

1. Основные понятия теории баз данных: база данных, система управления базами данных, основные требования к информации в БД

База данных – совокупность взаимосвязанных данных.

Система управления базами данных – программная реализация технологии хранения, извлечения, обновления и обработки данных в базе данных.

Основные требования к информации БД:

- Полезность - уменьшает информационную энтропию системы
- Полнота информации - информации должно быть достаточно, чтобы осуществить качественное управление
- Точность
- Достоверность - заведомо ошибочные данные не должны храниться в базе данных
- Непротиворечивость
- Актуальность

2. Модели данных, основная терминология реляционных баз данных

Модели данных:

- Иерархическая – это модель данных, где используется представление базы данных в виде древовидной (иерархической) структуры.
- Сетевая – как древовидная, но предков может быть больше, чем один.
- Реляционная – модель данных, основанная на отношениях (на теории множеств).

Терминология реляционных баз данных:

- домен: множество;
- таблица: отношение;
- атрибут: имя столбца таблицы (имя домена);
- заголовок таблицы: множество всех атрибутов;
- кортеж: элемент отношения или строка таблицы
- Relation – отношение
- Отношение может быть представлено в виде двумерной таблицы
- Реляционная база данных представляет собой набор взаимосвязанных таблиц
- Все объекты разделяются на типы
- Объекты одного и того же типа имеют свой набор атрибутов
- Один из атрибутов однозначно идентифицирует объект в таблице – первичный ключ

3. Нормализация таблиц базы данных. Нормальные формы таблиц.

Нормализация данных – процесс преобразования таблиц базы данных к нормальной форме.

Чтобы таблица соответствовала 1-й нормальной форме (1NF), необходимо, чтобы все значения ее полей были неделимыми (атомарными) и не вычисляемыми, а все записи – уникальными (не должно быть полностью совпадающих строк, повторяющихся данных). Устранить повторяющиеся группы в отдельных таблицах, создать отдельную таблицу для каждого набора связанных данных, идентифицировать каждый набор связанных данных с помощью первичного ключа.

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.

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.

4. Язык SQL. Группы операторов SQL: DDL, DML, DCL, TCL.

Язык SQL (Structured Query Language, язык структурированных запросов) – специализированный язык, предназначенный для написания запросов к реляционной БД.

Операторы SQL:

DDL - Data Definition Language - язык определения данных

предназначены для создания, удаления и изменения объектов БД или сервера СУБД: **CREATE, DROP, ALTER**.

DML - Data Manipulation Language - язык манипулирования данными

предназначены для работы со строками таблиц: **INSERT, DELETE, SELECT, UPDATE**.

TCL - Transaction Control Language - язык управления транзакциями

предназначены для управления транзакциями: **BEGIN TRAN, SAVE TRAN, COMMIT TRAN, ROLLBACK TRAN**.

DCL - Data Control Language - язык управления данными

предназначены для управления процессом авторизации: **GRANT, REVOKE, DENY** (еще есть **LOCK / UNLOCK, SET LOCK MODE**).

5. Системные базы данных: master, msdb, model, tempdb.

Системная база данных	Назначение
master	Хранит все системные данные Database Engine, а также информацию о других БД.
msdb	Используется службами SQL Server Agent (выполнение заданий по расписанию), Database Mail (формирование уведомлений по электронной почте), а также хранит информацию о резервном копировании БД.
tempdb	Пространство для временных объектов Database Engine и пользовательских временных таблиц. База данных пересоздается при каждой перезагрузке
model	Шаблон, используемый при создании всех БД, управляемых экземпляром Database Engine.
resource	БД, используемая только для чтения. Содержит системные объекты экземпляра Database Engine. Файлы БД являются скрытыми и не отображаются в MSMS.

6. Структура файлов базы данных

Хранение данных на диске: страницы, экстененты, файлы, файловые группы.

Страницы – основная единица хранилища данных.

Размер страницы постоянен и составляет 8 Кбайт. Каждая страница имеет заголовок 96 байтов, в котором хранится системная информация.

Строки данных размещаются на странице сразу же после заголовка.

Виды страниц: страницы данных, страницы индексов.

Экстененты – физическая единица дискового пространства, используемая для выделения памяти для таблиц и индексов.

Размер экстенента составляет 8 последовательно расположенных страниц или 64 Кбайт.

Существует два следующих типа экстенентов:

- однородные экстененты – содержат данные одной таблицы или индекса;
- смешанные экстененты – могут содержать данные до восьми таблиц или индексов.

Свойства страниц данных: все типы страниц данных имеют фиксированный размер (8 Кбайт) и состоят из следующих частей: заголовка страницы, пространства для данных и таблицы смещений строк.

- Если в странице нет места для новой строки той же таблицы, эта строка сохраняется в следующей странице цепочки страниц.
- Для таблиц, которые имеют только столбцы фиксированного размера, в каждой странице сохраняется одинаковое количество строк.
- Если в таблице есть хотя бы один столбец переменной длины, количество строк в странице может колебаться, и в странице сохраняется столько строк, сколько поместится.

7. Создание файлов базы данных. Файловые группы. Создание таблиц в файловой группе. Дисковое хранение файлов базы данных. Страницы, экстенды.

Файлы базы данных:

- первичный файл (.mdf) - содержит сведения, необходимые для запуска базы данных, и ссылки на другие файлы в базе данных. В каждой базе данных имеется один первичный файл данных.
- вторичные файлы (.ndf) - необязательные определяемые пользователем файлы данных. Данные могут быть распределены на несколько дисков, в этом случае каждый файл записывается на отдельный диск.
- файлы журнала транзакций (.log) - журнал содержит информацию для восстановления базы данных. Для каждой базы данных должен существовать хотя бы один файл журнала.

Файловая группа — это способ организации файлов базы данных. По умолчанию для любой базы данных создается файловая группа PRIMARY, и все создаваемые файлы базы данных по умолчанию будут относиться именно к ней. При создании таблиц и индексов дисковая память для них автоматически отводится в файловой группе по умолчанию. Для размещения в другой файловой группе следует явно указывать ее имя в операторе CREATE, создающем таблицу или индекс.

create database DEZH_UNIVER on primary

8. Типы данных Microsoft SQL Server.

