Наименование;
```

6. Изменить представление **Количество_кафедр**, созданное в задании 2 так, чтобы оно было привязано к базовым таблицам.

Продемонстрировать свойство привязанности представления к базовым таблицам.

Использовать опцию SCHEMABINDING.

Опция SCHEMABINDING устанавливает запрещение на операции с таблицами и представлениями, которые могут привести к нарушению работоспособности представления.

```
ALTER VIEW [Сравнение цен] WITH SCHEMABINDING
as SELECT zk.Наименование_товара [Товар],
              tv.Цена [Исходная цена],
              zk.Цена_продажи [Цена продажи]
FROM dbo.Заказы zk join dbo.Товары tv
ON zk.Наименование_товара = tv.Наименование;
```

При использовании опции SCHEMABINDING требуется использовать в SELECT-запросе для имен таблиц и представлений двухкомпонентный формат (в имени присутствует наименование схемы).

7. Разработать представления для базы данных **X_MyBASE**.

8*. Разработать представление для таблицы **TIMETABLE** (лабораторная работа № 4) в виде расписания. Изучить оператор PIVOT и использовать его.

Тест "Использование представлений"

[В начало практикума](#)

Лабораторная работа № 9. Основы программирования на T-SQL

Использование программ на языке T-SQL позволяет расширить круг решаемых задач, возникающих при работе с базами данных.

Задание	Краткие теоретические сведения
<p>1. Разработать скрипт, в котором:</p> <ul style="list-style-type: none">– объявить переменные типа char, varchar, datetime, time, int, smallint, tinyint, numeric(12, 5);– первые две переменные проинициализировать в операторе объявления;– присвоить произвольные значения переменным с помощью операторов SET и SELECT;– значения одних переменных вывести с помощью оператора SELECT, значения других переменных распечатать с помощью оператора PRINT. <p>Проанализировать результаты.</p>	<p>Для объявления переменных, используемых в программах, предназначен оператор DECLARE:</p> <pre>DECLARE @i int = 1, @b varchar(4) = 'БГТУ', @c datetime = getdate(); SELECT @i i, @b b, @c c DECLARE @h TABLE (num int identity(1, 1), fil varchar(30) default 'XXX'); INSERT @h default values; -- добавление строки в табличную переменную SELECT * from @h;</pre> <p>Имя переменной должно начинаться с символа @.</p> <p>С помощью оператора SET можно одной переменной присвоить значение и выполнять вычисления. Оператор SELECT позволяет нескольким переменным присвоить значения.</p> <p>Здесь @h – переменная типа TABLE. Этот тип позволяет создавать таблицы в памяти и использовать их для хранения промежуточных данных.</p>

2. Разработать скрипт, в котором определяется общая вместимость аудиторий.

Если общая вместимость превышает 200, то вывести количество аудиторий, среднюю вместимость аудиторий, количество аудиторий, вместимость которых меньше средней, и процент таких аудиторий.

Если общая вместимость аудиторий меньше 200, то вывести сообщение о размере общей вместимости.

Вывод данных в T-SQL возможен двумя способами: оператором SELECT можно сформировать выходной результирующий набор и с помощью оператора PRINT можно вывести строку в стандартный выходной поток.

Если одновременно выводятся данные, сформированные оператором SELECT и данные, сформированные оператором PRINT, то посмотреть последние можно на вкладке Messages.

```
DECLARE @d numeric(5,2) = 4.7, @a char(2), @f float(4)=1;
SET @a = 'РБ'; SET @f = 11.4+@f;
print 'd= '+cast(@d as varchar(10));
print 'a= '+cast(@a as varchar(10));
print 'f= '+cast(@f as varchar(10));
```

Функция CAST используется для преобразования типов.

```
DECLARE @Количество int = (select count(*) from Заказы)
print 'Количество : '+ cast (@Количество as varchar(10));
```

Пусть требуется определить общую сумму заказанных товаров, их количество, среднюю цену продажи, количество товаров, цены которых превышают среднюю и процент таких товаров в том случае, когда общая сумма превышает 1000.

В противном случае вывести сообщение о размере общей суммы:

```

DECLARE @y1 numeric(8, 3)=(select cast(sum(Цена_продажи)
    as numeric(8,3)) from Заказы), @y2 real, @y3 numeric(8, 3), @y4 real
IF @y1>1000
begin
    SELECT @y2 = (select cast( count(*) as numeric(8, 3)) from Заказы),
        @y3 = (select cast(AVG(Цена_продажи )
            as numeric(8,3)) from Заказы)
    SET @y4= (select cast(COUNT(*) as numeric(8, 3)) from Заказы
        where Цена_продажи > @y3)
    SELECT @y1 'Общая сумма', @y2 'Количество', @y3 'Средняя цена',
        @y4 'Количество товаров с ценой выше средней'
    end
else IF @y1>500 print 'Общая сумма от 500 до 1000'
else IF @y1>100 print 'Общая сумма от 100 до 500'
else print 'Общая сумма < 100'

```

3. Разработать T-SQL-скрипт, который выводит на печать глобальные переменные:

- @@ROWCOUNT (число обработанных строк);
- @@VERSION (версия SQL Server);
- @@SPID (возвращает системный идентификатор процесса, назначенный сервером текущему подключению);

MSS поддерживает широкий набор встроенных математических функций, которые можно применять в сценариях T-SQL для вычислений:

```

print 'Округление      : '+ cast(round(12345.12345, 2) as varchar(12));
print 'Нижнее целое   : '+ cast(floor(24.5) as varchar(12));
print 'Возведение в степень: '+ cast(power(12.0, 2) as varchar(12));
print 'Логарифм       : '+ cast(log(144.0) as varchar(12));
print 'Корень квадратный : '+ cast(sqrt(144.0) as varchar(12));
print 'Экспонента     : '+ cast(exp(4.96981) as varchar(12));
print 'Абсолютное значение : '+ cast(abs(-5) as varchar(12));

```

- @@ERROR (код последней ошибки);
 - @@SERVERNAME (имя сервера);
 - @@TRANCOUNT (возвращает уровень вложенности транзакции);
 - @@FETCH_STATUS (проверка результата считывания строк результирующего набора);
 - @@NESTLEVEL (уровень вложенности текущей процедуры).
- Проанализировать результат.

4. Разработать T-SQL-скрипты, выполняющие:

- вычисление значений переменной **Z**

$$z = \begin{cases} \sin^2(t), & t > x \\ 4 \cdot (t + x), & t < x \\ 1 - e^{x-2}, & t = x \end{cases}$$

для различных значений исходных данных;

```

print 'Синус      : '+ cast(sin(pi()) as varchar(12));
print 'Подстрока   : '+ substring('1234567890', 3,2);
print 'Удалить пробелы справа : '+ rtrim('12345   ')+'X';
print 'Удалить пробелы слева : '+ 'X'+ltrim('   67890');
print 'Нижний регистр   : '+ lower ('ВЕРХНИЙ РЕГИСТР');
print 'Верхний регистр   : '+ upper ('нижний регистр');
print 'Заменить       : '+ replace('1234512345', '5', 'X');
print 'Строка пробелов : '+ 'X'+ space(5) +'X';
print 'Повторить строку : '+ replicate('12', 5);
print 'Найти по шаблону : '+ cast (patindex ('%Y_Y%', '123456YxY7890') as
varchar(5));
DECLARE @t time(7) = sysdatetime(), @dt datetime = getdate();
print 'Текущее время   : '+ convert (varchar(12), @t);
print 'Текущая дата    : '+ convert (varchar(12), @dt, 103);
print '+1 день         : '+ convert(varchar(12), dateadd(d, 1, @dt), 103);

```

Функция CONVERT также используется для преобразования типов. Третий параметр этой функции применяется при преобразованиях, использующих типы данных для хранения даты и времени, и задает стиль представления этих данных.

Пусть требуется вычислить значение **y**:

$$x = \operatorname{tg}(a^2 + 1), \quad y = \begin{cases} 7a + x, & 3x < ab \\ \cos(a), & 3x \geq ab \end{cases}$$

– преобразование полного ФИО студента в сокращенное (например, Макейчик Татьяна Леонидовна в Макейчик Т. Л.);

– поиск студентов, у которых день рождения в следующем месяце, и определение их возраста;

– поиск дня недели, в который студенты некоторой группы сдавали экзамен по БД.

5. Продемонстрировать конструкцию IF... ELSE на примере анализа данных таблиц базы данных X_UNIVER.

6. Разработать сценарий, в котором с помощью CASE анализируются оценки, полученные студентами некоторого факультета при сдаче экзаменов.

```
DECLARE @a int = 1, @b float = 0.3, @x float, @y float;
SET @x = TAN(@a*@a+1);
IF (3*@x < @a*@b) SET @y = 7*@a+@x;
else
SET @y = cos(@a);
PRINT 'y= ' +cast(@y as varchar(10));
```

С помощью операторных скобок BEGIN END можно объединять операторы в группы. Пусть надо проанализировать количество заказанных товаров:

```
DECLARE @x int = (select count(*) FROM Заказы);
IF (select count(*) FROM Заказы)>20
begin
PRINT 'Количество товаров больше 20';
PRINT 'Количество = ' +cast(@x as varchar(10));
end;
begin
PRINT 'Количество товаров меньше 20';
PRINT 'Количество = ' +cast(@x as varchar(10));
end;
```

Выражение CASE, как правило, применяется в SELECT-списке, секциях WHERE, ORDER BY, HAVING и служит для формирования одного из нескольких возможных значений.

Например, для анализа цен товаров можно использовать следующие операторы:

```

SELECT CASE
    when Цена_продажи between 0 and 10 then 'дешево'
    when Цена_продажи between 10 and 100 then 'нормально'
    when Цена_продажи between 100 and 500 then 'дорого'
        else 'очень дорого'
    end Цена , count(*) [Количество]
FROM dbo.Заказы
GROUP BY CASE
    when Цена_продажи between 0 and 10 then 'дешево'
    when Цена_продажи between 10 and 100 then 'нормально'
    when Цена_продажи between 100 and 500 then 'дорого'
        else 'очень дорого'
    end

```

7. Создать временную локальную таблицу из трех столбцов и 10 строк, заполнить ее и вывести содержимое. Использовать оператор WHILE.

Основное отличие *временных* таблиц от постоянных в том, что они хранятся в системной базе данных TEMPDB и удаляются после окончания сеанса работы с базой.

Локальные временные таблицы имеют имена, начинающиеся с символа # и доступны только пользователю, ее создавшему.

Сформировать временную локальную таблицу можно с помощью оператора CREATE:

```

CREATE table #EXPLRE
( TIND int,
  TFIELD varchar(100)
);

```

Заполнить таблицу можно, например, с помощью следующего скрипта:

```
SET nocount on;      --не выводить сообщения о вводе строк
DECLARE @i int=0;
WHILE @i<1000
    begin
        INSERT #EXPLRE(TIND, TFIELD)
            values(floor(30000*rand()), replicate('строка', 10));
        IF(@i % 100 = 0)
            print @i;    --вывести сообщение
        SET @i = @i + 1;
    end;
```

Функция RAND генерирует случайное число.

Глобальные временные таблицы имеют имена, начинающиеся с символов ## и доступны всем пользователям, подключенным к серверу.

8. Разработать скрипт, демонстрирующий использование оператора RETURN.

Оператор RETURN служит для немедленного завершения работы пакета:

```
DECLARE @x int = 1
print @x+1
print @x+2
RETURN
print @x+3
```

9. Разработать сценарий с ошибками, в котором используются для обработки ошибок блоки TRY и CATCH.

Применить функции ER-
ROR_NUMBER (код последней ошибки),
ERROR_MESSAGE (сообщение об ошибке), ERROR_LINE
(номер строки с ошибкой), ER-
ROR_PROCEDURE (имя процедуры или NULL),
ERROR_SEVERITY (уровень серьезности ошибки),
ERROR_STATE (метка ошибки).
Проанализировать результат.

Для обработки ошибок выполнения в сценарии T-SQL предусмотрена конструкция, состоящая из двух блоков: TRY и CATCH. Блок TRY содержит код T-SQL, в котором могут возникнуть ошибки, а блок CATCH – код, предназначенный для обработки ошибок.

Ошибка, возникающая в охраняемом коде, приводит к передаче управления в блок обработки ошибок:

```
begin TRY
    UPDATE dbo.Заказы set Номер_заказа = '5'
        where Номер_заказа= '6'
end try
begin CATCH
    print ERROR_NUMBER()
    print ERROR_MESSAGE()
    print ERROR_LINE()
    print ERROR_PROCEDURE()
    print ERROR_SEVERITY()
    print ERROR_STATE()
end catch
```

Тест "Основы программирования на T-SQL"

[В начало практикума](#)

Лабораторная работа № 10. Создание и применение индексов

Индекс – это объект базы данных, позволяющий **ускорить поиск** в определенной таблице, так как при этом данные организуются в виде сбалансированного бинарного дерева поиска.

Как и любой другой объект базы данных, индекс может быть создан с помощью оператора CREATE, модифицирован с помощью ALTER и удален с помощью оператора DROP. Для одной таблицы возможно построение нескольких индексов.

Индексы бывают кластеризованные, некластеризованные, уникальные, неуникальные и др.

Задание	Краткие теоретические сведения
<p>1. Определить все индексы, которые имеются в БД UNIVER.</p> <p>Создать временную локальную таблицу. Заполнить ее данными (не менее 1000 строк).</p> <p>Разработать SELECT-запрос. Получить план запроса и определить его стоимость.</p> <p>Создать кластеризованный индекс, уменьшающий стоимость SELECT-запроса.</p>	<p>Обычно <i>кластеризованные</i> индексы создаются автоматически при создании таблицы если в ней присутствует первичный ключ (ограничение PRIMARY KEY).</p> <p>Кластеризованные индексы физически упорядочены в соответствии со значениями индексируемых столбцов. В таблице может быть только один кластеризованный индекс.</p> <p>С помощью системной процедуры SP_HELPINDEX можно получить перечень индексов, связанных с заданной таблицей:</p> <pre>use UNIVER exec SP_HELPINDEX 'AUDITORIUM_TYPE'</pre> <p>Пусть сформирована и заполнена временная локальная таблица с именем #EXPLRE (п.7 в лабораторной работе № 8).</p> <p>План запроса можно посмотреть, выполнив в контекстном меню запроса команду Display Estimated Execution Plan (Показать предполагаемый план выполнения) или нажав соответствующую кнопку на панели инструментов.</p> <pre>SELECT * FROM #EXPLRE where TIND between 1500 and 2500 order by TIND</pre>

	<p>Общая стоимость запроса (Estimated Subtree Cost) появляется во всплывающем окне, если подвести курсор к компоненту Table Scan (она равна 0,011). Чтобы объективно оценить время выполнения следующего запроса, надо очистить буферный кэш:</p> <p>checkpoint; --фиксация БД DBCC DROPCLEANBUFFERS; --очистить буферный кэш</p> <p>Если создать кластеризованный индекс:</p> <p>CREATE clustered index #EXPLRE_CL on #EXPLRE(TIND asc)</p> <p>и вновь выполнить запрос, то его стоимость станет равна 0,0033, т. е. уменьшится.</p>
<p>2. Создать временную локальную таблицу. Заполнить ее данными (10000 строк или больше).</p> <p>Разработать SELECT-запрос. Получить план запроса и определить его стоимость.</p> <p>Создать некластеризованный неуникальный составной индекс.</p> <p>Оценить процедуры поиска информации.</p>	<p><i>Некластеризованные</i> индексы не влияют на физический порядок строк в таблице. Пусть сформирована и заполнена временная локальная таблица #EX:</p> <pre>CREATE table #EX (TKEY int, CC int identity(1, 1), TF varchar(100)); ;</pre> <pre>set nocount on; declare @i int = 0; while @i < 20000 -- добавление в таблицу 20000 строк begin INSERT #EX(TKEY, TF) values(floor(30000*RAND()), replicate('строка ', 10)); set @i = @i + 1;</pre>

	<pre><code>end; SELECT count(*)[количество строк] from #EX; SELECT * from #EX</code></pre> <p>Можно создавать индексы по нескольким столбцам – такие индексы называются <i>составными</i>. Можно создать составной неуникальный, некластеризованный индекс #EX_NONCLU по двум столбцам TKEY и CC таблицы #EX с помощью оператора:</p> <p>CREATE index #EX_NONCLU on #EX(TKEY, CC)</p> <p>Этот индекс не применяется оптимизатором ни при фильтрации, ни при сортировке строк таблицы #EX, в чем можно убедиться, посмотрев планы следующих запросов:</p> <pre><code>SELECT * from #EX where TKEY > 1500 and CC < 4500; SELECT * from #EX order by TKEY, CC</code></pre> <p>Но, если хотя бы одно из индексируемых значений зафиксировать (задать одно значение), то оптимизатор применит индекс. Это можно проверить, выполнив запрос:</p> <pre><code>SELECT * from #EX where TKEY = 556 and CC > 3</code></pre>
<p>3. Создать временную локальную таблицу. Заполнить ее данными (не менее 10000 строк).</p> <p>Разработать SELECT-запрос. Получить план запроса и определить его стоимость.</p> <p>Создать некластеризованный индекс покрытия, уменьшающий</p>	<p><i>Некластеризованный индекс покрытия</i> запроса позволяет включить в состав индексной строки значения одного или нескольких неиндексируемых столбцов. Например, индекс покрытия #EX_TKEY_X включает значения столбца CC (ключевое слово INCLUDE):</p> <p>CREATE index #EX_TKEY_X on #EX(TKEY) INCLUDE (CC)</p> <p>Чтобы оценить процедуры поиска можно посмотреть планы выполнения запроса без применения индексов и с использованием индекса покрытия.</p>

стоимость SELECT-запроса.	<p>SELECT CC from #EX where TKEY>15000</p>
<p>4. Создать и заполнить временную локальную таблицу. Разработать SELECT-запрос, получить план запроса и определить его стоимость. Создать <i>некластеризованный фильтруемый индекс</i>, уменьшающий стоимость SELECT-запроса.</p>	<p>Если запросы основаны на WHERE-фильтрации строк, то может быть эффективным применение <i>фильтруемых некластеризованных индексов</i>. Пусть имеется три запроса.</p> <p>SELECT TKEY from #EX where TKEY between 5000 and 19999; SELECT TKEY from #EX where TKEY>15000 and TKEY < 20000 SELECT TKEY from #EX where TKEY=17000</p> <p>Надо оценить планы их выполнения. Затем можно создать фильтрующий индекс с именем, например, #EX_WHERE:</p> <p>CREATE index #EX_WHERE on #EX(TKEY) where (TKEY>=15000 and TKEY < 20000);</p> <p>Здесь фильтруемый индекс создается только для строк таблицы #EX, которые удовлетворяют логическому условию. Стоимость запросов уменьшится.</p>
<p>5. Заполнить временную локальную таблицу. Создать некластеризованный индекс. Оценить уровень фрагментации индекса. Разработать сценарий на T-SQL, выполнение которого приводит к уровню фрагментации индекса выше 90%. Оценить уровень фрагментации</p>	<p>Операции добавления и изменения строк базы данных могут повлечь образование неиспользуемых фрагментов в области памяти индекса. Процесс образования неиспользуемых фрагментов памяти называется <i>фрагментацией</i>. Фрагментация индексов снижает эффект от их применения. Пусть создан индекс:</p> <p>CREATE index #EX_TKEY ON #EX(TKEY);</p> <p>Получить информацию о степени фрагментации индекса можно с помощью операторов:</p> <p>SELECT name [Индекс], avg_fragmentation_in_percent [Фрагментация (%)] FROM sys.dm_db_index_physical_stats(DB_ID('TEMPDB'), OBJECT_ID('#EX'), NULL, NULL, NULL) ss JOIN sys.indexes ii on ss.object_id =</p>

<p>индекса.</p> <p>Выполнить процедуру <i>реорганизации</i> индекса, оценить уровень фрагментации.</p> <p>Выполнить процедуру <i>перестройки</i> индекса и оценить уровень фрагментации индекса.</p>	<p>ii.object_id and ss.index_id = ii.index_id WHERE name is not null;</p> <p>Если вставить 10000 строк с помощью</p> <p>INSERT top(10000) #EX(TKEY, TF) select TKEY, TF from #EX;</p> <p>и получить данные о фрагментации с помощью операторов, приведенных выше, то можно увидеть, что уровень фрагментации превысит 99%.</p> <p>Для избавления от фрагментации индекса предусмотрены две специальные операции: реорганизация и перестройка индекса.</p> <p><i>Реорганизация</i> (REORGANIZE) выполняется быстро, но после нее фрагментация будет убрана только на самом нижнем уровне.</p> <p>Пусть выполнена реорганизация с помощью оператора ALTER для индекса #EX_TKEY.</p> <p>ALTER index #EX_TKEY on #EX reorganize;</p> <p>Тогда выполнение соответствующего запроса покажет, что уровень фрагментации значительно снизился, но не до конца.</p> <p>Операция <i>перестройки</i> (REBUILD) затрагивает все узлы дерева, поэтому после ее выполнения степень фрагментации равна нулю.</p> <p>Пусть выполнена перестройка с помощью оператора ALTER для индекса #EX_TKEY в режиме OFFLINE.</p> <p>ALTER index #EX_TKEY on #EX rebuild with (online = off);</p> <p>Выполнением запроса о фрагментации можно оценить ее уровень.</p>
--	--

<p>6. Разработать пример, демонстрирующий применение параметра FILLFACTOR при создании некластеризованного индекса.</p>	<p>Уровнем фрагментации можно в некоторой степени управлять, если при создании или изменении индекса использовать параметры FILLFACTOR и PAD_INDEX. Параметр FILLFACTOR указывает процент заполнения индексных страниц нижнего уровня.</p> <p>Пусть индекс пересоздан со значением параметра FILLFACTOR равным 65:</p> <pre>DROP index #EX_TKEY on #EX; CREATE index #EX_TKEY on #EX(TKEY) with (fillfactor = 65);</pre> <p>После добавления строк в таблицу #EX можно оценить уровень фрагментации:</p> <pre>INSERT top(50)percent INTO #EX(TKEY, TF) SELECT TKEY, TF FROM #EX; SELECT name [Индекс], avg_fragmentation_in_percent [Фрагментация (%)] FROM sys.dm_db_index_physical_stats(DB_ID('TEMPDB'), OBJECT_ID(N'#EX'), NULL, NULL, NULL) ss JOIN sys.indexes ii ON ss.object_id = ii.object_id and ss.index_id = ii.index_id WHERE name is not null;</pre>
---	---

7. Создать необходимые индексы и проанализировать планы запросов с использованием этих индексов для таблицы базы данных **X_MyBASE**.

Тест "Применение индексов"

[В начало практикума](#)

Лабораторная работа № 11. Обработка результатов запросов с помощью курсоров

Курсор является программной конструкцией, которая дает возможность пользователю обрабатывать строки результирующего набора запись за записью. Курсоры бывают *локальные* и *глобальные* (по умолчанию), *статические* и *динамические* (по умолчанию).

Задание	Краткие теоретические сведения
<p>1. Разработать сценарий, формирующий список дисциплин на кафедре ИСиТ. В отчет должны быть выведены краткие названия дисциплин из таблицы SUBJECT в одну строку через запятую.</p> <p>Использовать встроенную функцию RTRIM.</p>	<p>Пусть требуется создать список заказанных товаров из таблицы Заказы. Наименования товаров должны выводиться в одну строку через запятую.</p> <p>Курсор объявляется в операторе DECLARE:</p> <pre>DECLARE @tv char(20), @t char(300) = ''; DECLARE ZkTovar CURSOR for SELECT Наименование_товара from Заказы; OPEN ZkTovar; FETCH ZkTovar into @tv; print 'Заказанные товары'; while @@fetch_status = 0 begin set @t = rtrim(@tv) + ',' + @t; FETCH ZkTovar into @tv; end; print @t; CLOSE ZkTovar;</pre> <p>Курсор открывается с помощью оператора OPEN.</p>

	<p>Оператор FETCH считывает одну строку из результирующего набора и продвигает указатель на следующую строку. Количество переменных в списке после ключевого слова INTO должно быть равно количеству столбцов результирующего набора, а порядок их должен соответствовать порядку перечисления столбцов в SELECT-списке.</p> <p>После выполнения FETCH проверяется значение функции <code>@@fetch_status</code>, которая возвращает значение 0, если оператор FETCH выполнен успешно; -1, если достигнут конец результирующего набора и строка не считывается; -2, если выбранная строка отсутствует в БД. В зависимости от полученного результата цикл продолжается и считывается следующая строка, или цикл заканчивается.</p> <p>Курсор закрывается с помощью оператора CLOSE.</p>
2. Разработать сценарий, демонстрирующий отличие глобального курсора от локального на примере базы данных UNIVER.	<p><i>Локальный</i> курсор может применяться в рамках одного пакета и ресурсы, выделенные ему при объявлении, освобождаются сразу после завершения работы пакета.</p> <pre> DECLARE Tovary CURSOR LOCAL for SELECT Наименование, Цена from Товары; DECLARE @tv char(20), @cena real; OPEN Tovary; fetch Tovary into @tv, @cena; print '1. '+@tv+cast(@cena as varchar(6)); go DECLARE @tv char(20), @cena real; fetch Tovary into @tv, @cena; print '2. '+@tv+cast(@cena as varchar(6)); </pre>

go

Выполняется первый пакет, а при попытке выполнить следующий появляется сообщение, что курсора с именем **Tovary** не существует.

Глобальный курсор может быть объявлен, открыт и использован в разных пакетах. Выделенные ему при объявлении ресурсы освобождаются только после выполнения оператора DEALLOCATE или при завершении сеанса пользователя.

```
DECLARE Tovary CURSOR GLOBAL
    for select Наименование, Цена from Товары;
DECLARE @tv char(20), @cena real;
    OPEN Tovary;
    fetch Tovary into @tv, @cena;
    print '1. '+@tv+cast(@cena as varchar(6));
    go
DECLARE @tv| char(20), @cena real;
    fetch Tovary into @tv, @cena;
    print '2. '+@tv+cast(@cena as varchar(6));
    close Tovary;
    deallocate Tovary;
    go
```

3. Разработать сценарий, демонстрирующий отличие статических курсоров от динамических на примере базы

Открытие *статического* курсора приводит к выгрузке результирующего набора во временную таблицу системной БД **TEMPDB**, и все дальнейшие операции осуществляются с этой таблицей.

После открытия курсора все текущие изменения в исходных таблицах не будут

данных UNIVER.

отражаться в результирующем наборе.

```
DECLARE @tid char(10), @tnm char(40), @tgn char(1);
DECLARE Zakaz CURSOR LOCAL STATIC
    for SELECT Наименование_товара, Цена_продажи, Количество
        FROM dbo.Заказы where Заказчик = 'Луч';
open Zakaz;
print 'Количество строк : '+cast(@@CURSOR_ROWS as varchar(5));
UPDATE Заказы set Количество = 5 where Наименование_товара = 'Стул';
DELETE Заказы where Наименование_товара = 'Шкаф';
INSERT Заказы (Номер_заказа, Наименование_товара, Цена_продажи,
    Количество, Дата_поставки, Заказчик)
values (18, 'Шкаф', 340, 1, '2014-08-02', 'Луч');
FETCH Zakaz into @tid, @tnm, @tgn;
while @@fetch_status = 0
begin
    print @tid + ''+ @tnm + ''+ @tgn;
    fetch Zakaz into @tid, @tnm, @tgn;
end;
CLOSE Zakaz;
```

Здесь значение функции @@CURSOR_ROWS равно **-n** (количество записей) при асинхронной выборке, равно **n** при синхронной выборке, равно **0**, если курсор не открыт. При выполнении курсора все изменения (UPDATE, DELETE и INSERT) в исходной таблице **Заказы** не отражаются на результате выборки строк.

Если заменить LOCAL STATIC на LOCAL DYNAMIC, то изменения будут отражаться в результирующем наборе.

<p>4. Разработать сценарий, демонстрирующий свойства навигации в результирующем наборе курсора с атрибутом SCROLL на примере базы данных UNIVER.</p> <p>Использовать все известные ключевые слова в операторе FETCH.</p>	<p>По умолчанию для курсора установлен атрибут SCROLL, позволяющий применять оператор FETCH с дополнительными опциями позиционирования.</p> <p>В примере ниже выводятся названия товаров, которые заказаны фирмой с названием «Луч». Перед каждым названием выводится номер строки результирующего набора, определяемый функцией ROW_NUMBER(). Выбор строки определяется соответствующим ключевым словом в операторе FETCH.</p> <pre> DECLARE @tc int, @rn char(50); DECLARE Primer1 cursor local dynamic SCROLL for SELECT row_number() over (order by Наименование_товара) N, Наименование_товара FROM dbo.Заказы where Заказчик = 'Луч' OPEN Primer1; FETCH Primer1 into @tc, @rn; print 'следующая строка : ' + cast(@tc as varchar(3))+ rtrim(@rn); FETCH LAST from Primer1 into @tc, @rn; print 'последняя строка : ' + cast(@tc as varchar(3))+ rtrim(@rn); CLOSE Primer1;</pre> <p>Можно дописать этот пример, используя другие ключевые слова: FIRST (первая строка), NEXT (следующая строка за текущей), PRIOR (предыдущая строка от текущей), ABSOLUTE 3 (третья строка от начала), ABSOLUTE -3 (третья строка от конца), RELATIVE 5 (пятая строка вперед от текущей), RELATIVE -5 (пятая</p>
--	---

	строка назад от текущей).
<p>5. Создать курсор, демонстрирующий применение конструкции CURRENT OF в секции WHERE с использованием операторов UPDATE и DELETE.</p> <p>6. Разработать SELECT-запрос, с помощью которого из таблицы PROGRESS удаляются строки, содержащие информацию о студентах, получивших оценки ниже 4 (использовать объединение таблиц PROGRESS, STUDENT, GROUPS).</p> <p>Разработать SELECT-запрос, с помощью которого в таблице PROGRESS для студента с конкретным номером IDSTUDENT корректируется оценка (увеличивается на единицу).</p>	<p>Курсыры с установленным свойством FOR UPDATE помимо чтения данных из строк с помощью оператора FETCH, могут эти строки изменять или удалять с помощью операторов UPDATE и DELETE, если в секции WHERE эти операторы используют операцию CURRENT OF, для которой указывается имя курсора (в примере ниже Primer2). Такой формат операторов позволяет удалять или изменять строки в таблице (в примере таблица Заказы), соответствующих <i>текущей</i> позиции курсора в результирующем наборе.</p> <p>В примере удаляется строка первая строка в таблице Заказы и увеличивается на единицу количество товаров в следующей строке.</p> <pre> declare @tn char(20), @tc real, @tk int; declare Primer2 cursor local dynamic for SELECT Наименование_товара, Цена_продажи, Количество FROM Заказы FOR UPDATE; OPEN Primer2; fetch Primer2 into @tn, @tc, @tk; DELETE Заказы where CURRENT OF Primer2; fetch Primer2 into @tn, @tc, @tk; UPDATE Заказы set Количество = Количество + 1 Where CURRENT OF Primer2; CLOSE Primer2; </pre>
7. Разработать курсоры для базы данных X_MyBASE и продемонстрировать их работу.	

8*. Отчет на рисунке справа содержит информацию о факультетах (таблица **FACULTY**), кафедрах (таблица **PULPIT**), количестве преподавателей на кафедрах (таблица **TEACHER**), а также перечень закрепленных за кафедрой дисциплин (таблица **SUBJECT**).

Разработать сценарий, формирующий подобный отчет. При этом, если нет дисциплин, закрепленных за кафедрой, в отчет выводится слово **нет**; коды дисциплин перечисляются в одной строке через запятую, список заканчивается точкой.

Примечание: использовать локальный статический курсор; в SELECT-запросе применить внешнее соединение таблиц, использовать функции RTRIM, SUBSTRING, LEN.

Факультет: ИДиП Кафедра: БФ Количество преподавателей: 0 Дисциплины: нет.	Кафедра: ИСиГ Количество преподавателей: 16 Дисциплины: БД, ДМ, ИНФ, КБ, КТ	Кафедра: ПОиСОИ Количество преподавателей: 5 Кафедра: РИТ Количество преподавателей: 0 Дисциплины: нет.	Факультет: ИЭФ Кафедра: МиЭП Количество преподавателей: 1 Дисциплины: ЭП
--	---	---	---

Тест "Обработка результатов запросов с помощью курсоров"

[В начало практикума](#)

Лабораторная работа № 12. Особенности использования транзакций

Транзакция – это механизм базы данных, позволяющий таким образом объединять несколько операторов, изменяющих базу данных, чтобы все выполнились или все не выполнились.

Основные свойства транзакции: *атомарность* (операторы изменения БД, включенные в транзакцию, либо выполняются все, либо не выполнится ни один); *согласованность* (транзакция должна фиксировать новое согласованное состояние БД); *изолированность* (отсутствие взаимного влияния параллельных транзакций на результаты их выполнения); *долговечность* (изменения в БД, выполненные и зафиксированные транзакцией, могут быть отменены только с помощью новой транзакции).

Задание	Краткие теоретические сведения
1. Разработать сценарий, демонстрирующий работу в режиме <i>неявной</i> транзакции. Проанализировать пример, приведенный справа, в котором создается таблица X, и создать сценарий для другой таблицы.	<p>Режим <i>неявной транзакции</i> может быть включен для текущего соединения с сервером БД с помощью специальной инструкции:</p> <p>SET IMPLICIT_TRANSACTIONS ON</p> <p>Обратное переключение осуществляется с использованием ключевого слова OFF вместо ON.</p> <p>Неявная транзакция начинается, если выполняется один из следующих операторов:</p> <p>CREATE, DROP; ALTER TABLE; INSERT, DELETE, UPDATE, SELECT, TRUNCATE TABLE; OPEN, FETCH; GRANT (выдача разрешений), REVOKE (запрещение разрешений).</p> <p>Неявная транзакция продолжается до тех пор, пока не будет выполнен оператор фиксации (COMMIT) или оператор отката (ROLLBACK) транзакции.</p> <p>В примере ниже неявная транзакция стартует при выполнении оператора CREATE TABLE и завершается фиксацией изменений с помощью оператора COMMIT.</p>

После этого осуществляется возврат в режим автофиксации (инструкция SET OFF). Созданная таблица с именем X сохранилась.