- BIT: хранит значение 0 или 1. Фактически является аналогом булевого типа в языках программирования. Занимает 1 байт.
- TINYINT: хранит числа от 0 до 255. Занимает 1 байт. Хорошо подходит для хранения небольших чисел.
- SMALLINT: хранит числа от -32 768 до 32 767. Занимает 2 байта
- INT: хранит числа от -2 147 483 648 до 2 147 483 647. Занимает 4 байта. Наиболее используемый тип для хранения чисел.

- **BIGINT**: хранит очень большие числа от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, которые занимают в памяти 8 байт.

- **DECIMAL**: хранит числа с фиксированной точностью. Занимает от 5 до 17 байт в зависимости от количества чисел после запятой.

Данный тип может принимать два параметра `precision` и `scale`: `DECIMAL(precision, scale)`.

Параметр `precision` представляет максимальное количество цифр, которые может хранить число. Это значение должно находиться в диапазоне от 1 до 38. По умолчанию оно равно 18.

Параметр `scale` представляет максимальное количество цифр, которые может содержать число после запятой. Это значение должно находиться в диапазоне от 0 до значения параметра `precision`. По умолчанию оно равно 0.

- **NUMERIC**: данный тип аналогичен типу **DECIMAL**.

- **SMALLMONEY**: хранит дробные значения от -214 748.3648 до 214 748.3647. Предназначено для хранения денежных величин. Занимает 4 байта. Эквивалентен типу `DECIMAL(10,4)`.

- **MONEY**: хранит дробные значения от -922 337 203 685 477.5808 до 922 337 203 685 477.5807. Представляет денежные величины и занимает 8 байт. Эквивалентен типу `DECIMAL(19,4)`.

- **FLOAT**: хранит числа от $-1.79E+308$ до $1.79E+308$. Занимает от 4 до 8 байт в зависимости от дробной части.

Может иметь форму определения в виде `FLOAT(n)`, где `n` представляет число бит, которые используются для хранения десятичной части числа (мантиссы). По умолчанию `n = 53`.

- **REAL**: хранит числа от $-340E+38$ to $3.40E+38$. Занимает 4 байта. Эквивалентен типу `FLOAT(24)`.

Типы данных, представляющие дату и время

- **DATE**: хранит даты от 0001-01-01 (1 января 0001 года) до 9999-12-31 (31 декабря 9999 года). Занимает 3 байта.

- **TIME**: хранит время в диапазоне от 00:00:00.0000000 до 23:59:59.9999999. Занимает от 3 до 5 байт.

Может иметь форму `TIME(n)`, где `n` представляет количество цифр от 0 до 7 в дробной части секунд.

- **DATETIME**: хранит даты и время от 01/01/1753 до 31/12/9999. Занимает 8 байт.

- **DATETIME2**: хранит даты и время в диапазоне от 01/01/0001 00:00:00.0000000 до 31/12/9999 23:59:59.9999999. Занимает от 6 до 8 байт в зависимости от точности времени.

Может иметь форму `DATETIME2(n)`, где `n` представляет количество цифр от 0 до 7 в дробной части секунд.

- **SMALLDATETIME**: хранит даты и время в диапазоне от 01/01/1900 до 06/06/2079, то есть ближайшие даты. Занимает от 4 байта.

- **DATETIMEOFFSET**: хранит даты и время в диапазоне от 0001-01-01 до 9999-12-31. Сохраняет детальную информацию о времени с точностью до 100 наносекунд. Занимает 10 байт.

Строковые типы данных

- **CHAR**: хранит строку длиной от 1 до 8 000 символов. На каждый символ выделяет по 1 байту. Не подходит для многих языков, так как хранит символы не в кодировке Unicode.

Количество символов, которое может хранить столбец, передается в скобках. Например, для столбца с типом **CHAR(10)** будет выделено 10 байт. И если мы сохраним в столбце строку менее 10 символов, то она будет дополнена пробелами.

- **VARCHAR**: хранит строку. На каждый символ выделяется 1 байт. Можно указать конкретную длину для столбца - от 1 до 8 000 символов, например, **VARCHAR(10)**. Если строка должна иметь больше 8000 символов, то задается размер **MAX**, а на хранение строки может выделяться до 2 Гб: **VARCHAR(MAX)**.

Не подходит для многих языков, так как хранит символы не в кодировке Unicode.

В отличие от типа **CHAR** если в столбец с типом **VARCHAR(10)** будет сохранена строка в 5 символов, то в столце будет сохранено именно пять символов.

- **NCHAR**: хранит строку в кодировке Unicode длиной от 1 до 4 000 символов. На каждый символ выделяется 2 байта. Например, **NCHAR(15)**

- **NVARCHAR**: хранит строку в кодировке Unicode. На каждый символ выделяется 2 байта. Можно задать конкретный размер от 1 до 4 000 символов: . Если строка должна иметь больше 4000 символов, то задается размер **MAX**, а на хранение строки может выделяться до 2 Гб.

Бинарные типы данных

- **BINARY**: хранит бинарные данные в виде последовательности от 1 до 8 000 байт.

- **VARBINARY**: хранит бинарные данные в виде последовательности от 1 до 8 000 байт, либо до $2^{31}-1$ байт при использовании значения **MAX** (**VARBINARY(MAX)**).

9. Таблицы. Создание, изменение и удаление таблиц

Процесс проектирования логической схемы реляционной БД заключается в представлении данных в виде набора логически связанных таблиц. Таблицы нормализованы (см. вопрос Нормализация)

1. Создание. Таблица в БД создается с помощью DDL-оператора **CREATE TABLE**.

```

create table AUDITORIUM -- аудитории вуза
(
    AUDITORIUM          char(20)  -- идентификатор аудитории (304-1, 401-4,...)
                        constraint AUDITORIUM_PK primary key,
    AUDITORIUM_TYPE      char(10)  -- тип аудитории (ЛК, ЛБ-К, ...)
                        constraint AUDITORIUM_AUDITORIUM_TYPE_FK foreign key
                        references AUDITORIUM_TYPE (AUDITORIUM_TYPE),
    AUDITORIUM_CAPACITY  int        -- вместимость (макс. кол. слушателей)
                        constraint AUDITORIUM_CAPACITY_CHECK default 1
                        check (AUDITORIUM_CAPACITY between 1 and 300),
    AUDITORIUM_NAME      varchar(50) -- текстовое наименование аудитории (комментарий)
) on FG1; -- в файловой группе FG1

```

Таблица AUDITORIUM включает четыре столбца с именами: AUDITORIUM, AUDITORIUM_TYPE, AUDITORIUM_CAPACITY, AUDITORIUM_NAME. Следует обратить внимание на то, что таблица будет размещена в файловой группе с именем FG1.