```
set nocount on
if exists (select * from SYS.OBJECTS      -- таблица X есть?
           where OBJECT_ID= object_id(N'DBO.X') )
drop table X;

declare @c int, @flag char = 'c';        -- commit или rollback?
SET IMPLICIT_TRANSACTIONS ON    -- включ. режим неявной транзакции
CREATE table X(K int);            -- начало транзакции
INSERT X values (1),(2),(3);
set @c = (select count(*) from X);
print 'количество строк в таблице X: ' + cast( @c as varchar(2));
if @flag = 'c' commit;             -- завершение транзакции: фиксация
else rollback;                   -- завершение транзакции: откат
SET IMPLICIT_TRANSACTIONS OFF   -- выключ. режим неявной транзакции

if exists (select * from SYS.OBJECTS      -- таблица X есть?
           where OBJECT_ID= object_id(N'DBO.X') )
print 'таблица X есть';
else print 'таблицы X нет'
```

Если в данном сценарии изменить значение переменной `@flag = 'r'`, то будет выполняться оператор ROLLBACK, что приведет к отмене всех изменений в БД, осуществленных последней транзакцией (созданная таблица X не сохранится).

2. Разработать сценарий, демонстрирующий свойство *автомарности явной транзакции* на примере базы данных UNIVER.

В блоке CATCH предусмотреть выдачу соответствующих сообщений об ошибках.

Опробовать работу сценария при использовании различных операторов модификации таблиц.

Переключение в режим *явной транзакции* осуществляется с помощью оператора BEGIN TRANSACTION. Транзакцию должен завершать один из операторов: COMMIT TRAN или ROLLBACK TRAN. После завершения явной транзакции происходит возврат в исходный режим (автофиксации или неявной транзакции).

Пусть требуется осуществить изменения в таблице **Товары** так, чтобы в случае ошибки изменения не были осуществлены.

```
begin try
    begin tran          -- начало явной транзакции
        delete Товары where Наименование='Стол';
        insert Товары values ('Стул',15, 20);
        insert Товары values ('Телевизор', 500,5);
        commit tran;      -- фиксация транзакции
    end try
    begin catch
        print 'ошибка: '+ case
        when error_number() = 2627 and patindex('%PK_Товары%', error_message()) > 0
            then 'дублирование товара'
            else 'неизвестная ошибка: '+ cast(error_number() as varchar(5))+ error_message()
        end;
        if @@trancount > 0 rollback tran ;
    end catch;
```

Транзакция начинается внутри TRY-блока и в случае успешного выполнения завершается оператором COMMIT. Если при выполнении возникла ошибка, то в CATCH-блоке формируется и выводится соответствующее сообщение, а затем выполняется откат

	<p>(ROLLBACK).</p> <p>Системная функция @@TRANCOUNT возвращает уровень вложенности транзакции. (если значение больше нуля, то транзакция не завершена).</p> <p>Встроенная функция PATINDEX определяет в строке позицию первого символа подстроки, заданную шаблоном. С помощью этой функции в тексте сообщения об ошибке отыскивается имя ограничения целостности.</p> <p>Изменения в таблице будут осуществлены только в случае отсутствия ошибок в операторах удаления и вставки.</p>
<p>3. Разработать сценарий, демонстрирующий применение оператора SAVE TRAN на примере базы данных UNIVER.</p> <p>В блоке CATCH предусмотреть выдачу соответствующих сообщений об ошибках.</p> <p>Опробовать работу сценария при использовании различных контрольных точек и различных операторов модификации таблиц.</p>	<p>Если транзакция состоит из нескольких независимых блоков операторов T-SQL, изменяющих базу данных, то может быть использован оператор SAVE TRANSACTION, формирующий <i>контрольную точку</i> транзакции.</p> <pre> declare @point varchar(32); -- макс. длина имени 32 begin try begin tran -- начало явной транзакции delete Товары where Наименование='Стол'; set @point = 'p1'; save tran @point; -- контрольная точка p1 insert Товары values ('Стул', 15, 20); set @point = 'p2'; save tran @point; -- контрольная точка p2 insert Товары values ('Телевизор', 500, 5); commit tran; -- фиксация транзакции end tran end try </pre>

```

begin catch
    print 'ошибка: '+ case when error_number() = 2627
        and patindex('%PK_Товары%', error_message()) > 0
        then 'дублирование товара'
        else 'неизвестная ошибка: '+ cast(error_number() as varchar(5))
    end
    if @@trancount > 0
        begin
            print 'контрольная точка: '+ @point;
            rollback tran @point; -- откат к контрольной точке
            commit tran; -- фиксация изменений, выполненных до контрольной точки
        end;
end catch;

```

4. Разработать два сценария А и В на примере базы данных UNIVER.

Сценарий А представляет собой явную транзакцию с уровнем изолированности READ UNCOMMITTED, сценарий В – явную транзакцию с уровнем изолированности READ COMMITTED (по умолчанию).

Пусть имеются две параллельные транзакции А и В для базы данных ПРОДАЖИ:

```

-- А --
set transaction isolation level READ UNCOMMITTED
begin transaction
----- t1 -----
select @@SPID, 'insert Товары' 'результат', * from Товары
    where Наименование = 'Блокнот';
select @@SPID, 'update Заказы' 'результат', Наименование_товара,
    Цена_продажи from Заказы where Наименование_товара = 'Блокнот';
commit;
----- t2 -----

```

Сценарий А должен демонстрировать, что уровень READ UNCOMMITTED допускает неподтвержденное, неповторяющееся и фантомное чтение.

--- В ---

```
begin transaction  
select @@SPID  
insert Товары values ('Блокнот', 2, 80);  
update Заказы set Наименование_товара = 'Блокнот'  
    where Наименование_товара = 'Стол'  
----- t1 -----  
----- t2 -----  
rollback;
```

Здесь системная функция @@SPID возвращает системный идентификатор процесса, назначенный сервером текущему подключению.

При параллельных транзакциях могут возникать три проблемы.

Неподтвержденное чтение. До момента t1 транзакцией В выполняются два оператора: INSERT и UPDATE. Эти операторы изменяют таблицы БД, но до момента времени t2 не фиксируют и не откатывают эти изменения. После момента t1 транзакция А считывает содержимое таблиц, измененных транзакцией В и «видит» измененные или добавленные строки. При этом изменения остаются до момента t2 в неподтвержденном состоянии, т. е. могут быть как зафиксированными, так и отмененными.

Неповторяющееся чтение. Одна транзакция читает данные несколько раз, а другая изменяет те же данные между двумя операциями чтения в первом процессе. По этой причине данные, прочитанные в различных операциях, будут разными.

Фантомное чтение. Две последовательные операции чтения могут получать различные значения, т. к. дополнительные строки, называемые фантомными, могут добавляться другими транзакциями.

	<p>Чтобы такие проблемы не возникали, определяются различные уровни изолированности: READ COMMITTED, REPEATABLE READ, SERIALIZABLE и др.</p> <p>Для моделирования процесса в этом и следующих заданиях следует выполнять сценарии в три этапа: последовательно оба сценария до отметки t1; последовательно оба сценария с отметкой t1 до отметки t2; последовательно оба сценария с отметкой t2 и до конца.</p>
<p>5. Разработать два сценария А и В на примере базы данных UNIVER.</p> <p>Сценарии А и В представляют собой явные транзакции с уровнем изолированности READ COMMITTED.</p> <p>Сценарий А должен демонстрировать, что уровень READ COMMITTED не допускает неподтвержденного чтения, но при этом возможно неповторяющееся и фантомное чтение.</p>	<p>Пример для базы данных ПРОДАЖИ:</p> <p>--- A ---</p> <pre>set transaction isolation level READ COMMITTED begin transaction select count(*) from Заказы where Наименование_товара = 'Стул'; ----- t1 ----- ----- t2 ----- select 'update Заказы' 'результат', count(*) from Заказы where Наименование_товара = 'Стул'; commit;</pre> <p>--- B ---</p> <pre>begin transaction ----- t1 ----- update Заказы set Наименование_товара = 'Стул' where Наименование_товара = 'Стол' commit; ----- t2 -----</pre>

6. Разработать два сценария А и В на примере базы данных UNIVER.

Сценарий А представляет собой явную транзакцию с уровнем изолированности REPEATABLE READ. Сценарий В – явную транзакцию с уровнем изолированности READ COMMITTED.

Сценарий А должен демонстрировать, что уровень REAPETABLE READ не допускает неподтвержденного чтения и неповторяющегося чтения, но при этом возможно фантомное чтение.

Пример для базы данных ПРОДАЖИ:

-- А ---

```
set transaction isolation level REPEATABLE READ  
begin transaction  
select Заказчик from Заказы where Наименование_товара = 'Стул';
```

t1

t2

```
select case  
when Заказчик = 'Луч' then 'insert Заказы' else ''  
end 'результат', Заказчик from Заказы where Наименование_товара = 'Стул';  
commit;
```

-- В ---

```
begin transaction
```

t1

```
insert Заказы values (12, 'Стул', 78, 10, '01.12.2014', 'Луч');  
commit;
```

t2

7. Разработать два сценария А и В на примере базы данных UNIVER.

Сценарий А представляет собой явную транзакцию с

Пример для базы данных ПРОДАЖИ:

-- А ---

```
set transaction isolation level SERIALIZABLE  
begin transaction  
delete Заказы where Заказчик = 'Луч';
```

<p>уровнем изолированности SERIALIZABLE.</p> <p>Сценарий В – явную транзакцию с уровнем изолированности READ COMMITTED.</p> <p>Сценарий А должен демонстрировать отсутствие фантомного, неподтвержденного и неповторяющегося чтения.</p>	<pre><code>insert Заказы values (14, 'Стул', 78, 10, '01.12.2014', 'Луч'); update Заказы set Заказчик = 'Луч' where Наименование_товара = 'Стул'; select Заказчик from Заказы where Наименование_товара = 'Стул'; ----- t1 ----- select Заказчик from Заказы where Наименование_товара = 'Стул'; ----- t2 ----- commit; --- В --- begin transaction delete Заказы where Заказчик = 'Луч'; insert Заказы values (14, 'Стул', 78, 10, '01.12.2014', 'Луч'); update Заказы set Заказчик = 'Луч' where Наименование_товара = 'Стул'; select Заказчик from Заказы where Наименование_товара = 'Стул'; ----- t1 ----- commit; select Заказчик from Заказы where Наименование_товара = 'Стул'; ----- t2 -----</code></pre>
<p>8. Разработать сценарий, демонстрирующий свойства вложенных транзакций, на примере базы данных UNIVER.</p>	<p>Транзакция, выполняющаяся в рамках другой транзакции, называется <i>вложенной</i>. При работе с вложенными транзакциями нужно учитывать следующее:</p> <ul style="list-style-type: none"> – оператор COMMIT вложенной транзакции действует только на внутренние операции вложенной транзакции; – оператор ROLLBACK внешней транзакции отменяет зафиксированные операции внутренней транзакции;

- оператор ROLLBACK вложенной транзакции действует на операции внешней и внутренней транзакции, а также завершает обе транзакции;
- уровень вложенности транзакции можно определить с помощью системной функции @@TRANCOUNT.

```
begin tran          -- внешняя транзакция
insert Заказчики values ('Луч', 'Минск', 10234);
begin tran          -- внутренняя транзакция
update Заказы set Наименование_товара = 'Луч' where Заказчик = 'Луч';
commit;            -- внутренняя транзакция
if @@trancount > 0 rollback;    -- внешняя транзакция
select
    (select count(*) from Заказы where Заказчик = 'Луч') 'Заказы',
    (select count(*) from Заказчики where Наименование_фирмы = 'Луч') 'Заказчики';
```

9. Разработать скрипты с использованием транзакций для базы данных X_MyBASE.

Тест "Особенности использования транзакций"

[В начало практикума](#)

Лабораторная работа № 13. Разработка хранимых процедур

Хранимая процедура – это поименованный код на языке Transact-SQL. Хранимая процедура может быть создана с помощью CREATE, изменена с помощью ALTER и удалена с помощью оператора DROP. Процедура может принимать входные и формировать выходные параметры. Результатом ее выполнения может быть целочисленное значение, которое возвращается к точке вызова оператором RETURN, либо один или более результирующих наборов, сформированных операторами SELECT, либо содержимое стандартного выходного потока, полученного при выполнении операторов PRINT.

Вызов процедуры осуществляется оператором EXECUTE (EXEC). В хранимых процедурах допускается применение основных DDL, DML и TSQL-операторов, конструкций TRY/CATCH, курсоров, временных таблиц.

Задание	Краткие теоретические сведения																											
<p>1. Разработать хранимую процедуру без параметров с именем PSUBJECT. Процедура формирует результирующий набор на основе таблицы SUBJECT, аналогичный набору, представленному на рисунке:</p> <table border="1"><thead><tr><th>код</th><th>дисциплина</th><th>кафедра</th></tr></thead><tbody><tr><td>БД</td><td>Базы данных</td><td>ИСиТ</td></tr><tr><td>БЛЗиПсОО</td><td>Биология лесных зверей и птиц с осн. охотов.</td><td>ОВ</td></tr><tr><td>ВТЛ</td><td>Водный транспорт леса</td><td>ТЛ</td></tr><tr><td>ДМ</td><td>Дискретная математика</td><td>ИСиТ</td></tr><tr><td>ИГ</td><td>Инженерная геодезия</td><td>ЛУ</td></tr><tr><td>ИНФ</td><td>Информационные технологии</td><td>ИСиТ</td></tr><tr><td>КБ</td><td>Комбинаторика</td><td>ИСиТ</td></tr><tr><td>КГ</td><td>Компьютерная геометрия</td><td>ИСиТ</td></tr></tbody></table> <p>Процедура должна возвращать количество строк,</p>	код	дисциплина	кафедра	БД	Базы данных	ИСиТ	БЛЗиПсОО	Биология лесных зверей и птиц с осн. охотов.	ОВ	ВТЛ	Водный транспорт леса	ТЛ	ДМ	Дискретная математика	ИСиТ	ИГ	Инженерная геодезия	ЛУ	ИНФ	Информационные технологии	ИСиТ	КБ	Комбинаторика	ИСиТ	КГ	Компьютерная геометрия	ИСиТ	<p>Процедура ниже предназначена для подсчета количества строк в таблице Заказы и вывода ее содержимого:</p> <pre>use ПРОДАЖИ go CREATE PROCEDURE PrZakazy -- создание процедуры as begin declare @k int = (select count (*) from Заказы); select * from Заказы; return @k; end;</pre> <p>Обращение к процедуре осуществляется оператором EXEC:</p>
код	дисциплина	кафедра																										
БД	Базы данных	ИСиТ																										
БЛЗиПсОО	Биология лесных зверей и птиц с осн. охотов.	ОВ																										
ВТЛ	Водный транспорт леса	ТЛ																										
ДМ	Дискретная математика	ИСиТ																										
ИГ	Инженерная геодезия	ЛУ																										
ИНФ	Информационные технологии	ИСиТ																										
КБ	Комбинаторика	ИСиТ																										
КГ	Компьютерная геометрия	ИСиТ																										

выведенных в результирующий набор.

2. Найти процедуру **PSUBJECT** с помощью обозревателя объектов (Object Explorer) и через контекстное меню создать сценарий на изменение процедуры оператором ALTER.

Изменить процедуру **PSUBJECT**, созданную в задании 1, таким образом, чтобы она принимала два параметра с именами **@p** и **@c**. Параметр **@p** является входным, имеет тип **varchar(20)** и значение по умолчанию **NULL**. Параметр **@c** является выходным, имеет тип **INT**.

Процедура **PSUBJECT** должна формировать результирующий набор, аналогичный набору, представленному на рисунке выше, но при этом содержать строки, соответствующие коду кафедры, заданному параметром **@p**. Кроме того, процедура должна формировать значение выходного параметра **@c**, равное количеству строк в результирующем наборе, а также возвращать значение к точке вызова, равное общему количеству дисциплин (количество строк в таблице **SUBJECT**).