Синтаксис оператора CREATE TABLE допускает другую форму записи ограничения:

```

create table FACULTY -- факульт
(
    FACULTY          char(10),      -- идентификатор факультета
    FACULTY_NAME      varchar(50),  -- полное наименование
    constraint PK_FACULTY_FACULTY primary key (FACULTY)
);

```

Пример создания таблицы с составным первичным ключом. В этом случае вторая форма записи ограничения является единственно возможной:

```

create table SHEDULE_TEACHER -- расписание преподавателя
(
    [DATETIME]        smalldatetime, -- дата и время занятий
    TEACHER            char(10),       -- преподаватель
    [SUBJECT]          char(10),       -- дисциплина
    AUDITORIUM         char(10),       -- аудитория
    constraint PK_SHEDULE_TEACHER primary key ([DATETIME], TEACHER)
) on FG2

```

2. Изменение. Модификация таблицы — изменение ее структуры. Можно добавлять/удалять столбцы и ограничения, для числовых данных изменять точность/тип, а для символьных — их размерность. Вообще можно просто удалить таблицу (DROP TABLE) и создать ее (CREATE TABLE) с

новой структурой. Но иногда это нерационально, например, когда уже распределены права доступа или когда в таблице дохуя данных (но не всякая модификация сохранит эти данные).

Перед модификацией таблицы целесообразно ознакомиться с ее структурой. Обычно для этого применяют системную процедуру **SP_HELP**, при вызове которой указывают в качестве параметра имя исследуемой таблицы: *exec SP_HELP TEACHER*

Изменяем таблицы при помощи Alter Table. В примере оператор ALTER изменяет существующий столбец **TEACHER_NAME**: уменьшает максимальный размер столбца до 50 символов и добавляет ограничение NOT NULL. Второй оператор добавляет новый столбец с именем **BDATE** типа DATE, не допускающий значений NULL и с заданным значением по умолчанию.

```
use BSTU
go
alter table TEACHER -- изменение столбца
    alter column TEACHER_NAME varchar(50) not null;
alter table TEACHER -- добавление нового столбца
    add BDATE date not null default '19600101'
```

3. Удаление. Таблицу можно удалить с помощью оператора DROP TABLE:

drop table TEACHERS

Проблема с удалением таблицы может быть в трех следующих случаях.

1. Пользователь не имеет достаточных прав на удаление таблицы.
2. Таблица заблокирована транзакцией другого сеанса. (нужно подождать, пока транзакция отработает)
3. На первичный ключ удаляемой таблицы ссылается внешний ключ другой таблицы. (нужно удалить ограничение целостности или эти самые таблицы)

10. Таблицы. Локальные и глобальные временные таблицы.

Основное отличие временных таблиц от постоянных в том, что они хранятся в системной БД **TEMPDB** и не могут иметь внешние ключи. Как правило, временные таблицы создаются для временного хранения результатов

SELECT-запросов. При применении временных таблиц следует помнить, что БД **TEMPDB** снова создается при каждом перезапуске сервера, поэтому сохранить или восстановить временную таблицу в случае сбоя, приведшего к перезапуску сервера СУБД, невозможно.

Существует два вида временных таблиц: локальные и глобальные. Они отличаются друг от друга форматом имени, областью видимости и жизненным циклом.

Локальные временные таблицы имеют имена, начинающиеся с символа #, доступны только создавшему ее пользователю и могут быть удалены с помощью оператора DROP TABLE. Если пользователь временную таблицу не удалил сам, то она удалится автоматически при его отключении.

Как правило, локальные временные таблицы применяются для временного хранения результатов трудоемких SELECT-запросов.

Глобальные временные таблицы имеют имена, начинающиеся с символа ##, доступны всем пользователям, подключенным к серверу, и могут быть удалены с помощью оператора DROP TABLE. Если глобальная временная таблица не удалена одним из пользователей, то она удалится автоматически при отключении всех пользователей, которые работали с этой таблицей. Если таблица использовалась только создавшим ее пользователем, то она будет удалена сразу после его отключения. Обычно глобальные временные таблицы применяются для обмена данными между несколькими сеансами.

11. Ограничения целостности. Создание, изменение и удаление ограничений целостности.

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. Может быть задано имя. Если это имя не задано, при создании таблицы сервер назначает ограничениям собственные имена.

Условное обозначение ограничения	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 ; предотвращает не согласованные операции между этими таблицами
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

Alter table => Drop, alter, add constraint. Constraint в столбцах таблицы.

12. Операторы DML: SELECT, INSERT, UPDATE, DELETE

Операторы DML предназначены для работы с одним (наиболее важным) типом объектов БД – таблицами. DML включает четыре оператора: SELECT, INSERT, DELETE, UPDATE. Иногда к этой группе относят оператор TRUNCATE (операция мгновенного удаления всех строк в таблице).

Наиболее мощным DML-оператором является SELECT. Он позволяет выбрать множество строк из одной или нескольких таблиц. При успешном выполнении этого оператора формируется результирующий набор, представляющий собой множество однотипных (с одинаковыми столбцами) строк. В общем случае результирующий набор может содержать ни одной, одну или более строк.

Любой оператор SELECT содержит список, определяющий перечень столбцов (в общем случае и содержимое) результирующего набора. Дополнительная часть оператора описывает множество строк из одной или

нескольких таблиц, служащее источником для формирования результирующего набора.

select *список* дополнение

Добавить одну или несколько строк в существующую таблицу можно с помощью оператора **INSERT**. Ключевое слова **INTO** указывает на то, что далее следует имя таблицы, в которую будут добавляться строки. Далее структура оператора и примеры использования

insert into *таблица* дополнение

Можно вставлять данные только в определенные столбцы, можно во все столбцы и не указывать, куда вставляем. Можно вставлять одну строчку, а можно много. Можно вставлять при помощи select.

```
insert into AUDITORIUM_TYPE (AUDITORIUM_TYPE, AUDITORIUM_TYPENAME )
values ('ЛК', 'Лекционная');
insert into GROUPS (FACULTY, PROFESSION, YEAR_FIRST )
values ('ИдиП', '1-40 01 02', 2013),
('ИдиП', '1-40 01 02', 2012),
('ИдиП', '1-40 01 02', 2011);
insert into TTTT(PN, FY)
select PULPIT_NAME, FACULTY from PULPIT;
insert AUDITORIUM_TYPE values ('ЛБ-Ф', 'Лаборатория физики');
```

Для удаления строк из таблицы предназначен оператор **DELETE**. Можно удалить все, а можно указать условие в секции where

```
delete from TTTT;
delete SUBJECT where PULPIT = 'ЛЗидВ'
delete from TEACHER where TEACHER_NAME like '%СМ%'
```

delete from *таблица* дополнение

Для изменения строк таблицы предназначен оператор **UPDATE**.
Структура и примеры использования:

update таблица дополнение

```
update AUDITORIUM set AUDITORIUM_CAPACITY *=1.15 where AUDITORIUM_TYPE = 'ЛБ-К';  
update TEACHER set PULPIT = 'ИСИТ';
```

13. Операторы DDL: CREATE, ALTER, DROP

Операторы DDL предназначены для создания, удаления и изменения объектов БД (таблицы, функции, процедуры, индексы и т.д.) или сервера СУБД. DDL включает три оператора: **CREATE**, **ALTER**, **DROP**

Оператор **CREATE** предназначен для создания объектов БД или сервера СУБД. Структура оператора:

create тип имя дополнение

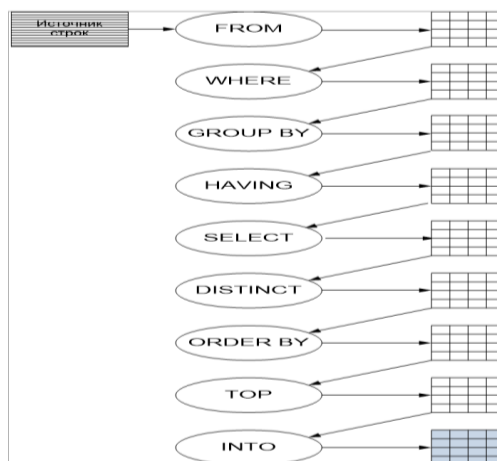
Для модификации существующих объектов БД или сервера СУБД применяется оператор **ALTER**. Структура оператора:

alter тип имя дополнение

Удалить существующий объект сервера или БД можно с помощью оператора **DROP**. Структура оператора :

drop тип имя

14. Оператор SELECT. Секции FROM, WHERE, GROUP BY, HAVING, ORDER BY, DISTINCT, TOP.



- **SELECT** определяет список возвращаемых столбцов (как существующих, так и вычисляемых), их имена, ограничения на уникальность строк в возвращаемом наборе, ограничения на количество строк в возвращаемом наборе;

- **FROM** задаёт табличное выражение, которое определяет базовый набор данных для применения операций, определяемых в других предложениях оператора;

- **WHERE** задает ограничение на строки табличного выражения из предложения **FROM**;

- **GROUP BY** объединяет ряды, имеющие одинаковое свойство с применением агрегатных функций

- **HAVING** выбирает среди групп, определённых параметром **GROUP BY**

- **DISTINCT** позволяет не выводить повторяющиеся строки

- **ORDER BY** задает критерии сортировки строк; отсортированные строки передаются в точку вызова.

- **TOP** ограничивает количество результирующих строк

- **INTO** позволяет создать новую таблицу и заполнить ее результатом SQL запроса

15. Подзапросы. Конструкции IN, EXISTS, ALL, ANY, NOT.

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.

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. (В скобках может быть запрос, а может быть просто вручную заданное множество:

in ('man', 'women'))

```

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%'))

```

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

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

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

Not — инвертирует логическое значение.

16. Оператор SELECT. 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.

```

use BSTU
go
select PDATE      [дата],
       [SUBJECT]  [дисциплина],
       count(*)   [количество студентов],
       max(NOTE)  [максимальная оценка]
from PROGRESS group by PDATE, [SUBJECT]

```

дата	дисциплина	количество студентов	максимальная оценка
2013-06-12	БД	5	9
2013-06-15	КГ	6	9
2013-01-10	ОАиП	7	8
2013-01-19	ОХ	6	9
2013-01-18	СУБД	5	9
2013-01-15	ЭТ	5	9

Рис. 7.91. Группировка строк таблицы **PROGRESS** по двум столбцам **PDATE**, **SUBJECT**

```

use BSTU
go
select month(BDAY) [месяц],
       count(*)    [кол.родившихся в этот месяц],
       100*count(*)/(select count(*) from STUDENT) [%]
from STUDENT group by month(BDAY)

```

месяц	кол.родившихс...	%
1	6	3
2	14	7
3	11	6
4	18	10
5	12	6
6	8	4
7	18	10
8	30	16
9	15	8
10	11	6
11	16	8
12	19	10

Рис. 7.92. Группировка строк таблицы **STUDENT** по значению, возвращаемому функцией

Следует помнить, что если в SELECT-запросе применяются секции WHERE и GROUP BY, первой исполняется секция WHERE. Затем выполняется группировка результата фильтрации в соответствии с выражением, указанным в секции GROUP BY

Если в SELECT-запросе указана опция TOP, ограничивающая количество строк в окончательном результирующем наборе, следует помнить, что действие опции осуществляется после группировки.

Также можно использовать rollup, cube.

17. Оператор SELECT. Использование агрегатных функций.

Агрегатные функции SQL нужны, чтобы получать результирующее значение из нескольких строк.

Наиболее часто применяемые агрегатные функции

Наименование функции	Назначение
AVG	Вычисление среднего значения
COUNT	Вычисление количества строк
MAX	Вычисление максимального значения
MIN	Вычисление минимального значения
SUM	Вычисление суммы значений

Можно использовать агрегатные функции просто, тогда мы будем получать значение, вычисленное для всех строк результирующего набора.

```
SELECT min(Цена_продажи) [Минимальная цена],  
max(Цена_продажи) [Максимальная цена],  
count(*) [Количество товаров],  
sum(Цена_продажи) [Заказы на общую сумму]  
From Заказы
```

Можно использовать агрегатные функции вместе с Group By, тогда значение будет вычисляться отдельно для каждой группы строк.