```
declare @k int = 0;
EXEC @k = PrZakazy; -- вызов процедуры
print 'кол-во товаров = ' + cast(@k as varchar(3));
```

С помощью оператора ALTER можно изменить процедуру:

```
alter procedure PrZakazy @p varchar(20), @c int output
as begin
declare @k int = (select count(*) from Заказы );
print 'параметры: @p = ' + @p + ', @c = ' + cast(@c as varchar(3));
select * from Заказы where Заказчик = @p;
set @c = @@rowcount;
return @k;
end;
```

Здесь процедура принимает два параметра: входной параметр **@p** и выходной (ключевое слово **OUTPUT**) параметр **@c**. Процедура формирует значение выходного параметра **@c**, выводит в стандартный поток сообщение (оператор **PRINT**), а также формирует результирующий набор. Вызов процедуры:

```
declare @k int = 0, @r int = 0, @p varchar(20);
exec @k = PrZakazy @p = 'Луч', @c = @r output;
print 'кол-во товаров всего = ' + cast(@k as varchar(3));
print 'кол-во товаров, заказанных фирмой ' + cast(@p
as varchar(3)) + ' = ' + cast(@r as varchar(3));
```

3. Создать временную локальную таблицу с именем **#SUBJECT**. Наименование и тип столбцов таблицы должны соответствовать столбцам результирующего набора процедуры **PSUBJECT**, разработанной в задании 2.

Изменить процедуру **PSUBJECT** таким образом, чтобы она не содержала выходного параметра.

Применив конструкцию **INSERT... EXECUTE** с модифицированной процедурой **PSUBJECT**, добавить строки в таблицу **#SUBJECT**.

Процедуры, формирующие результирующий набор и не имеющие выходных параметров, могут быть применены в операторе **INSERT** в качестве источника строк для добавления в некоторую таблицу.

Пусть с помощью **ALTER** внесены изменения в процедуру:

```
alter procedure PrZakazy @p varchar(20)
as begin
declare @k int = (select count(*) from Заказы );
select * from Заказы where Заказчик = @p;
end;
```

Временная таблица может быть создана с помощью операторов:

```
CREATE table #Zk
( Номер_заказа int primary key,
Наименование_товара nvarchar(50),
Цена_продажи real,
Количество int,
Дата_поставки date,
Заказчик nvarchar(50))
```

Операторы **INSERT** добавляют строки во временную таблицу **#Zk**:

```
INSERT #Zk exec PrZakazy @p = 'Zte';
INSERT #Zk exec PrZakazy @p = 'Белвест';
```

Просмотреть содержимое временной таблицы можно с помощью оператора **SELECT**:

```
select * from #Zk
```

4. Разработать процедуру с именем **PAUDITORIUM_INSERT**. Процедура принимает четыре входных параметра: **@a**, **@n**, **@c** и **@t**. Параметр **@a** имеет тип CHAR(20), параметр **@n** имеет тип VARCHAR(50), параметр **@c** имеет тип INT и значение по умолчанию 0, параметр **@t** имеет тип CHAR(10).

Процедура добавляет строку в таблицу **AUDITORIUM**. Значения столбцов **AUDITORIUM**, **AUDITORIUM_NAME**, **AUDITORIUM_CAPACITY** и **AUDITORIUM_TYPE** добавляемой строки задаются соответственно параметрами **@a**, **@n**, **@c** и **@t**.

Процедура **PAUDITORIUM_INSERT** должна применять механизм TRY/CATCH для обработки ошибок. В случае возникновения ошибки, процедура должна формировать сообщение, содержащее код ошибки, уровень серьезности и текст сообщения в стандартный выходной поток.

Процедура должна возвращать к точке вызова значение **-1** в том случае, если произошла ошибка и **1**, если выполнение успешно.

Опробовать работу процедуры с различными значениями исходных данных.

Процедура, представленная ниже, осуществляет вставку строки в таблицу **Товары** и обработку ошибки, если таковая появится:

```
use ПРОДАЖИ
go
create procedure TovaryInsert
    @t NVARCHAR(50), @cn REAL, @kl INT = null
as declare @rc int = 1;
begin try
    insert into Товары (Наименование, Цена, Количество)
        values (@t, @cn, @kl)
    return @rc;
end try
begin catch      -- обработка ошибки
    print 'номер ошибки : ' + cast(error_number() as varchar(6));
    print 'сообщение : ' + error_message();
    print 'уровень : ' + cast(error_severity() as varchar(6));
    print 'метка : ' + cast(error_state() as varchar(8));
    print 'номер строки : ' + cast(error_line() as varchar(8));
    if error_procedure() is not null
        print 'имя процедуры : ' + error_procedure();
    return -1;
end catch;
```

Обращение к процедуре:

	<pre>declare @rc int; exec @rc = TovaryInsert @t = 'Планшет', @cn = 160, @kl = 90; print 'код ошибки : ' + cast(@rc as varchar(3));</pre>
--	---

Если в операторе EXEC допустить ошибку, например, ввести повторно наименование уже имеющегося товара или неправильно использовать имена переменных, то будет выдано сообщение об ошибке.

5. Разработать процедуру с именем **SUBJECT_REPORT**, формирующую в стандартный выходной поток отчет со списком дисциплин на конкретной кафедре. В отчет должны быть выведены краткие названия (поле SUBJECT) из таблицы SUBJECT в одну строку через запятую (использовать встроенную функцию RTRIM). Процедура имеет входной параметр с именем @p типа CHAR(10), который предназначен для указания кода кафедры.

В том случае, если по заданному значению @p невозможно определить код кафедры, процедура должна генерировать ошибку с сообщением **ошибка в параметрах**.

Процедура **SUBJECT_REPORT** должна возвращать к точке вызова количество дисциплин, отображенных в отчете.

Пусть требуется разработать процедуру, которая выводит отчет о товарах, заказанных конкретным заказчиком:

```
create procedure Zkz_REPORT @p CHAR(50)
as
declare @rc int = 0;
begin try
declare @tv char(20), @t char(300) = '';
declare ZkTov CURSOR for
select Наименование_товара from Заказы where Заказчик = @p;
if not exists (select Наименование_товара
from Заказы where Заказчик = @p)
raiserror('ошибка', 11, 1);
else
open ZkTov;
fetch ZkTov into @tv;
print 'Заказанные товары';
while @@fetch_status = 0
```

```
begin
    set @t = rtrim(@tv) + ',' + @t;
    set @rc = @rc + 1;
    fetch ZkTov into @tv;
end;
print @t;
close ZkTov;
return @rc;
end try
begin catch
    print 'ошибка в параметрах'
    if error_procedure() is not null
        print 'имя процедуры : ' + error_procedure();
    return @rc;
end catch;
```

Здесь для формирования сообщения об ошибке применяется встроенная функция RAISERROR, которая содержит три параметра: текстовое сообщение об ошибке, уровень серьезности ошибки и метку. Если уровень серьезности равен 11, то управление передается в блок обработки ошибок. Вызов процедуры:

```
declare @rc int;
exec @rc = Zkz_REPORT @p = 'Луч';
print 'количество товаров = ' + cast(@rc as varchar(3));
```

6. Разработать процедуру с именем **PAUDITORIUM_INSERTX**. Процедура принимает пять входных параметров: **@a, @n, @c, @t** и **@tn**.

Параметры **@a, @n, @c, @t** аналогичны параметрам процедуры **PAUDITORIUM_INSERT**. Параметр **@tn** является входным, имеет тип **VARCHAR(50)**, предназначен для ввода значения в столбец **AUDITORIUM_TYPE.AUDITORIUM_TYPENAME**.

Процедура добавляет две строки. Первая строка добавляется в таблицу **AUDITORIUM_TYPE**. Значения столбцов **AUDITORIUM_TYPE** и **AUDITORIUM_TYPENAME** задаются соответственно параметрами **@t** и **@tn**. Вторая строка добавляется путем вызова процедуры **PAUDITORIUM_INSERT**.

Добавление строки в таблицу **AUDITORIUM_TYPE** и вызов процедуры **PAUDITORIUM_INSERT** должны выполняться в рамках одной транзакции с уровнем изолированности **SERIALIZABLE**.

В процедуре должна быть предусмотрена обработка ошибок с помощью механизма TRY/CATCH. Все ошибки должны быть обработаны с выдачей соответствующего сообщения в стандартный выходной поток.

Для базы данных **ПРОДАЖИ** представленная ниже процедура осуществляет добавление информации в таблицу **Заказы** и **Товары**:

```
use ПРОДАЖИ
go
create procedure TovaryInsert_X
    @a int, @b NVARCHAR(50), @c REAL, @d INT = null,
    @e date, @f NVARCHAR(50)
as declare @rc int=1;
begin try
    set transaction isolation level SERIALIZABLE;
    begin tran
        insert into Заказы (Номер_заказа, Наименование_товара,
        Цена_продажи, Количество, Дата_поставки, Заказчик)
        values (@a, @b, @c, @d, @e, @f)
        exec @rc=TovaryInsert @b, @c, @d;
    commit tran;
    return @rc;
end try
begin catch
    print 'номер ошибки : ' + cast(error_number() as varchar(6));
    print 'сообщение : ' + error_message();
    print 'уровень : ' + cast(error_severity() as varchar(6));
    print 'метка : ' + cast(error_state() as varchar(8));
end catch;
```

Процедура **PAUDITORIUM_INSERTX** должна возвращать к точке вызова значение **-1** в том случае, если произошла ошибка и **1**, если выполнения процедуры завершилось успешно.

```
print 'номер строки : ' + cast(error_line() as varchar(8));
if error_procedure() is not null
    print 'имя процедуры : ' + error_procedure();
    if @@trancount > 0 rollback tran ;
    return -1;
end catch;
```

Вызов процедуры:

```
declare @rc int;
exec @rc = TovaryInsert_X @a = 20, @b = 'Стол', @c = 78,
@d = 10, @e = '01.12.2014', @f = 'Луч';
print 'код ошибки=' + cast(@rc as varchar(3));
```

7. Разработать хранимые процедуры для базы данных **X_MyBASE** и продемонстрировать их работу.

8*. Разработать процедуру с именем **PRINT_REPORT**, формирующую в стандартный выходной поток отчет, аналогичный отчету, представленному на рисунке в задании 8 лабораторной работы № 12.

Процедура имеет два входных параметра с именами: **@f** и **@p**. Параметр **@f** является входным, имеет тип CHAR(10) и значение по умолчанию NULL. Параметр предназначен для указания кода факультета (столбец **FACULTY.FACULTY**).

Параметр **@p** является входным, имеет тип CHAR(10) и значение по умолчанию NULL. Параметр предназначен для указания кода кафедры (столбец **PULPIT.PULPIT**).

Если значение параметра **@f** не равно NULL, а значение **@p** равно NULL, то отчет формируется только для заданного параметром **@f** факультета. Если значение параметра **@f** не равно NULL и значение **@p** тоже не равно NULL, то отчет формируется для заданной кафедры (параметр **@p**) заданного факультета (параметр **@f**).

Если значение параметра `@f` равно NULL, а значение `@p` не равно NULL, то факультет должен быть определен по значению столбца **PULPIT.FACULY** в строке соответствующей значению `@p`. В том случае, если по заданному значению `@p` невозможно определить код факультета, процедура должна генерировать ошибку с сообщением **ошибка в параметрах**.

Процедура **PRINT_REPORT** должна возвращать к точке вызова, количество кафедр, отображенных в отчете.

Разработать сценарий,зывающий процедуру **PRINT_REPORT** и предусматривающий обработку ошибок, возникших в процедуре.

Примечание: при разработке процедуры использовать сценарий пункта 1 и применить механизм TRY/CATCH.

Тест "Разработка хранимых процедур"

[В начало практикума](#)

Лабораторная работа № 14. Разработка и использование функций

Функция – это объект БД, представляющий собой поименованный код T-SQL. Для создания, удаления и изменения функций надо использовать операторы CREATE, DROP и ALTER соответственно.

Отличие функций от хранимых процедур в ограничениях, накладываемых на код функции, в форме представления результата работы, а также в способе вызова. В функции не допускается применение DDL-операторов, DML-операторов, изменяющих БД (INSERT, DELETE, UPDATE), конструкций TRY/CATCH, а также использование транзакций.

Результатом выполнения функции является возвращаемое к точке вызова значение. Если функция возвращает единственное значение (число, строка, дата, время и пр.), то она называется *скалярной*. Функция, возвращающая таблицу, называется *табличной*. В зависимости от структуры кода, различают *встроенные* функции и *многооператорные* табличные функции.

Задание	Краткие теоретические сведения
<p>1. Разработать <i>скалярную</i> функцию с именем COUNT_STUDENTS, которая вычисляет количество студентов на факультете, код которого задается параметром типа varchar(20) с именем @faculty. Использовать внутреннее соединение таблиц FACULTY, GROUPS, STUDENT. Опробовать работу функции.</p> <p>Внести изменения в текст функции с помощью оператора ALTER с тем, чтобы функция принимала второй параметр @prof типа varchar(20), обозначающий специальность студентов. Для параметров определить значения по умолчанию</p>	<p>Пусть требуется подсчитать количество товаров, заказанных фирмой:</p>

NULL. Опробовать работу функции с помощью SELECT-запросов.

```
use ПРОДАЖИ
go
create function COUNT_Zakazy(@f varchar(20)) returns int
as begin declare @rc int = 0;
set @rc = (select count(Номер_заказа)
            from Заказы z join Заказчики zk
                          on z.Заказчик = zk.Наименование_фирмы
                         where Наименование_фирмы = @f);
return @rc;
end;
```

При вызове функции надо указывать ее имя с точностью до схемы БД. Если при создании функции имя схемы не указано, то она размещается по умолчанию в схеме DBO.

```
declare @f int = dbo.COUNT_Zakazy('Луч');
print 'Количество заказов=' + cast(@f as varchar(4));
```

Другой запрос:

```
select Наименование_фирмы,
       dbo.COUNT_Zakazy(Наименование_фирмы)
      from Заказчики;
```

2. Разработать скалярную функцию с именем **FSUBJECTS**, принимающую параметр **@p** типа **varchar(20)**, значение которого задает код кафедры (столбец **SUBJECT.PULPIT**).

Пусть требуется создать отчет о заказанных товарах тех фирм, информация о которых имеется в таблице **Заказчики**, с помощью скалярной функции **FZakazy**:

```
create FUNCTION FZakazy(@tz char(20)) returns char(300)
```

Функция должна возвращать строку типа **varchar(300)** с перечнем дисциплин в отчете.

Создать и выполнить сценарий, который создает отчет, аналогичный представленному ниже.

Использовать локальный статический курсор на основе SELECT-запроса к таблице **SUBJECT**.

select PULPIT, dbo.FSUBJECTS (PULPIT) from PULPIT	
PULPIT	(No column name)
БФ	Дисциплины.
ИСиТ	Дисциплины: БД, ДМ, ИНФ, КБ, КГ, КМС, ЛЭВМ, МП, ...
ЛВ	Дисциплины.
ЛЗиДВ	Дисциплины: ЛВ.
ЛКиП	Дисциплины.
ЛМиЛЗ	Дисциплины: ТиОЛ.
ЛПиСПС	Дисциплины: ОСПиЛПХ.
ЛУ	Дисциплины: ИГ.

```
as
begin
declare @tv char(20);
declare @t varchar(300) = 'Заказанные товары: ';
declare ZkTovar CURSOR LOCAL
for select Наименование_товара from Заказы
      where Заказчик = @tz;
open ZkTovar;
fetch ZkTovar into @tv;
while @@fetch_status = 0
begin
    set @t = @t + ',' + rtrim(@tv);
    FETCH ZkTovar into @tv;
end;
return @t;
end;
```

Обращение к функции:

```
select Наименование_фирмы, dbo.FZakazy
      (Наименование_фирмы) from Заказчики;
```

3. Разработать *табличную* функцию **FFACPUL**, результаты работы которой продемонстрированы на рисунке ниже.

Функция принимает два параметра, задающих код факультета (столбец **FACULTY.FACULTY**) и код

Встроенная *табличная* функция **FTovCena** выводит информацию об исходных ценах и ценах продажи, используя таблицы **Товары** и **Заказы**:

кафедры (столбец **PULPIT.PULPIT**). Использует SELECT-запрос с левым внешним соединением между таблицами **FACULTY** и **PULPIT**.

Если оба параметра функции равны NULL, то она возвращает список всех кафедр на всех факультетах.

Если задан первый параметр (второй равен NULL), функция возвращает список всех кафедр заданного факультета.

Если задан второй параметр (первый равен NULL), функция возвращает результирующий набор, содержащий строку, соответствующую заданной кафедре.