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

18. Оператор SELECT. Группировка данных с использованием CUBE, ROLLUP

ROLLUP – оператор Transact-SQL, который формирует промежуточные итоги для каждого указанного элемента и общий итог.

```
SELECT otdel, god, sum(summa) as itog  
  
FROM test_table  
  
GROUP BY ROLLUP(otdel, god)
```

CUBE — оператор Transact-SQL, который формирует результаты для всех возможных перекрестных вычислений.

```
SELECT otdel, god, sum(summa) as itog  
  
FROM test_table  
  
GROUP BY CUBE(otdel, god)
```

19. Представления. Создание, изменение и удаление 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.

Создание:

```
CREATE VIEW [Название] AS SELECT * FROM TEST_TABLE
```

Удаление:

```
DROP VIEW [Название]
```

Изменение:

```
ALTER VIEW [Название] AS SELECT * FROM TEST_TABLE2
```

Использование:

```
SELECT * FROM [Название]
```

20. Представления. Операции DML над представлениями. Использование представлений с указанием WITH CHECK OPTION.

При создании представлений, позволяющих выполнять операции INSERT, DELETE и UPDATE, базовый SELECT-запрос должен удовлетворять правилам:

- * 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.

Чтобы операция вставки не могла осуществиться в том случае, когда информация не удовлетворяет условию, записанному в секции Where, то следует создавать представление с опцией WITH CHECK OPTION.

```
CREATE VIEW [Название] AS SELECT * FROM Test_table WHERE price
> 200 with check option;
```

```
INSERT [Название] VALUES ('я', 201)
```

21. Представления. Использование представлений с указанием SCHEMABINDING.

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

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

Любая попытка модифицировать структуру представлений или таблиц, на которые ссылается созданное таким образом представление, будет неудачной. Чтобы такие таблицы или представления можно было модифицировать (инструкцией ALTER) или удалять (инструкцией DROP), нужно удалить это представление или убрать из него предложение SCHEMABINDING.

Схема – это поименованный контейнер объектов БД, позволяющий разграничить объекты с одинаковыми именами.

22. Функции преобразования типов данных CAST и CONVERT.

```
CAST ([Аргумент] as [Тип_Данных])
PRINT 'Кол-во: ' + CAST(12 as varchar(3))
```

```
CONVERT([Тип_Данных], [Аргумент])
PRINT 'Кол-во: ' + CONVERT(varchar(3), 12)
```

Convert более крутой! он позволяет еще стили накладывать после преобразования

23. Операторы работы с множествами UNION (ALL), INTERSECT, EXCEPT

UNION - объединение без копий

UNION ALL - объединение с копиями

```
SELECT * FROM Товары WHERE Название = 'Коля'
UNION
```

```
SELECT * FROM Товары WHERE Название = 'Николя'
```

INTERSECT - пересечение двух исходных наборов

EXCEPT - разность двух исходных наборов

24. Соединение таблиц. Внутреннее соединение INNER JOIN

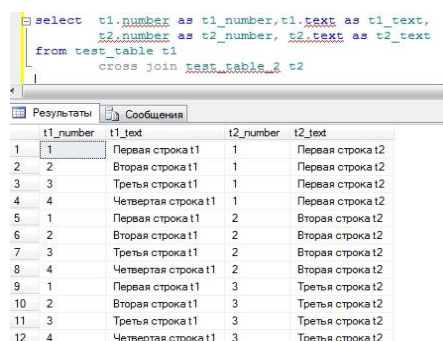
JOIN - позволяет извлекать данные более чем из одной таблицы. Самый простой вид соединения INNER JOIN – внутреннее соединение. Этот вид джойна выведет только те строки, если условие соединения выполняется (является истинным, т.е. TRUE). В запросах необязательно прописывать INNER – если написать только JOIN, то СУБД по умолчанию выполнить именно внутреннее соединение. Внутреннее соединение содержит только те строки одной таблицы, для которых имеются соответствующие строки в другой таблице.

Пример

```
SELECT * FROM student LEFT OUTER JOIN teacher ON teacher.Id =
student.Teacher
```

25. Ортогональное соединение CROSS JOIN

CROSS JOIN (декартово произведение) – это объединение SQL по которым каждая строка одной таблицы объединяется с каждой строкой другой таблицы. Количество строк $n \times m$.



```
select t1.number as t1_number, t1.text as t1_text,
       t2.number as t2_number, t2.text as t2_text
from test_table t1
cross join test_table_2 t2
```

	t1_number	t1_text	t2_number	t2_text
1	1	Первая строка t1	1	Первая строка t2
2	2	Вторая строка t1	1	Первая строка t2
3	3	Третья строка t1	1	Первая строка t2
4	4	Четвертая строка t1	1	Первая строка t2
5	1	Первая строка t1	2	Вторая строка t2
6	2	Вторая строка t1	2	Вторая строка t2
7	3	Третья строка t1	2	Вторая строка t2
8	4	Четвертая строка t1	2	Вторая строка t2
9	1	Первая строка t1	3	Третья строка t2
10	2	Вторая строка t1	3	Третья строка t2
11	3	Третья строка t1	3	Третья строка t2
12	4	Четвертая строка t1	3	Третья строка t2

26. Внешние соединения: LEFT(RIGHT) OUTER JOIN, FULL OUTER JOIN

Внешнее соединение OUTER JOIN двух таблиц формирует набор строк, состоящий из двух частей: результат внутреннего соединения двух таблиц и строки из одной из двух таблиц, которые не смогли соединиться. Значения в столбцах, соответствующих незаполненной (несоединенной) части строки будет NULL.

Имеется два вида внешнего соединения: LEFT OUTER JOIN – левое внешнее соединение и RIGHT OUTER JOIN – правое внешнее соединение.

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

FULL OUTER JOIN определяет объединение правого и левого соединения.

27. Язык T-SQL. Пакеты. Объявление переменных

Transact-SQL — процедурное расширение языка SQL, созданное компанией Microsoft и Sybase. SQL был расширен такими дополнительными возможностями как: управляющие операторы, локальные и глобальные переменные, различные дополнительные функции для обработки строк, дат, математики и т. п.

Пакет – это группа операторов T-SQL, которая обрабатывается сервером СУБД вместе.

```
---- пакет 1 -----
use TEMPDB
go
---- пакет 2 -----
create table #A ([Номер] int identity (1,1));
go
---- пакет 3 -----
insert into #A default values;
go 3          -- повторить 3 раза
---- пакет 4 -----
select * from #A;
go
---- пакет 5 -----
drop table #A;
```

Номер
1
2
3

Для объявления переменных, используемых в программах, предназначен оператор DECLARE:

```

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;

```

Имя переменной должно начинаться с символа @.

Областью видимости переменной являются все инструкции между ее объявлением и концом пакета или хранимой процедуры, где она объявлена.

28. Язык T-SQL. Операторы присвоения.

Переменную можно инициализировать в DECLARE. С помощью оператора SET можно переменной присвоить значение и выполнять вычисления. Оператор SELECT позволяет нескольким переменным присвоить значения.

29. Язык T-SQL. Операторы print, if-else, case

С помощью оператора PRINT можно вывести строку в стандартный выходной поток.

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

```

select case (select COUNT(*) from TEACHER where TEACHER_NAME LIKE '%Владимир%')
        when 0 then 'нет'
        when 1 then 'один'
        when 2 then 'два'
        else 'тьма'
        end 'Владимир'

```

Владимир
тьма

```

declare @x1 int = 0

set @x1 = (select SUM(AUDITORIUM_CAPACITY) from AUDITORIUM)

if @x1 > 200
    print 'общее количество мест больше 200'
else if @x1 > 100
    print 'общее количество мест от 100 до 200'
else if @x1 > 50
    print 'общее количество мест от 50 до 100'
else
    print 'общее количество мест меньше 50'
go

```

30. Язык T-SQL. Операторы begin-end, waitfor и return.

С помощью операторных скобок BEGIN END можно объединять операторы в группы.

```

declare @x1 int = 0

set @x1 = (select SUM(AUDITORIUM_CAPACITY) from AUDITORIUM)

if @x1 > 200
begin
    print 'общее количество мест больше 200'
    select CAST (COUNT(*) as numeric(5,3)) /
           CAST( (select COUNT(*) from AUDITORIUM) as numeric(5,3)) *
           100 as '%'
    from AUDITORIUM
    where AUDITORIUM_CAPACITY > (select AVG(AUDITORIUM_CAPACITY) from AUDITORIUM);
end
else if @x1 > 100
    print 'общее количество мест от 100 до 200'
else if @x1 > 50
    print 'общее количество мест от 50 до 100'
else
    print 'общее количество мест меньше 50'

```

%
58.333333300

WAITFOR блокирует выполнение пакета, хранимой процедуры или транзакции до тех пор, пока не истечет заданное время или интервал времени, либо указанная инструкция не изменит или не вернет хотя бы одну строку.


```

declare @ct char(8)
set @ct= (
    select CAST(DATEPART(hh,DATEADD(mi,1,SYSDATETIME())) as CHAR(2))
           +':'+
           CAST(DATEPART(mi,DATEADD(mi,1,SYSDATETIME())) as CHAR(2))
)
select SYSDATETIME(), @ct
waitfor time @ct
select SYSDATETIME()

```

(Отсутствует имя столбца)	(Отсутствует имя столбца)
2011-03-11 02:16:19.2777872	2:17

(Отсутствует имя столбца)
2011-03-11 02:17:00.0016880

```

select SYSDATETIME(), @ct
waitfor time @ct
select SYSDATETIME()

```

```

declare @i int = 0
select SYSDATETIME()
while @i < 5
begin
    waitfor delay '00:00:05'
    select SYSDATETIME()
    set @i = @i+1
end

```

2011-03-11 02:26:11.0584448
(Отсутствует имя столбца)
2011-03-11 02:26:16.0604000
(Отсутствует имя столбца)
2011-03-11 02:26:21.0613520
(Отсутствует имя столбца)
2011-03-11 02:26:26.0623040
(Отсутствует имя столбца)
2011-03-11 02:26:31.0632560
(Отсутствует имя столбца)
2011-03-11 02:26:36.0652112

Оператор RETURN служит для немедленного завершения работы пакета

```

print 'Метка 1';
return;           -- выход из пакета
print 'Метка 2'; -- не выполняется
go
print 'Метка 3';

```

31. Язык T-SQL. Оператор цикла while

Оператор WHILE предназначен для организации программного цикла. Принцип работы оператора WHILE такой же, как в большинстве алгоритмических языков (например C++, Java).

Оператор WHILE содержит две составляющие: логическое выражение и тело цикла. Логическое выражение определяет условие выполнения тела цикла. Тело цикла содержит один

```

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;

```

или более операторов, которые выполняются в том случае и до тех пор, пока логическое выражение принимает значение «истина».

32. Язык T-SQL. Обработка ошибок в конструкциях try-catch. Функция RAISERROR.

Для обработки ошибок выполнения в сценарии T-SQL предусмотрена конструкция, состоящая из двух блоков: TRY и CATCH. При этом блок TRY содержит код T-SQL, в котором могут возникнуть ошибки (говорят – охраняемый код), а блок CATCH – код, предназначенный для обработки ошибок. Ошибка, возникающая в охраняемом коде, приводит к немедленной 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

```

Разработчик сценария на языке T-SQL может сам сгенерировать ошибку с помощью специальной инструкции RAISERROR. В простейшем случае, при вызове инструкции можно передать три параметра: текстовое сообщение об ошибке, уровень серьезности ошибки и метку. Если уровень серьезности равен 11, то управление передается в блок обработки ошибок.

33. Язык T-SQL. Встроенные функции работы с датами

Для работы с датами и временем в языке T-SQL предусмотрен специальный набор встроенных функций. На рисунке представлен сценарий, демонстрирующий наиболее часто применяемые функции для работы с датами.