```
select * from dbo.FFACPUL(NULL,NULL);
FACULTY PULPIT
ИДиП БФ
ИДиП ИСиТ
ЛХФ ЛВ
ТТЛП ЛМиЛЗ
ХТИТ МиАХиСП
ИЭФ МиЭП

select * from dbo.FFACPUL ('ИДиП',NULL);
FACULTY PULPIT
ИДиП БФ
ИДиП ИСиТ
```

Если заданы два параметра, функция возвращает результирующий набор, содержащий строку,

```
create function FTovCena(@f varchar(50), @p real)
returns table
as return
select f.Наименование, f.Цена, p.Цена_продажи
from Товары f left outer join Заказы p
on f.Наименование = p.Наименование_товара
where f.Наименование = isnull(@f, f.Наименование)
and
p.Цена_продажи = isnull(@p, p.Цена_продажи);
```

Различные варианты обращения к функции позволяют проанализировать ее работу:

```
select * from dbo.FTovCena(NULL, NULL);
select * from dbo.FTovCena('Стул', NULL);
select * from dbo.FTovCena(NULL, 400);
select * from dbo.FTovCena('Шкаф', 340);
```

соответствующую заданной кафедре на заданном факультете.

Если по заданным значениям параметров невозможно сформировать строки, функция возвращает пустой результирующий набор.

4. На рисунке ниже показан сценарий, демонстрирующий работу *скалярной* функции **FCTEACHER**.

Функция принимает один параметр, задающий код кафедры. Функция возвращает количество преподавателей на заданной параметром кафедре. Если параметр равен NULL, то возвращается общее количество преподавателей.

Скалярная функция **FKolTov** выводит информацию о товарах, заказанных конкретным заказчиком, используя таблицу **Заказы**. Если передается в функцию параметр, равный NULL, то выводится сообщение об общем количестве заказов:

```
create function FKolTov(@p varchar(50)) returns int
as
begin
    declare @rc int =(select count(*) from Заказы
    where Заказчик = isnull(@p, Заказчик));
    return @rc;
end;
go
```

Обращения к функции:

```
select PULPIT, dbo.FCTEACHER(PULPIT) [Количество преподавателей]
from PULPIT
```

PULPIT	Количество преподавателей
БФ	0
ИСиТ	16
ЛВ	2
ЛЗиДВ	1
ЛКиП	0
ЛМиЛЗ	1
ЛПиСПС	2
ЛУ	2
МиАХиСП	0
МиЭП	1

```
select dbo.FCTEACHER(NULL) [Всего преподавателей]
```

Всего преподавателей
39

```
select Заказчик, dbo.FKoltov(Заказчик)
[Количество заказов] from Заказы
```

```
select dbo.FKoltov(NULL) [Всего заказов]
```

Разработать функцию **FCTEACHER**.

5. Разработать различные типы функций для базы данных **X_MyBASE** и продемонстрировать их работу.
6. Проанализировать *многооператорную табличную* функцию **FACULTY_REPORT**, представленную ниже:

```
create function FACULTY_REPORT(@c int) returns @fr table
( [Факультет] varchar(50), [Количество кафедр] int, [Количество групп] int,
[Количество студентов] int, [Количество специальностей] int )
as begin
```

```

declare cc CURSOR static for
select FACULTY from FACULTY
      where dbo.COUNT_STUDENTS(FACULTY, default) > @c;
declare @f varchar(30);
open cc;
fetch cc into @f;
while @@fetch_status = 0
begin
  insert @fr values( @f, (select count(PULPIT) from PULPIT where FACULTY = @f),
  (select count(IDGROUP) from GROUPS where FACULTY = @f), dbo.COUNT_STUDENTS(@f, default),
  (select count(PROFESSION) from PROFESSION where FACULTY = @f) );
  fetch cc into @f;
end;
return;
end;

```

Изменить эту функцию так, чтобы количество кафедр, количество групп, количество студентов и количество специальностей вычислялось отдельными скалярными функциями.

7*. Рассмотреть хранимую процедуру с именем **PRINT_REPORT** из пункта 8 лабораторной работы № 14. Создать новую версию этой процедуры с именем **PRINT_REPORTX**.

Процедура **PRINT_REPORTX** должна работать аналогично процедуре **PRINT_REPORT** и иметь тот же набор параметров, но SELECT-запрос курсора в новой процедуре должен использовать функции **FSUBJECTS**, **FFACPUL** и **FCTEACHER**.

Сравнить результаты, полученные процедурами **PRINT_REPORT** и **PRINT_REPORTX** и убедиться в работоспособности новой функции.

Тест "Разработка и использование функций"

[В начало практикума](#)

Лабораторная работа № 15. Применение триггеров

Триггер – это особый вид хранимой процедуры, предназначеннной для обработки событий в БД. Поддерживается два типа триггеров: *DDL-триггеры* и *DML-триггеры*. Для каждого типа определено свое семейство событий, обработку которых триггер этого типа может выполнять.

DML-триггеры бывают двух типов: AFTER-триггеры и INSTEAD OF-триггеры. Триггеры типа AFTER исполняются *после* выполнения оператора, вызвавшего соответствующее событие. При этом создаются автоматически две псевдотаблицы INSERTED и DELETED.

Триггер типа INSTEAD OF выполняется *вместо* оператора, вызвавшего соответствующее событие. Выполнение INSTEAD OF триггера предшествует проверке установленных для таблицы ограничений целостности.

Задание	Краткие теоретические сведения
<p>1. С помощью сценария, представленного на рисунке, создать таблицу TR_AUDIT.</p> <pre>create table TR_AUDIT (ID int identity, -- номер STMT varchar(20) -- DML-оператор check (STMT in ('INS', 'DEL', 'UPD')), TRNAME varchar(50), -- имя триггера CC varchar(300) -- комментарий)</pre> <p>Таблица предназначена для добавления в нее строк триггерами.</p>	<p>Событие INSERT при выполнении AFTER-триггера приводит к тому, что в псевдотаблицу INSERTED помещаются строки, добавленные оператором INSERT, вызвавшим это событие. Псевдотаблица DELETED остается пустой.</p> <p>При возникновении события DELETE в таблицу DELETED копируются удаленные строки, а таблица INSERTED остается пустой.</p> <p>При изменении строк таблицы с помощью оператора UPDATE заполняются обе псевдотаблицы, при этом таблица INSERTED содержит обновленные версии строк, а таблица DELETED – версию строк до их изменения.</p> <p>Пусть вспомогательная таблица для отражения информации об операциях с таблицей Товары для базы данных ПРОДАЖИ имеет имя TR_Tov:</p>

В столбец **STMT** триггер должен поместить событие, на которое он среагировал, а в столбец **TRNAME** – собственное имя.

Разработать AFTER-триггер с именем **TR_TEACHER_INS** для таблицы **TEACHER**, реагирующий на событие **INSERT**. Триггер должен записывать строки вводимых данных в таблицу **TR_AUDIT**. В столбец **CC** помещаются значения столбцов вводимой строки.

2. Создать AFTER-триггер с именем **TR_TEACHER_DEL** для таблицы **TEACHER**, реагирующий на событие **DELETE**. Триггер должен записывать строку данных в таблицу **TR_AUDIT** для каждой удаляемой строки. В столбец **CC** помещаются значения столбца **TEACHER** удаляемой строки.

3. Создать AFTER-триггер с именем **TR_TEACHER_UPD** для таблицы **TEACHER**, реагирующий на событие **UPDATE**. Триггер должен записывать строку данных в таблицу **TR_AUDIT** для каждой изменяемой строки. В столбец **CC** помещаются значения столбцов изменяемой строки до и после изменения.

```
use ПРОДАЖИ  
go  
create table TR_Tov
```

```
(  
    ID int identity, -- номер  
    ST varchar(20) check (ST in ('INS', 'DEL', 'UPD')),  
    TRN varchar(50), -- имя триггера  
    C varchar(300) -- комментарий  
)
```

AFTER-триггер с именем **TRIG_Tov_Ins** для таблицы **Товары** реагирует на событие **INSERT**. В столбец **ST** триггер помещает событие, на которое он среагировал, в столбец **TRN** – собственное имя, в столбец **C** – значения столбцов вводимой строки, которые берутся из псевдотаблицы **INSERTED**.

```
create trigger TRIG_Tov_Ins  
    on Товары after INSERT  
as declare @a1 varchar(20), @a2 real, @a3 int, @in varchar(300);  
print 'Операция вставки';  
set @a1 = (select [Наименование] from INSERTED);  
set @a2 = (select [Цена] from INSERTED);  
set @a3 = (select [Количество] from INSERTED);  
set @in = @a1 + ' ' + cast(@a2 as varchar(20)) + ' ' + cast(@a3 as  
varchar(20));  
insert into TR_Tov(ST, TRN, C) values('INS', 'TRIG_Tov_Ins', @in);  
return;
```

	<p>Триггер реагирует на событие вставки информации:</p> <pre>insert into Товары(Наименование, Цена, Количество) values('Планшет', 140, 20);</pre> <p>Содержимое таблицы TR_Tov отображается с помощью запроса:</p> <pre>select * from TR_Tov</pre>
<p>4. Создать AFTER-триггер с именем TR_TEACHER для таблицы TEACHER, реагирующий на события INSERT, DELETE, UPDATE.</p> <p>Триггер должен записывать строку данных в таблицу TR_AUDIT для каждой изменяемой строки. В коде триггера определить событие, активизировавшее триггер и поместить в столбец СС соответствующую событию информацию.</p> <p>Разработать сценарий, демонстрирующий работоспособность триггера.</p>	<p>Пример триггера TRIG_Tov, который записывает данные в таблицу TR_Tov для каждой изменяемой строки таблицы Товары:</p> <pre>create trigger TRIG_Tov on Товары after INSERT, DELETE, UPDATE as declare @a1 varchar(20), @a2 real, @a3 int, @in varchar(300); declare @ins int = (select count(*) from inserted), @del int = (select count(*) from deleted); if @ins > 0 and @del = 0 begin print 'Событие: INSERT'; set @a1 = (select [Наименование] from INSERTED); set @a2 = (select [Цена] from INSERTED); set @a3 = (select [Количество] from INSERTED); set @in = @a1+' '+cast(@a2 as varchar(20))+' '+cast(@a3 as varchar(20)); insert into TR_Tov(ST, TRN, C) values('INS', 'TRIG_Tov', @in); end; else if @ins = 0 and @del > 0 begin</pre>

```
print 'Событие: DELETE';
set @a1 = (select [Наименование] from deleted);
set @a2 = (select [Цена] from deleted);
set @a3 = (select [Количество] from deleted);
set @in = @a1+' '+cast(@a2 as varchar(20))+' '+cast(@a3 as varchar(20));
insert into TR_Tov(ST, TRN, C) values('DEL', 'TRIG_Tov', @in);
end;
else
if @ins > 0 and @del > 0
begin
print 'Событие: UPDATE';
set @a1 = (select [Наименование] from inserted);
set @a2 = (select [Цена] from inserted);
set @a3 = (select [Количество] from inserted);
set @in = @a1+' '+cast(@a2 as varchar(20))+' '+cast(@a3 as varchar(20));
set @a1 = (select [Наименование] from deleted);
set @a2 = (select [Цена] from deleted);
set @a3 = (select [Количество] from deleted);
set @in = @a1+' '+cast(@a2 as varchar(20))+' '+cast(@a3 as
varchar(20))+ ''+@in;
insert into TR_Tov(ST, TRN, C) values('UPD', 'TRIG_Tov', @in);
end;
return;
```

	<p>DML-операторы изменения таблицы Товары:</p> <pre>insert into Товары(Наименование, Цена, Количество) values('Стол', 140, 20); delete from Товары where Наименование = 'Стол'; update Товары set Количество = 20 where Наименование = 'Стул';</pre> <p>Оператор select * from TR_Tov позволяет проверить содержимое TR_Tov.</p>
5. Разработать сценарий, который демонстрирует на примере базы данных UNIVER, что проверка ограничения целостности выполняется до срабатывания AFTER-триггера.	<p>Важной особенностью AFTER-триггера является то, что он вызывается после выполнения активизирующего его оператора. Поэтому, если оператор нарушает ограничение целостности, то возникшая ошибка не допускает выполнения этого оператора и соответствующих триггеров.</p> <pre>alter table Товары add constraint Цена check(Цена >= 15) go update Товары set Цена = 10 where Наименование = 'Стул';</pre>
6. Создать для таблицы TEACHER три AFTER-триггера с именами: TR_TEACHER_DEL1 , TR_TEACHER_DEL2 и TR_TEACHER_DEL3 . Триггеры должны реагировать на событие DELETE и формировать соответствующие строки в таблицу TR_AUDIT . Получить список триггеров таблицы	<p>Пусть для таблицы Товары имеется три триггера, реагирующих на событие UPDATE:</p> <pre>create trigger AUD_AFTER_UPDA on Товары after UPDATE as print 'AUD_AFTER_UPDATE_A'; return; go</pre>

TEACHER.

Упорядочить выполнение триггеров для таблицы **TEACHER**, реагирующих на событие **DELETE** следующим образом: первым должен выполняться триггер с именем **TR_TEACHER_DEL3**, последним – триггер **TR_TEACHER_DEL2**.

Использовать системные представления **SYS.TRIGGERS** и **SYS.TRIGGERS_EVENTS**, а также системную процедуру **SP_SETTRIGGERORDERS**.

```
create trigger AUD_AFTER_UPDB on Товары after UPDATE
```

```
    as print 'AUD_AFTER_UPDATE_B';
```

```
return;
```

```
go
```

```
create trigger AUD_AFTER_UPDC on Товары after UPDATE
```

```
    as print 'AUD_AFTER_UPDATE_C';
```

```
return;
```

```
go
```

Проверить порядок выполнения триггеров можно следующим запросом:

```
select t.name, e.type_desc
  from sys.triggers t join sys.trigger_events e
    on t.object_id = e.object_id
   where OBJECT_NAME(t.parent_id) = 'Товары' and
         e.type_desc = 'UPDATE' ;
```

Изменение порядка выполнения триггеров выполняется с помощью системных процедур:

```
exec SP_SETTRIGGERORDER @triggername = 'AUD_AFTER_UPDC',
                           @order = 'First', @stmttype = 'UPDATE';
```

```
exec SP_SETTRIGGERORDER @triggername = 'AUD_AFTER_UPDA',
                           @order = 'Last', @stmttype = 'UPDATE';
```

<p>7. Разработать сценарий, демонстрирующий на примере базы данных UNIVER утверждение: AFTER-триггер является частью транзакции, в рамках которого выполняется оператор, активизировавший триггер.</p>	<p>Пусть разработан триггер, который ограничивает общее количество товаров в таблице Товары некоторой величиной:</p> <pre>create trigger Tov_Tran on Товары after INSERT, DELETE, UPDATE as declare @c int = (select sum(Количество) from Товары); if (@c > 2000) begin raiserror('Общая количество товаров не может быть >2000', 10, 1); rollback; end; return;</pre> <p>Если попробовать обновить информацию с помощью оператора UPDATE, то это вызовет сообщение об ошибке и транзакция завершится аварийно:</p> <pre>update Товары set Количество = 1990 where Наименование = 'Стол'</pre>
<p>8. Для таблицы FACULTY создать INSTEAD OF-триггер, запрещающий удаление строк в таблице.</p> <p>Разработать сценарий, который демонстрирует на примере базы данных UNIVER, что проверка ограничения целостности выполнена, если есть INSTEAD OF-триггер.</p>	<p>INSTEAD OF-триггер, запрещающий удалять строки из таблицы Товары:</p> <pre>use ПРОДАЖИ go create trigger Tov_INSTEAD_OF on Товары instead of DELETE as raiserror ('Удаление запрещено', 10, 1); return;</pre>

<p>С помощью оператора DROP удалить все DML-триггеры, созданные в этой лабораторной работе.</p>	<p>Операция удаления строки не будет выполнена:</p> <pre>delete from Товары where Наименование = 'Стол';</pre>
<p>9. Создать DDL-триггер, реагирующий на все DDL-события в БД UNIVER.</p> <p>Триггер должен запрещать создавать новые таблицы и удалять существующие. Свое выполнение триггер должен сопровождать сообщением, которое содержит: тип события, имя и тип объекта, а также пояснительный текст, в случае запрещения выполнения оператора.</p> <p>Разработать сценарий, демонстрирующий работу триггера.</p>	<p>Пусть разработан триггер DDL_PRODAJI, который запрещает вносить изменения в базу данных ПРОДАЖИ:</p> <pre>use ПРОДАЖИ go create trigger DDL_PRODAJI on database for DDL_DATABASE_LEVEL_EVENTS as declare @t varchar(50) = EVENTDATA().value('/EVENT_INS- TANCE/EventType[1]', 'varchar(50)'); declare @t1 varchar(50) = EVENTDATA().value('/EVENT_INS- TANCE/ObjectName[1]', 'varchar(50)'); declare @t2 varchar(50) = EVENTDATA().value('/EVENT_INS- TANCE/ObjectType[1]', 'varchar(50)'); if @t1 = 'Товары' begin print 'Тип события: '+@t; print 'Имя объекта: '+@t1; print 'Тип объекта: '+@t2; raiserror(N'операции с таблицей Товары запрещены', 16, 1); rollback; end;</pre>

Если попытаться внести изменения в таблицу **Товары**, например,
alter table Товары Drop Column Количество;
то будет выведена соответствующая информация.

10. Разработать различные виды триггеров для базы данных **X_MyBASE** и продемонстрировать их работу.
- 11*. Создать таблицу **WEATHER** (город, начальная дата, конечная дата, температура). Создать триггер, проверяющий корректность ввода и изменения данных.
Например, если в таблице есть строка (Минск, 01.01.2022 00:00, 01.01.2022 23:59, -6), то в нее не может быть вставлена строка (Минск, 01.01.2022 00:00, 01.01.2022 23:59, -2). Временные периоды могут быть различными.

Тест "Применение DML-триггеров"

[В начало практикума](#)

Лабораторная работа № 16. Использование XML

XML (Extensible Markup Language) – расширяемый язык разметки. XML-формат часто используется для обмена данными между компонентами информационных систем. При работе с базами данных важными являются две задачи: преобразование *табличных данных в XML-структуры* и преобразование *XML-структур в строки реляционной таблицы*.

Задание	Краткие теоретические сведения
1. Разработать сценарий создания XML-документа в режиме PATH из таблицы TEACHER для преподавателей кафедры ИСиТ.	<p>Для преобразования результата SELECT-запроса в формат XML в операторе SELECT применяется секция FOR XML. При этом могут использоваться режимы RAW, AUTO, PATH.</p> <p>В режиме RAW в результате SELECT-запроса создается XML-фрагмент, состоящий из последовательности элементов с именем row. Каждый элемент row соответствует строке результирующего набора, имена его атрибутов совпадают с именами столбцов результирующего набора, а значения атрибутов равны их значениям. Чтобы раскрыть полностью XML-фрагмент в результирующем наборе надо по фрагменту дважды щелкнуть.</p> <pre>use ПРОДАЖИ go select p.Наименование 'Наименование_товара', p.Цена 'Цена_товара', t.Цена_продажи 'Цена продажи' from Товары p join Заказы t on p.Наименование = t.Наименование_товара where t.Заказчик = 'Луч' for xml RAW('Заказчик'), root('Список_товаров'), elements;</pre>

2. Разработать сценарий создания XML-документа в режиме AUTO на основе SELECT-запроса к таблицам **AUDITORIUM** и **AUDITORIUM_TYPE**, который содержит следующие столбцы: наименование аудитории, наименование типа аудитории и вместимость. Найти только лекционные аудитории.

Особенность режима AUTO проявляется в многотабличных запросах. В этом случае режим AUTO позволяет построить XML-фрагмент с применением вложенных элементов.