```

declare @dt datetime = getdate(), -- текущая дата
        @ed datetime = dateadd(d, 200, getdate()); -- текущая дата +200 дней
print 'Текущая дата' : '+' convert(varchar(12), @dt, 103 );
print 'Текущая дата+200 дней' : '+' convert(varchar(12), @ed, 103 );
print 'День' : '+' convert(varchar(12), day(@dt));
print 'День недели (1-вс)' : '+' convert(varchar(12), datepart(dw, @dt));
print 'Неделя' : '+' convert(varchar(12), datepart(wk, @dt));
print 'Месяц' : '+' convert(varchar(12), month(@dt));
print 'Квартал' : '+' convert(varchar(12), datepart(q, @dt));
print 'Год' : '+' convert(varchar(12), year(@dt));
print '+1 день' : '+' convert(varchar(12), dateadd(d, 1, @dt), 103);
print '+1 неделя' : '+' convert(varchar(12), dateadd(wk, 1, @dt), 103);
print '+1 месяц' : '+' convert(varchar(12), dateadd(m, 1, @dt), 103);
print '+1 квартал' : '+' convert(varchar(12), dateadd(q, 1, @dt), 103);
print '+1 год' : '+' convert(varchar(12), dateadd(y, 1, @dt), 103);
print 'Разница в днях' : '+' convert(varchar(12), datediff(d, @dt, @ed));
print 'Разница в неделях' : '+' convert(varchar(12), datediff(wk, @dt, @ed));
print 'Разница в месяцах' : '+' convert(varchar(12), datediff(m, @dt, @ed));
print 'Разница в годах' : '+' convert(varchar(12), datediff(yy, @dt, @ed));

```

```

Текущая дата : 05/03/2014
Текущая дата+200 дней : 21/09/2014
День : 5
День недели : 4
Неделя : 10
Месяц : 3
Квартал : 1
Год : 2014
+1 день : 06/03/2014
+1 неделя : 12/03/2014
+1 месяц : 05/04/2014
+1 квартал : 05/06/2014
+1 год : 06/03/2014
Разница в днях : 200
Разница в неделях : 29
Разница в месяцах : 6
Разница в годах : 0

```

34. Язык T-SQL. Встроенные функции работы со строками

```
print 'Подстрока          : '+' substring('1234567890', 3,2);
print 'Удалить пробелы слева : '+' ltrim('      67890');
print 'Удалить пробелы справа : '+' rtrim('12345      ') +'X';
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));
```

```
Подстрока          : 34
Удалить пробелы слева : X67890
Удалить пробелы справа : 12345X
Нижний регистр       : верхний регистр
Верхний регистр      : НИЖНИЙ РЕГИСТР
Заменить            : 1234X1234X
Строка пробелов     : X      X
Повторить строку     : 1212121212
Найти по шаблону     : 7
```

35. Язык T-SQL. Встроенные функции работы с числовыми данными

ROUND: округляет число. В качестве первого параметра передается число. Второй параметр указывает на длину. Если длина представляет положительное число, то оно указывает, до какой цифры после запятой идет округление. Если длина представляет отрицательное число, то оно указывает, до какой цифры с конца числа до запятой идет округление.

```
SELECT ROUND(1342.345, 2) -- 1342.350
```

```
SELECT ROUND(1342.345, -2) -- 1300.000
```

ISNUMERIC: определяет, является ли значение числом. В качестве параметра функция принимает выражение. Если выражение является числом, то функция возвращает 1. Если не является, то возвращается 0.

```
SELECT ISNUMERIC(1342.345) -- 1
```

```
SELECT ISNUMERIC('SQL') -- 0
```

SQUARE: возводит число в квадрат.

```
SELECT SQUARE(5) -- 25
```

SQRT: получает квадратный корень числа.

```
SELECT SQRT(225) -- 15
```

RAND: генерирует случайное число с плавающей точкой в диапазоне от 0 до 1.

```
SELECT RAND()          -- 0.707365088352935
```

```
SELECT RAND()          -- 0.173808327956812
```

Встроенные функции

```
print 'Округление      : '+ cast(round(12345.12345, 2) as varchar(12));
print 'Нижнее целое    : '+ cast(floor(24.5) as varchar(12));
print 'Верхнее целое   : '+ cast(ceiling(24.5) as varchar(12));
print 'Возведение в степень: '+ cast(power(12.0, 2) as varchar(12));
print 'Логарифм по ехр  : '+ 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));
print 'Рadiany         : '+ cast(radians(180.0) as varchar(20));
print 'Число пи        : '+ cast(pi() as varchar(12));
print 'Градусы         : '+ cast(degrees(pi()) as varchar(20));
print 'Синус           : '+ cast(sin(pi()) as varchar(12));
print 'Косинус         : '+ cast(cos(pi()) as varchar(12));
```

36. Курсоры. Объявление курсора. Общая схема работы с курсором: **declare, open, fetch, close, deallocate**

Курсор – программная конструкция, которая служит для хранения результата запроса и для обработки строк результирующего набора запись за записью. Механизм, позволяющий обрабатывать отдельные строки, полученные в результате select-запроса. Область памяти сервера, предназначенная для хранения и обработки результата select-запроса.

1.Курсор объявляется в операторе DECLARE.

2.Курсор открывается с помощью оператора OPEN.

3.С помощью оператора FETCH считывается одна или несколько строк результирующего набора, связанного с курсором SELECT-оператора, и обрабатывается нужным образом. Результат каждого считывания проверяется с помощью системной функции @@FETCH_STATUS.

4.Курсор закрывается оператором CLOSE.

5.Если курсор глобальный, то он должен быть освобожден с использованием оператора DEALLOCATE.

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.

```
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;
```

37. Курсоры. Типы курсоров: global/local, static/dynamic

Курсор – программная конструкция, которая служит для хранения результата запроса и для обработки строк результирующего набора запись за записью

Курсоры могут быть глобальными и локальными.

Локальный курсор может применяться в рамках одного пакета и ресурсы, выделенные ему при объявлении, освобождаются сразу после завершения работы пакета. Признаком того, что курсор является локальным, служит атрибут LOCAL, указанный при объявлении курсора

```
declare specialtiesCursor cursor local for....
```

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. Для того чтобы объявить глобальный курсор, следует применить атрибут GLOBAL

```
declare specialtiesCursor cursor global for....
```



```

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;

```

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 строка назад от текущей).

39. Курсоры. Функция fetch_status

После выполнения FETCH проверяется значение функции @@fetch_status. В зависимости от полученного результата цикл продолжается и считывается следующая строка, или цикл заканчивается.

@@FETCH_STATUS

- 0 – успешная выборка,
- 1 – вышли за диапазон таблицы,
- 2 – запись удалена после открытия курсора

40. Курсоры. Применение секции where current of в операторах update, delete

Курсоры 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. Такой формат операторов позволяет удалять или изменять строки в таблице, соответствующих текущей позиции курсора в результирующем наборе.