```
select [Заказчик].Заказчик [Заказчик],  
       [Товар].Наименование [Наименование_товара],  
       [Товар].Цена [Цена_товара]  
  from Товары [Товар] join Заказы [Заказчик]  
    on [Товар].Наименование = [Заказчик].Наименование_товара  
      where [Заказчик].Заказчик in ('Луч', 'Белвест')  
order by [Заказчик] for xml AUTO,  
root('Список_товаров'), elements;
```

При использовании режима PATH каждый столбец конфигурируется независимо с помощью псевдонима этого столбца.

```
select [Заказчик].Заказчик [Заказчик],  
       [Товар].Наименование [Наименование_товара],  
       [Товар].Цена [Цена_товара]  
  from Товары [Товар] join Заказы [Заказчик]  
    on [Товар].Наименование = [Заказчик].Наименование_товара  
      where [Заказчик].Заказчик in ('Луч', 'Белвест')  
order by [Заказчик] for xml PATH('Заказчик'),  
root('Список_товаров'), elements;
```

3. Разработать XML-документ, содержащий данные о трех новых учебных дисциплинах, которые следует добавить в таблицу **SUBJECT**.

Разработать сценарий, извлекающий данные о дисциплинах из XML-документа и добавляющий их в таблицу **SUBJECT**.

При этом применить системную функцию **OPENXML** и конструкцию **INSERT... SELECT**.

Пример преобразования XML-структуры в строки реляционной таблицы:

```
use ПРОДАЖИ
go
declare @h int = 0,
@x varchar(2000) = '<?xml version="1.0" encoding="windows-1251" ?>
<товары>
<товар="стол" цена="40" количество="5" />
<товар="стул" цена="10" количество="3" />
<товар="шкаф" цена="400" количество="1" />
</товары>';
exec sp_xml_preparedocument @h output, @x; -- подготовка документа
select * from openxml(@h, '/товары/товар', 0)
with([товар] nvarchar(20), [цена] real, [количество] int )
exec sp_xml_removedocument @h; -- удаление документа
```

Для преобразования XML-данных в строки таблицы предназначена функция **OPENXML**, которая принимает три входных параметра: дескриптор, выражение XPATH и целое положительное число, определяющее режим работы функции.

Дескриптор определяется процедурой **SP_XML_PREPAREDOCUMENT**, которая должна быть выполнена до **SELECT**-запроса, применяющего **OPENXML**. Процедура принимает в качестве входного параметра XML-документ (в формате строки) и возвращает дескриптор.

Выражение XPATH предназначено для выбора требуемых данных из исходного XML-документа.

	<p>Режим работы указывает на тип преобразования (0 – используется атрибутивная модель сопоставления, каждый XML-атрибут преобразовывается в столбец таблицы; 1 – аналогично типу 0, но для необработанных столбцов применяется сопоставление на основе элементов XML-документа; 2 – используется сопоставление на основе элементов, каждый элемент преобразовывается в столбец таблицы).</p> <p>С помощью выражения WITH должна быть указана структура формируемого результата.</p> <p>Для того, чтобы извлечь данные о товарах из XML-документа и добавить их в таблицу Товары, надо заменить оператор</p> <pre><code>select * from openxml(@h, '/товары/товар', 0) with([товар] nvarchar(20), [цена] real, [количество] int)</code></pre> <p>на</p> <pre><code>insert Товары select [товар], [цена], [количество] from openxml(@h, '/товары/товар', 0) with([товар] nvarchar(20), [цена] real, [количество] int)</code></pre>
<p>4. Используя таблицу STUDENT разработать XML-структуру, содержащую паспортные данные студента: серию и номер паспорта, личный номер, дата выдачи и адрес прописки.</p> <p>Разработать сценарий, в который включен оператор INSERT, добавляющий строку с</p>	<p>Пусть требуется создать таблицу Поставщики, содержащую адреса поставщиков товаров:</p> <pre><code>create table Поставщики (Организация nvarchar(50) primary key, Адрес xml -- столбец XML-типа);</code></pre>

XML-столбцом.

Включить в этот же сценарий оператор UPDATE, изменяющий столбец **INFO** у одной строки таблицы **STUDENT** и оператор SELECT, формирующий результирующий набор, аналогичный представленному на рисунке.

В SELECT-запросе использовать методы QUERY и VALUEXML-типа.

NAME	серия паспорта	номер паспорта	адрес
Манакова Анастасия Владимировна	MP	22223333	<адрес><страна>Беларусь</страна>
Махно Нестор Петрович	MP	1234555	<адрес><страна>Украина</страна>
Дыбенко Павел Ефимович	MP	55666777	<адрес><страна>Россия</страна>

В таблицу помещается информация о поставщиках, например,

insert into Поставщики (Организация, Адрес)

```
values ('Пинскдрев', '<адрес> <страна>Беларусь</страна>
        <город>Пинск</город> <улица>Кирова</улица>
        <дом>52</дом> </адрес>');
```

insert into Поставщики (Организация, Адрес)

```
values ('Минскдрев', '<адрес> <страна>Беларусь</страна>
        <город>Минск</город> <улица>Кальварийская</улица>
        <дом>35</дом> </адрес>');
```

Для обновления, например, номера дома можно использовать оператор UPDATE:

update Поставщики

```
set Адрес = '<адрес> <страна>Беларусь</страна>
            <город>Минск</город> <улица>Кальварийская</улица>
            <дом>45</дом> </адрес>'  
where Адрес.value('(/адрес/дом)[1]', 'varchar(10)') = 35;
```

Содержимое таблицы **Поставщики** показывает оператор SELECT:

select Организация,

Адрес.value('(/адрес/страна)[1]', 'varchar(10)') [страна],

Адрес.query('/адрес') [адрес]

from Поставщики;

5. Изменить (ALTER TABLE) таблицу **STUDENT** в базе данных **UNIVER** таким образом, чтобы значения *типовизированного* столбца с именем **INFO** контролировались коллекцией XML-схем (XML SCHEMACOLLECTION), представленной в правой части.

Разработать сценарии, демонстрирующие ввод и корректировку данных (операторы INSERT и UPDATE) в столбец **INFO** таблицы **STUDENT**, как содержащие ошибки, так и правильные.

Разработать другую XML-схему и добавить ее в коллекцию XML-схем в БД UNIVER.

Пусть имеется объект XML SCHEMACOLLECTION с именем **Student**:

```
use UNIVER
go
create xml schema collection Student as
N'<?xml version="1.0" encoding="utf-16" ?>
<xs:schema attributeFormDefault="unqualified"
    elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="студент">
<xs:complexType><xs:sequence>
<xs:element name="паспорт" maxOccurs="1" minOccurs="1">
<xs:complexType>
<xs:attribute name="серия" type="xs:string" use="required" />
<xs:attribute name="номер" type="xs:unsignedInt" use="required"/>
<xs:attribute name="дата" use="required" >
<xs:simpleType> <xs:restriction base ="xs:string">
<xs:pattern value="[0-9]{2}.[0-9]{2}.[0-9]{4}"/>
</xs:restriction> </xs:simpleType>
</xs:attribute> </xs:complexType>
</xs:element>
<xs:element maxOccurs="3" name="телефон" type="xs:unsignedInt"/>
<xs:element name="адрес"> <xs:complexType><xs:sequence>
<xs:element name="страна" type="xs:string" />
<xs:element name="город" type="xs:string" />
<xs:element name="улица" type="xs:string" />
```

```
<xs:element name="дом" type="xs:string" />
<xs:element name="квартира" type="xs:string" />
</xs:sequence></xs:complexType> </xs:element>
</xs:sequence></xs:complexType>
</xs:element>
</xs:schema>';
```

Документ XML-Schema, размещенный в коллекции **Student**, описывает XML-документ с корневым элементом **студент** (первый тэг **element**).

На втором уровне (внутри тега **студент**) расположено три тэга: **паспорт**, **телефон** и **адрес** (вложенные теги **element**). Причем тэг **паспорт** должен быть ровно один (атрибуты **maxOccurs** и **minOccurs**); тэг **телефон** является обязательным и может быть в количестве не более трех (атрибут **maxOccurs**); тэг **адрес** является обязательным, и количество таких тэгов не должно быть более одного.

Элементы третьего уровня (**страна**, **город**, **улица**, **дом**, **квартира**) присутствуют только внутри элемента **адрес**. Все эти элементы являются обязательными и должны присутствовать ровно один раз.

Данные в документе размещаются как значения атрибутов (теги **attribute**) или как значения, размещенные в теле элементов (**телефон**, **страна**, **город**, **улица**, **дом**, **квартира**).

Тип данных, размещаемых в атрибутах или теле элементов данных, определяется значением атрибута **type**.

Пример создания таблицы **STUDENT** с типизированной структурой:

drop table STUDENT;

```
go
create table STUDENT
(
    IDSTUDENT integer identity(1000,1) primary key,
    IDGROUP integer foreign key references GROUPS(IDGROUP),
    NAME nvarchar(100),
    BDAY date,
    STAMP timestamp,
    INFO xml(STUDENT), -- типизированный столбец XML-типа
    FOTO varbinary
);
```

6. Разработать сценарии, демонстрирующие использование XML для базы данных **X_MyBASE**.

7*. Разработать SELECT-запрос, формирующий XML-фрагмент такой же структуры, как фрагмент на рисунке ниже, содержащий описание структуры вуза, включающей перечень факультетов, кафедр и преподавателей.

Разработать сценарий, демонстрирующий работу SELECT-запроса.

Примечание: использовать подзапросы, режим PATH и ключевое слово TYPE (TYPE указывает на то, что формируемый XML-фрагмент следует рассматривать как вложенный).

```
<университет>
  <факультет код="ИДиП">
    <количество_кафедр>5</количество_кафедр>
    <кафедры>
      <кафедра код="БФ" />
      <кафедра код="ИСиТ">
        <преподаватели>
          <преподаватель код="СМЛВ">Смелов Владислав Владиславович</преподаватель>
          <!--.....-->
          <преподаватель код="МРС">Мороз Елена Станиславовна</преподаватель>
        </преподаватели>
      </кафедра>
      <кафедра код="ПОиСОИ">
        <преподаватели>
          <преподаватель код="ЮДНКВ">Юденков Виктор Степанович</преподаватель>
          <!--.....-->
        </кафедры>
      </кафедра>
    <!--.....-->
  </университет>
```

Тест "Использование XML "

[В начало практикума](#)

Таблицы базы данных UNIVER

Имя таблицы	Столбцы таблицы	Наименования столбцов, свойства столбцов
FACULTY (Факультеты)	FACULTY	код факультета, PK, char(10), not null
	FACULTY_NAME	наименование факультета, varchar(50), default ‘???’
PROFESSION (Специальности)	PROFESSION	код специальности, PK, char(20). not null
	FACULTY	код факультета, FK(FACULTY), char(10), not null
	PROFESSION_NAME	наименование специальности, varchar(100), null
	QUALIFICATION	квалификация, varchar(50), null
PULPIT (Кафедры)	PULPIT	код кафедры, PK, char(20), not null
	PULPIT_NAME	наименование кафедры, varchar(100), null
	FACULTY	код факультета, FK(FA-CULTY), char(10), not null
TEACHER (Преподаватели)	TEACHER	код преподавателя, PK, char(10), not null
	TEACHER_NAME	фамилия, имя, отчество преподавателя, varchar(100),null
	GENDER	пол, char(1), GENDER in ('м','ж')
	PULPIT	код кафедры, FK(PULPIT), char(10), not null
SUBJECT (Дисциплины)	SUBJECT	код дисциплины, PK, char(10), not null
	SUBJECT_NAME	наименование дисциплины, varchar(100), null, unique
	PULPIT	код кафедры, FK(PULPIT), char(20), not null
AUDITORIUM_	AUDITORIUM_TYPE	код типа аудитории, PK, char(10), not null

TYPE (Типы учебных аудиторий)	AUDITORIUM_TYPENAME	наименование типа аудитории, varchar(30), null
AUDITORIUM (Учебные аудитории)	AUDITORIUM	код аудитории, PK, char(20). not null
	AUDITORIUM_TYPE	код типа аудитории, FK(AUDITORIUM_TYPE), char(10), not null
	AUDITORIUM_CAPACITY	вместимость, int, default 1, check between 1 and 300
	AUDITORIUM_NAME	наименование аудитории, varchar(50), null
GROUP (Студенческие группы)	IDGROUP	идентификатор группы, PK, int, not null
	FACULTY	код факультета, FK(FACULTY), char(10), not null
	PROFESSION	код специальности, FK(PROFESSION) , char(20), not null
	YEAR_FIRST	год поступления, smallint, < текущий год +2
	COURSE	курс, вычисляемое поле, tinyint, вычисляется на основе текущей даты и значения YEAR_FIRST
STUDENT (Студент)	IDSTUDENT	Код студента, PK(STUDENT), int, identity (1000,1)
	IDGROUP	идентификатор группы, FK(GROUP), int, not null
	NAME	фамилия, имя, отчество, nvarchar(100)
	BDAY	дата рождения, date
	STAMP	штамп времени, timestamp
	INFO	дополнительная информация, xml, по умолчанию null
	FOTO	фотография, varbinary(max), по умолчанию null
PROGRESS (Оценки на экзамене)	SUBJECT	предмет, FK (SUBJECT), char(10)
	IDSTUDENT	Код студента, FK(STUDENT), int
	PDATE	дата экзамена, date
	NOTE	оценка, int (between 1 and 10)

Содержимое таблицы FACULTY

FACULTY	FACULTY_NAME
ТТЛП	Технологии и техника лесной промышленности
ТОВ	Технологии органических веществ
ХТИТ	Химические технологии и техника
ИЭФ	Инженерно-экономический
ЛХ	Лесохозяйственный
ИДиП	Издательское дело и полиграфия
ИТ	Информационных технологий

Содержимое таблицы PROFESSION

PROFESSION	FACULTY	PROFESSION_NAME	QUALIFICATION
1-36 06 01	ИДиП	Полиграфическое оборудование и системы обработки информации	инженер-электромеханик
1-36 07 01	ХТИТ	Машины и аппараты химических производств и предприятий строительных материалов	инженер-механик
1-40 01 02	ИТ	Информационные системы и технологии	инженер-программист-системотехник
1-46 01 01	ТТЛП	Лесоинженерное дело	инженер-технолог
1-47 01 01	ИДиП	Издательское дело	редактор-технолог
1-48 01 02	ТОВ	Химическая технология органических веществ, материалов и изделий	инженер-химик-технолог

1-48 01 05	ТОВ	Химическая технология переработки древесины	инженер-химик-технолог
1-54 01 03	ТОВ	Физико-химические методы и приборы контроля качества продукции	инженер по сертификации
1-75 01 01	ЛХФ	Лесное хозяйство	инженер лесного хозяйства
1-75 02 01	ЛХФ	Садово-парковое строительство	инженер садово-паркового строительства
1-89 02 02	ЛХФ	Туризм и природопользование	специалист в сфере туризма

Содержимое таблицы PULPIT

PULPIT	PULPIT_NAME	FACULTY
РИТ	Редакционно-издательских технологий	ИДиП
СБУАиА	Статистики, бухгалтерского учета, анализа и аудита	ИЭФ
ТДП	Технологий деревообрабатывающих производств	ТТЛП
ТиДИД	Технологии и дизайна изделий из древесины	ТТЛП
ТиП	Туризма и природопользования	ЛХФ
ТЛ	Транспорта леса	ТТЛП
ТНВиОХТ	Технологии неорганических веществ и общей химической технологии	ХТиТ

ТНХСиППМ	Технологии нефтехимического синтеза и переработки полимерных материалов	ТОВ
ХПД	Химической переработки древесины	ТОВ
ХТЭПиМЭЕ	Химии, технологии электрохимических производств и материалов электронной техники	ХТИТ
ЭТИМ	Экономической теории и маркетинга	ИЭФ

Содержимое таблицы TEACHER

TEACHER	TEACHER_NAME	GENDER	PULPIT
НСКВ	Носков Михаил Трофимович	NULL	ТТЛП
ПРКП	Прокопенко Николай Иванович	NULL	ТНХСиППМ
МРЗВ	Морозова Елена Степановна	NULL	ИСиТ
РВКС	Ровкас Андрей Петрович	NULL	ОВ
РЖКВ	Рыжиков Леонид Николаевич	NULL	ЛВ
РМНВ	Романов Дмитрий Михайлович	NULL	ИСиТ
СМЛВ	Смелов Владимир Владиславович	NULL	ИСиТ
КРЛВ	Крылов Павел Павлович	NULL	ИСиТ
ЧРН	Чернова Анна Викторовна	NULL	ХПД
МХВ	Мохов Михаил Сергеевич	NULL	ПОиСОИ

Содержимое таблицы SUBJECT

SUBJECT	SUBJECT_NAME	PULPIT
ПЗ	Представление знаний в компьютерных системах	ИСиТ
ПИС	Проектирование информационных систем	ИСиТ
ПСП	Программирование сетевых приложений	ИСиТ
ПЭХ	Прикладная электрохимия	ХТЭПиМЭЕ
СУБД	Системы управления базами данных	ИСиТ
ТиОЛ	Технология и оборудование лесозаготовок	ЛМиЛЗ
ТРИ	Технология резиновых изделий	ТНХСиППМ
ЭП	Экономика природопользования	МиЭП
ЭТ	Экономическая теория	ЭТиМ

Содержимое таблицы AUDITORIUM

AUDITORIUM	AUDITORIUM_TYPE	AUDITORIUM_CAPACITY	AUDITORIUM_NAME
301-1	ЛБ-К	15	301-1
304-4	ЛБ-К	90	304-4
313-1	ЛК-К	60	313-1
314-4	ЛК	90	314-4
320-4	ЛК	90	320-4
324-1	ЛК-К	50	324-1
413-1	ЛБ-К	15	413-1
423-1	ЛБ-К	90	423-1

Содержимое таблицы AUDITORIUM_TYPE

AUDITORIUM_TYPE	AUDITORIUM_TYPENAME
ЛБ-Х	Химическая лаборатория
ЛБ-К	Компьютерный класс
ЛБ-СК	Спец. компьютерный класс
ЛК	Лекционная
ЛК-К	Лекционная с уст. проектором

Содержимое таблицы GROUP

IDGROUP	FACULTY	PROFESSION	YEAR_FIRST
22	ЛХФ	1-75 02 01	2011
23	ЛХФ	1-89 02 02	2012
24	ЛХФ	1-89 02 02	2011
25	ТТЛП	1-36 05 01	2013
26	ТТЛП	1-36 05 01	2012
27	ТТЛП	1-46 01 01	2012
28	ИЭФ	1-25 01 07	2013
29	ИЭФ	1-25 01 07	2012
30	ИЭФ	1-25 01 07	2010
31	ИЭФ	1-25 01 08	2013
32	ИЭФ	1-25 01 08	2012

Содержимое таблицы STUDENT

IDSTUDENT	ID-GROUP	NAME	BDAY	STAMP	INFO	FOTO
1000	22	Пугач Михаил Трофимович	12/01/1996			
1001	23	Авдеев Николай Иванович	19/07/1996			
1002	24	Белова Елена Степановна	22/05/1996			
1003	25	Вилков Андрей Петрович	08/12/1996			
1004	26	Грушин Леонид Николаевич	11/11/1995			
1005	27	Дунаев Дмитрий Михайлович	24/08/1996			
1006	28	Клуни Иван Владиславович	15/09/1996			
1007	29	Крылов Олег Павлович	16/10/1996			

Содержимое таблицы PROGRESS

SUBJECT	IDSTUDENT	PDATE	NOTE
ОАиП	1000	12/01/2014	4
ОАиП	1001	19/01/2014	5
ОАиП	1003	08/01/2014	9
БД	1008	11/01/2014	8
БД	1010	15/01/2014	4
СУБД	1013	16/01/2014	7
СУБД	1014	27/01/2014	6

[В начало практикума](#)

Сценарии создания таблиц базы данных UNIVER

(данные произвольные)

```
drop table PROGRESS  
drop table STUDENT  
drop table GROUPS  
drop table SUBJECT  
drop table TEACHER  
drop table PULPIT  
drop table PROFESSION  
drop table FACULTY  
drop table AUDITORIUM  
drop table AUDITORIUM_TYPE
```

----- Создание и заполнение таблицы AUDITORIUM_TYPE

```
create table AUDITORIUM_TYPE  
( AUDITORIUM_TYPE char(10) constraint AUDITORIUM_TYPE_PK primary key,  
  AUDITORIUM_TYPENAME varchar(30)  
)  
insert into AUDITORIUM_TYPE (AUDITORIUM_TYPE, AUDITORIUM_TYPENAME) values ('ЛК', 'Лекционная');  
insert into AUDITORIUM_TYPE (AUDITORIUM_TYPE, AUDITORIUM_TYPENAME) values ('ЛБ-К', 'Компьютерный класс');  
insert into AUDITORIUM_TYPE (AUDITORIUM_TYPE, AUDITORIUM_TYPENAME) values ('ЛК-К', 'Лекционная с уст. проектором');  
insert into AUDITORIUM_TYPE (AUDITORIUM_TYPE, AUDITORIUM_TYPENAME) values ('ЛБ-Х', 'Химическая лаборатория');  
insert into AUDITORIUM_TYPE (AUDITORIUM_TYPE, AUDITORIUM_TYPENAME) values ('ЛБ-СК', 'Спец. компьютерный класс');
```

----- Создание и заполнение таблицы AUDITORIUM

```
create table AUDITORIUM
```

----- Создание и заполнение таблицы SUBJECT

```
create table SUBJECT  
( SUBJECT char(10) constraint SUBJECT_PK primary key,  
  SUBJECT_NAME varchar(100) unique,  
  PULPIT char(20) constraint SUBJECT_PULPIT_FK foreign key  
    references PULPIT(PULPIT)  
)  
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )  
  values ('СУБД', 'Системы управления базами данных', 'ИСиТ');  
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )  
  values ('БД', 'Базы данных', 'ИСиТ');  
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )  
  values ('ИНФ', 'Информационные технологии', 'ИСиТ');  
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )  
  values ('ОАиП', 'Основы алгоритмизации и программирования', 'ИСиТ');  
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )  
  values ('ПЗ', 'Представление знаний в компьютерных системах', 'ИСиТ');  
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )  
  values ('ПСП', 'Программирование сетевых приложений', 'ИСиТ');  
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )  
  values ('МСОИ', 'Моделирование систем обработки информации', 'ИСиТ');  
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )  
  values ('ПИС', 'Проектирование информационных систем', 'ИСиТ');  
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )  
  values ('КГ', 'Компьютерная геометрия', 'ИСиТ');  
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )  
  values ('ТИМАПЛ', 'Полиграф. машины, автоматы и поточные линии', 'ПОиСОИ');  
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )  
  values ('КМС', 'Компьютерные мультимедийные системы', 'ИСиТ');  
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )
```

```

( AUDITORIUM char(20) constraint AUDITORIUM_PK primary key,
  AUDITORIUM_TYPE  char(10) constraint AUDITORIUM_TYPE_FK foreign key
    references AUDITORIUM_TYPE(AUDITORIUM_TYPE),
  AUDITORIUM_CAPACITY integer constraint AUDITORIUM_CAPACITY_CHECK default 1 check (AUDITORIUM_CAPACITY between 1
and 300), -- вместимость
  AUDITORIUM_NAME  varchar(50)
)
insert into AUDITORIUM (AUDITORIUM, AUDITORIUM_NAME,
  AUDITORIUM_TYPE, AUDITORIUM_CAPACITY)
values ('206-1', '206-1','ЛБ-К', 15);
insert into AUDITORIUM (AUDITORIUM, AUDITORIUM_NAME,
  AUDITORIUM_TYPE, AUDITORIUM_CAPACITY)
values ('301-1', '301-1','ЛБ-К', 15);
insert into AUDITORIUM (AUDITORIUM, AUDITORIUM_NAME,
  AUDITORIUM_TYPE, AUDITORIUM_CAPACITY )
values ('236-1', '236-1','ЛК', 60);
insert into AUDITORIUM (AUDITORIUM, AUDITORIUM_NAME,
  AUDITORIUM_TYPE, AUDITORIUM_CAPACITY )
values ('313-1', '313-1','ЛК-К', 60);
insert into AUDITORIUM (AUDITORIUM, AUDITORIUM_NAME,
  AUDITORIUM_TYPE, AUDITORIUM_CAPACITY )
values ('324-1', '324-1','ЛК-К', 50);
insert into AUDITORIUM (AUDITORIUM, AUDITORIUM_NAME,
  AUDITORIUM_TYPE, AUDITORIUM_CAPACITY )
values ('413-1', '413-1','ЛБ-К', 15);
insert into AUDITORIUM (AUDITORIUM, AUDITORIUM_NAME,
  AUDITORIUM_TYPE, AUDITORIUM_CAPACITY )
values ('423-1', '423-1','ЛБ-К', 90);
insert into AUDITORIUM (AUDITORIUM, AUDITORIUM_NAME,
  AUDITORIUM_TYPE, AUDITORIUM_CAPACITY )
values ('408-2', '408-2','ЛК', 90);

```

-----Создание и заполнение таблицы FACULTY
create table FACULTY

```

values ('ОПП', 'Организация полиграф. производства','ПОиСОИ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME,PULPIT)
  values ('ДМ', 'Дискретная математика','ИСиТ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME,PULPIT )
  values ('МИ', 'Математическое программирование','ИСиТ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )
  values ('ЛЭВМ', 'Логические основы ЭВМ','ИСиТ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME,PULPIT )
  values ('ООП', 'Объектно-ориентированное программирование','ИСиТ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )
  values ('ЭП', 'Экономика природопользования','МиЭП');
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )
  values ('ЭТ', 'Экономическая теория','ЭТиМ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )
  values ('БЛЗиПсОО','Биология лесных зверей и птиц с осн. охотов. ','ОВ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )
  values ('ОСПиЛПХ','Основы садово-паркового и лесопаркового хозяйства',
'ЛПиСПС');
insert into SUBJECT (SUBJECT, SUBJECT_NAME,PULPIT )
  values ('ИГ', 'Инженерная геодезия ','ЛУ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME,PULPIT )
  values ('ЛВ', 'Лесоводство','ЛзиДВ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )
  values ('ОХ', 'Органическая химия','ОХ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME,PULPIT )
  values ('ТРИ', 'Технология резиновых изделий','ТНХСиППМ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )
  values ('ВТЛ', 'Водный транспорт леса','ТЛ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME,PULPIT )
  values ('ТиОЛ', 'Технология и оборудование лесозаготовок','ЛМиЛЗ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME,PULPIT )
  values ('ТОПИ', 'Технология обогащения полезных ископаемых
','ТНВиОХТ');
insert into SUBJECT (SUBJECT, SUBJECT_NAME, PULPIT )
  values ('ПЭХ', 'Прикладная электрохимия','ХТЭПиМЭ');

```

```

( FACULTY char(10) constraint FACULTY_PK primary key,
  FACULTY_NAME varchar(50) default '???'  

);  

insert into FACULTY (FACULTY, FACULTY_NAME )  

    values ('ХТиТ', 'Химическая технология и техника');  

insert into FACULTY (FACULTY, FACULTY_NAME )  

    values ('ЛХФ', 'Лесохозяйственный факультет');  

insert into FACULTY (FACULTY, FACULTY_NAME )  

    values ('ИЭФ', 'Инженерно-экономический факультет');  

insert into FACULTY (FACULTY, FACULTY_NAME )  

    values ('ТТЛП', 'Технология и техника лесной промышленности');  

insert into FACULTY (FACULTY, FACULTY_NAME )  

    values ('ТОБ', 'Технология органических веществ');  

insert into FACULTY (FACULTY, FACULTY_NAME )  

    values ('ИТ', 'Факультет информационных технологий');

-----Создание и заполнение таблицы PROFESSION  

create table PROFESSION  

( PROFESSION char(20) constraint PROFESSION_PK primary key,  

  FACULTY char(10) constraint PROFESSION_FACULTY_FK foreign key  

    references FACULTY(FACULTY),  

  PROFESSION_NAME varchar(100),  QUALIFICATION varchar(50)  

);
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ИТ', '1-40 01 02', 'Информационные системы и технологии', 'инженер-программист-системотехник' );
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ИТ', '1-47 01 01', 'Издательское дело', 'редактор-технолог' );
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ИТ', '1-36 06 01', 'Полиграфическое оборудование и системы обработки информации', 'инженер-электромеханик' );
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ХТиТ', '1-36 01 08', 'Конструирование и производство изделий из композиционных материалов', 'инженер-механик' );
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ХТиТ', '1-36 07 01', 'Машины и аппараты химических производств и предприятий строительных материалов', 'инженер-механик' );

```

-----Создание и заполнение таблицы GROUPS

```

create table GROUPS  

( IDGROUP integer identity(1,1) constraint GROUP_PK primary key,  

  FACULTY char(10) constraint GROUPS_FACULTY_FK foreign key  

    references FACULTY(FACULTY),  

  PROFESSION char(20) constraint GROUPS_PROFESSION_FK foreign key  

    references PROFESSION(PROFESSION),  

  YEAR_FIRST smallint check (YEAR_FIRST<=YEAR(GETDATE())),  

)
insert into GROUPS (FACULTY, PROFESSION, YEAR_FIRST )  

values ('ИДиП','1-40 01 02', 2013), --1  

      ('ИДиП','1-40 01 02', 2012),  

      ('ИДиП','1-40 01 02', 2011),  

      ('ИДиП','1-40 01 02', 2010),  

      ('ИДиП','1-47 01 01', 2013),---5 гр  

      ('ИДиП','1-47 01 01', 2012),  

      ('ИДиП','1-47 01 01', 2011),  

      ('ИДиП','1-36 06 01', 2010),----8 гр  

      ('ИДиП','1-36 06 01', 2013),  

      ('ИДиП','1-36 06 01', 2012),  

      ('ИДиП','1-36 06 01', 2011),  

      ('ИДиП','1-36 06 01', 2010),  

      ('ХТиТ','1-36 01 08', 2013),---12 гр  

      ('ХТиТ','1-36 01 08', 2012),  

      ('ХТиТ','1-36 07 01', 2011),  

      ('ХТиТ','1-36 07 01', 2010),  

      ('ТОБ','1-48 01 02', 2012), ---16 гр  

      ('ТОБ','1-48 01 02', 2011),  

      ('ТОБ','1-48 01 05', 2013),  

      ('ТОБ','1-54 01 03', 2012),  

      ('ЛХФ','1-75 01 01', 2013),--20 гр  

      ('ЛХФ','1-75 02 01', 2012),  

      ('ЛХФ','1-75 02 01', 2011),  

      ('ЛХФ','1-89 02 02', 2012),  

      ('ЛХФ','1-89 02 02', 2011),  

      ('ТТЛП','1-36 05 01', 2013),  

      ('ТТЛП','1-36 05 01', 2012),

```

```

insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ЛХФ', '1-75 01 01', 'Лесное хозяйство', 'инженер лесного хозяйства');
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ЛХФ', '1-75 02 01', 'Садово-парковое строительство', 'инженер садово-паркового строительства' );
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ЛХФ', '1-89 02 02', 'Туризм и природопользование', 'специалист в сфере туризма' );
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ИЭФ', '1-25 01 07', 'Экономика и управление на предприятиях', 'экономист-менеджер' );
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ИЭФ', '1-25 01 08', 'Бухгалтерский учет, анализ и аудит', 'экономист' );
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ТТЛП', '1-36 05 01', 'Машины и оборудование лесного комплекса', 'инженер-механик' );
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ТТЛП', '1-46 01 01', 'Лесоинженерное дело', 'инженер-технолог' );
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ТОВ', '1-48 01 02', 'Химическая технология органических веществ, материалов и изделий', 'инженер-химик-технолог' );
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ТОВ', '1-48 01 05', 'Химическая технология переработки древесины', 'инженер-химик-технолог' );
insert into PROFESSION(FACULTY, PROFESSION, PROFESSION_NAME, QUALIFICATION) values ('ТОВ', '1-54 01 03', 'Физико-химические методы и приборы контроля качества продукции', 'инженер по сертификации' );

```

-----Создание и заполнение таблицы PULPIT

```

create table PULPIT
( PULPIT char(20) constraint PULPIT_PK primary key,
  PULPIT_NAME varchar(100),
  FACULTY char(10) constraint PULPIT_FACULTY_FK foreign key
)

```

```

('ТТЛП','1-46 01 01', 2012),--27 гр
('ИЭФ','1-25 01 07', 2013),
('ИЭФ','1-25 01 07', 2012),
('ИЭФ','1-25 01 07', 2010),
('ИЭФ','1-25 01 08', 2013),
('ИЭФ','1-25 01 08', 2012) ---32 гр

```

-----Создание и заполнение таблицы STUDENT

```

create table STUDENT
( IDSTUDENT integer identity(1000,1) constraint STUDENT_PK primary key,
  IDGROUP integer constraint STUDENT_GROUP_FK foreign key
    references GROUPS(IDGROUP),
  NAME nvarchar(100),
  BDAY date,
  STAMP timestamp,
  INFO xml,
  FOTO varbinary
)
insert into STUDENT (IDGROUP,NAME, BDAY)
values (2, 'Силюк Валерия Ивановна', '12.07.1994'),
       (2, 'Сергель Виолетта Николаевна', '06.03.1994'),
       (2, 'Добродей Ольга Анатольевна', '09.11.1994'),
       (2, 'Подоляк Мария Сергеевна', '04.10.1994'),
       (2, 'Никитенко Екатерина Дмитриевна', '08.01.1994'),
       (3, 'Яцкевич Галина Иосифовна', '02.08.1993'),
       (3, 'Осадчая Эла Васильевна', '07.12.1993'),
       (3, 'Акулова Елена Геннадьевна', '02.12.1993'),
       (4, 'Плешкун Милана Анатольевна', '08.03.1992'),
       (4, 'Буянова Мария Александровна', '02.06.1992'),
       (4, 'Харченко Елена Геннадьевна', '11.12.1992'),
       (4, 'Крученок Евгений Александрович', '11.05.1992'),
       (4, 'Бороховский Виталий Петрович', '09.11.1992'),
       (4, 'Мацкевич Надежда Валерьевна', '01.11.1992'),
       (5, 'Логинова Мария Вячеславовна', '08.07.1995'),
       (5, 'Белько Наталья Николаевна', '02.11.1995'),
       (5, 'Селило Екатерина Геннадьевна', '07.05.1995'),

```

```

references FACULTY(FACULTY)
);
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY )
    values ('ИСиТ', 'Информационных систем и технологий ','ИТ' )
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
    values ('ЛВ', 'Лесоводства','ЛХФ')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
    values ('ЛУ', 'Лесоустройства','ЛХФ')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
    values ('ЛЗиДВ', 'Лесозащиты и древесиноведения','ЛХФ')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
    values ('ЛКиП', 'Лесных культур и почвоведения','ЛХФ')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
    values ('ТиП', 'Туризма и природопользования','ЛХФ')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
    values ('ЛПиСПС', 'Ландшафтного проектирования и садово-паркового
строительства','ЛХФ')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
    values ('ТЛ', 'Транспорта леса', 'ТТЛП')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
    values ('ЛМиЛЗ', 'Лесных машин и технологии лесозаготовок', 'ТТЛП')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
    values ('ТДП', 'Технологий деревообрабатывающих производств', 'ТТЛП')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
values ('ТиДИД', 'Технологии и дизайна изделий из древесины', 'ТТЛП')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
values ('ОХ', 'Органической химии', 'ТОВ')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
values ('ХПД', 'Химической переработки древесины', 'ТОВ')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
values ('ТНВиОХТ', 'Технологии неорганических веществ и общей химической
технологии ', 'ХТиТ')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
values ('ПиАХП', 'Процессов и аппаратов химических производств', 'ХТиТ')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
values ('ЭТиМ', 'Экономической теории и маркетинга', 'ИЭФ')

```

(5, 'Дрозд Анастасия Андреевна', '04.08.1995'), (6, 'Козловская Елена Евгеньевна', '08.11.1994'), (6, 'Потапнин Кирилл Олегович', '02.03.1994), (6, 'Равковская Ольга Николаевна', '04.06.1994'), (6, 'Ходоронок Александра Вадимовна', '09.11.1994'), (6, 'Рамук Владислав Юрьевич', '04.07.1994'), (7, 'Неруганенок Мария Владимировна', '03.01.1993'), (7, 'Цыганок Анна Петровна', '12.09.1993'), (7, 'Масилевич Оксана Игоревна', '12.06.1993'), (7, 'Алексиевич Елизавета Викторовна', '09.02.1993'), (7, 'Ватолин Максим Андреевич', '04.07.1993'), (8, 'Синица Валерия Андреевна', '08.01.1992'), (8, 'Кудряшова Алина Николаевна', '12.05.1992'), (8, 'Мигулина Елена Леонидовна', '08.11.1992'), (8, 'Шпиленя Алексей Сергеевич', '12.03.1992'), (9, 'Астафьев Игорь Александрович', '10.08.1995'), (9, 'Гайтюкевич Андрей Игоревич', '02.05.1995'), (9, 'Рученя Наталья Александровна', '08.01.1995'), (9, 'Тарасевич Анастасия Ивановна', '11.09.1995'), (10, 'Жоглин Николай Владимирович', '08.01.1994'), (10, 'Санько Андрей Дмитриевич', '11.09.1994'), (10, 'Пещур Анна Александровна', '06.04.1994'), (10, 'Бучалис Никита Леонидович', '12.08.1994')	insert into STUDENT (IDGROUP,NAME, BDAY) values (11, 'Лавренчук Владислав Николаевич', '07.11.1993'), (11, 'Власик Евгения Викторовна', '04.06.1993'), (11, 'Абрамов Денис Дмитриевич', '10.12.1993'), (11, 'Оленчик Сергей Николаевич', '04.07.1993'), (11, 'Савинко Павел Андреевич', '08.01.1993'), (11, 'Бакун Алексей Викторович', '02.09.1993'), (12, 'Бань Сергей Анатольевич', '11.12.1995'), (12, 'Сечайко Илья Александрович', '10.06.1995'), (12, 'Кузмичева Анна Андреевна', '09.08.1995'), (12, 'Бурко Диана Францевна', '04.07.1995'), (12, 'Даниленко Максим Васильевич', '08.03.1995'), (12, 'Зизюк Ольга Олеговна', '12.09.1995'),
---	---

```

insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
    values ('МиЭП', 'Менеджмента и экономики природопользования','ИЭФ')
insert into PULPIT (PULPIT, PULPIT_NAME, FACULTY)
    values ('СБУАиА', 'Статистики, бухгалтерского учета, анализа и аудита', 'ИЭФ')

```

----- Создание и заполнение таблицы TEACHER

```

create table TEACHER
( TEACHER char(10) constraint TEACHER_PK primary key,
TEACHER_NAME varchar(100),
GENDER char(1) CHECK (GENDER in ('м', 'ж')),
PULPIT char(20) constraint TEACHER_PULPIT_FK foreign key
    references PULPIT(PULPIT)
);
insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT )
    values ('СМЛВ', 'Смелов Владимир Владиславович', 'м', 'ИСиТ');
insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT )
    values ('ДТК', 'Дятко Александр Аркадьевич', 'м', 'ИВД');
insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT )
    values ('УРБ', 'Урбанович Павел Павлович', 'м', 'ИСиТ');
insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT )
    values ('ГРН', 'Турин Николай Иванович', 'м', 'ИСиТ');
insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT )
    values ('ЖЛК', 'Жиляк Надежда Александровна', 'ж', 'ИСиТ');
insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT )
    values ('MP3', 'Мороз Елена Станиславовна', 'ж', 'ИСиТ');
insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT )
    values ('БРТШВЧ', 'Барташевич Святослав Александрович', 'м', 'ПОиСОИ');
insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT )
    values ('APC', 'Арсентьев Виталий Арсентьевич', 'м', 'ПОиСОИ');
insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT )
    values ('HBPB', 'Неверов Александр Васильевич', 'м', 'МиЭП');
insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT )
    values ('РВКЧ', 'Ровкач Андрей Иванович', 'м', 'ЛВ');
insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT )
    values ('ДМДК', 'Демидко Марина Николаевна', 'ж', 'ЛПиСПС');
insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT )

```

```

(13, 'Шарапо Мария Владимировна', '08.10.1994'),
(13, 'Касперович Вадим Викторович', '10.02.1994'),
(13, 'Чупрыгин Арсений Александрович', '11.11.1994'),
(13, 'Воеводская Ольга Леонидовна', '10.02.1994'),
(13, 'Метушевский Денис Игоревич', '12.01.1994'),
(14, 'Ловецкая Валерия Александровна', '11.09.1993'),
(14, 'Дворак Антонина Николаевна', '01.12.1993'),
(14, 'Щука Татьяна Николаевна', '09.06.1993'),
(14, 'Коблинец Александра Евгеньевна', '05.01.1993'),
(14, 'Фомичевская Елена Эрнестовна', '01.07.1993'),
(15, 'Бесараб Маргарита Вадимовна', '07.04.1992'),
(15, 'Бадуро Виктория Сергеевна', '10.12.1992'),
(15, 'Тарасенко Ольга Викторовна', '05.05.1992'),
(15, 'Афанасенко Ольга Владимировна', '11.01.1992'),
(15, 'Чуйкевич Ирина Дмитриевна', '04.06.1992'),
(16, 'Брель Алексеевна', '08.01.1994'),
(16, 'Кузнецова Анастасия Андреевна', '07.02.1994'),
(16, 'Томина Карина Геннадьевна', '12.06.1994'),
(16, 'Дуброва Павел Игоревич', '03.07.1994'),
(16, 'Шпаков Виктор Андреевич', '04.07.1994'),
(17, 'Шнейдер Анастасия Дмитриевна', '08.11.1993'),
(17, 'Шыгина Елена Викторовна', '02.04.1993'),
(17, 'Клюева Анна Ивановна', '03.06.1993'),
(17, 'Доморад Марина Андреевна', '05.11.1993'),
(17, 'Линчук Михаил Александрович', '03.07.1993'),
(18, 'Васильева Дарья Олеговна', '08.01.1995'),
(18, 'Щигельская Екатерина Андреевна', '06.09.1995'),
(18, 'Сазонова Екатерина Дмитриевна', '08.03.1995'),
(18, 'Бакунович Алина Олеговна', '07.08.1995')

```

----- Создание и заполнение таблицы PROGRESS

```

create table PROGRESS
( SUBJECT char(10) constraint PROGRESS_SUBJECT_FK foreign key
    references SUBJECT(SUBJECT),
IDSTUDENT integer constraint PROGRESS_IDSTUDENT_FK foreign key
    references STUDENT(IDSTUDENT),

```

<pre> values ('БРГ', 'Бурганская Татьяна Минаевна', 'ж', 'ЛПиСПС'); insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT) values ('РЖК', 'Рожков Леонид Николаевич ', 'м', 'ЛВ'); insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT) values ('ЗВГЦВ', 'Звягинцев Вячеслав Борисович', 'м', 'ЛЗиДВ'); insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT) values ('БЗБРДВ', 'Безбородов Владимир Степанович', 'м', 'ОХ'); insert into TEACHER (TEACHER, TEACHER_NAME, GENDER, PULPIT) values ('НСКВЦ', 'Насковец Михаил Трофимович', 'м', 'ТЛ'); </pre>	<pre> PDATE date, NOTE integer check (NOTE between 1 and 10)) insert into PROGRESS (SUBJECT, IDSTUDENT, PDATE, NOTE) values ('ОАиП', 1001, '01.10.2013',8), ('ОАиП', 1002, '01.10.2013',7), ('ОАиП', 1003, '01.10.2013',5), ('ОАиП', 1005, '01.10.2013',4) insert into PROGRESS (SUBJECT, IDSTUDENT, PDATE, NOTE) values ('СУБД', 1014, '01.12.2013',5), ('СУБД', 1015, '01.12.2013',9), ('СУБД', 1016, '01.12.2013',5), ('СУБД', 1017, '01.12.2013',4) insert into PROGRESS (SUBJECT, IDSTUDENT, PDATE, NOTE) values ('КГ', 1018, '06.5.2013',4), ('КГ', 1019, '06.05.2013',7), ('КГ', 1020, '06.05.2013',7), ('КГ', 1021, '06.05.2013',9), ('КГ', 1022, '06.05.2013',5), ('КГ', 1023, '06.05.2013',6) </pre>
---	---

[В начало практикума](#)

Таблицы базы данных ПРОДАЖИ

Имя таблицы	Имя поля	Тип данных	Тип поля
Товары	Наименование	nvarchar(20)	Ключевое
	Цена	real	
	Количество	int	
Заказчики	Наименование фирмы	nvarchar(20)	Ключевое
	Адрес	nvarchar(50)	
	Расчетный счет	nvarchar(15)	
Заказы	Номер заказа	nvarchar(10)	Ключевое
	Наименование товара	nvarchar(20)	
	Цена продажи	real	
	Количество	int	
	Дата поставки	date	
	Заказчик	nvarchar(20)	

Сценарии создания таблиц базы данных ПРОДАЖИ

```
USE Master
CREATE database ПРОДАЖИ;
```

```
USE ПРОДАЖИ
GO
```

----- Создание и заполнение таблицы Товары

```
CREATE TABLE Товары
```

```
( Наименование nvarchar(20) primary key,
    Цена real,
    Количество int
);
```

----- Создание и заполнение таблицы Заказы

```
CREATE TABLE Заказы
(
    Номер_заказа int primary key,
    Наименование_товара nvarchar(20) foreign key
        references Товары (Наименование),
    Цена_продажи real,
    Количество int,
    Дата_поставки date,
    Заказчик nvarchar(20) foreign key
        references Заказчики(Наименование_фирмы)
)
```

```
insert into Товары (Наименование, Цена, Количество)
values ('Стол', 78, 10),
       ('Стул офисный', 12, 10),
       ('Диван', 400, 3),
       ('Шкаф', 450, 10),
       ('Скрепки', 5, 50),
       ('Бумага', 10, 30),
       ('Степлер', 4, 20)
```

----- Создание и заполнение таблицы Заказчики

```
CREATE TABLE Заказчики
(
    Наименование_фирмы nvarchar(20) primary key,
    Адрес nvarchar(50),
    Расчетный_счет nvarchar(20)
);
```

```
insert into Заказчики (Наименование_фирмы, Адрес, Расчетный_счет)
values ('Луч', 'Минск, ул. Короля, 3', '123456'),
       ('Белвест', 'Витебск, ул. Герцена, 20', '3423456'),
       ('Радуга', 'Минск, ул. Скорины, 19', '983456'),
       ('Zte', 'Смолевичи, ул. Кирова, 32', '883456')
```

```
insert into Заказы (Номер_заказа,Наименование_товара,Цена_продажи,
Количество,Дата_поставки,Заказчик)
values (1,'Шкаф книжный', 350, 10, '08.10.2014','Луч'),
       (2,'Стол', 78, 10, '01.12.2014','Луч'),
       (3,'Стул', 12, 10, '04.07.2014','Белвест'),
       (4,'Диван', 400, 3, '01.09.2014','Zte'),
       (5,'Шкаф', 450, 10, '08.10.2014','Луч'),
       (6,'Степлер', 5, 6, '01.09.2014','Радуга'),
       (7,'Диван', 400, 3, '01.09.2014','Zte'),
       (8,'Бумага', 4, 11, '08.10.2014','Луч'),
       (9,'Скрепки', 6, 3, '01.09.2014','Радуга')
```

[В начало практикума](#)