```

declare tmpcursor cursor local dynamic
for select val from #temp1 for update -- изменяем или удаляем строки
declare @str varchar(100) = ''
declare @line varchar(30) = ''
open tmpcursor
fetch tmpcursor into @line
set @str = @str + @line
while @@fetch_status = 0
begin
    fetch tmpcursor into @line
    set @str = @str + @line

```

```

                                delete #TEMP1 where current of tmpcursor
                                end
                                close tmpcursor
print @str
select * from #temp1
drop table #temp1
go

```

Курсоры – UPDATE CURRENT OF

```

declare @s char(10), @ps char(10), @p char(10), @n varchar(200)
declare ccc cursor local dynamic scroll
        for select subject, pulpit, subject_name from subject
open ccc
fetch ccc into @s, @p, @n
while @@FETCH_STATUS = 0
begin
    if @s = 'БД' update subject set subject_name = 'Самая важная дисциплина!!!'
                                where current of ccc
    fetch ccc into @s, @p, @n
end
close ccc
go

```

41. Хранимые процедуры. Создание, изменение и удаление хранимых процедур. Вызов хранимых процедур. Передача параметров

Хранимая процедура – это объект БД, представляющий собой поименованный код T-SQL, хранящийся в откомпилированном виде. Как и любой объект БД, хранимая процедура может быть создана с помощью CREATE, изменена с помощью ALTER и удалена с помощью оператора DROP.

Хранимая процедура может принимать входные и формировать выходные параметры, а результатом ее выполнения может быть целочисленное значение, возвращаемое к точке вызова с помощью оператора RETURN один или более результирующих наборов, сформированных операторами SELECT, а также содержимое стандартного выходного потока, полученного при выполнении операторов PRINT.

Вызов процедуры осуществляется с помощью оператора EXECUTE. Кроме того, результирующий набор хранимой процедуры может быть использован в качестве исходного источника строк для оператора INSERT.

```

use ФАЙЛОВАЯ_ГРУППА

go

create procedure PSUBJECT as

begin

    declare @count int = (select count(*) from SUBJECT)

```

```

        select SUBJECT, SUBJECT_NAME, PULPIT from SUBJECT

        return @count

    end

go

declare @count int

exec @count = PSUBJECT

print @count

```

42. Хранимые процедуры. DML в процедурах. Входные и выходные параметры.

Допускается применение :

- Основных DDL, DML и TCL-операторов
- Конструкций TRY/CATCH
- Курсоров
- Временных таблиц

Не допускается применение :

- CREATE or ALTER FUNCTION
- CREATE or ALTER TRIGGER
- CREATE or ALTER PROCEDURE
- CREATE or ALTER VIEW
- USE databasename

Передача параметров

```

create procedure SelectTEACHER
    @p char(10),
    @n int = 5,
    @c int output
as
    declare @rc int = 0
    begin
        begin try
            select * from (
                select ROW_NUMBER() over (order by TEACHER) rn, TEACHER, TEACHER_NAME
                from TEACHER
                where PULPIT = @p
            ) rrr where rrr.rn < @n
            set @c = (select COUNT(*) from TEACHER)
        end try
        begin catch
            set @rc = -1
        end catch
        return @rc
    end

```

Передача параметров

```
declare @code int, @ttt int
exec @code = SelectTEACHER 'ИСИТ', default, @ttt output
print 'код возврата = ' + CAST(@code as varchar(10))
print 'output = ' + CAST(@ttt as varchar(10))
```

n	TEACHER	TEACHER_NAME
1	?	Неизвестный
2	АКНВЧ	Акунович Станислав Иванович
3	БРКВЧ	Бракович Андрей Игорьевич
4	ГРМН	Герман Олег Витольдович

```
(строка обработано: 4)
код возврата = 0
output = 29
```

команду `RAISERROR` и обработку ошибок, если таковая понадобится.

```
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;
```

43. Функции, определенные пользователем. Виды функций. Обращение к функции. Передача параметров.

Функция – это объект БД, представляющий собой поименованный код T-SQL. Для создания, удаления и изменения функций необходимо применять 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. Если функция возвращает единственное значение (число, строка, дата, время и пр.), то она называется скалярной. Функция, возвращающая таблицу, называется табличной. + Multistatement.

В зависимости от структуры кода, различают встроенные функции и многооператорные табличные функции.

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

```
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);
```

```
alter function dbo.fnPROFESSION(@f varchar(20) = null, @like_q varchar(30) = null)
returns table
as return select PROFESSION, FACULTY, QUALIFICATION
from PROFESSION
where FACULTY = isnull(@f, FACULTY) and
      1 = case
            when @like_q is null          then 1
            when QUALIFICATION like @like_q then 1
            else 0
          end;
end;
```

44. Функции, определенные пользователем. Создание, изменение и удаление функций. DML в функциях.

- DROP FUCTION
- ALTER FUNCTION

Можно ли делать дмл в функциях? Можно, только не с таблицей.

```

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;

```

45. План запроса. Этапы обработки SELECT запроса. Понятие стоимости запроса. Понятия селективности и плотности. Оптимизация запросов

Обработка SQL-запроса:

Разбор - синтаксический разбор текста запроса для проверки его на соответствие правилам языка.

Разрешение имен - Проверка наличия используемых в запросе объектов БД: таблиц, представлений, столбцов, пользовательских и встроенных функций и пр.

Оптимизация :

Специальная компонента сервера – оптимизатор
Основная задача оптимизатора – построение плана запроса
План запроса представляет собой алгоритм выполнения SQL-запроса
Для каждого шага вычисляется стоимость – величина, пропорциональная продолжительности выполнения шага
Суммарная стоимость шагов плана составляет стоимость всего запроса
Задача – минимизация общей стоимости запроса.

Компиляция - откомпилированный план запроса помещается в специальную область памяти, называемую библиотечным кэшем. Кэш используется для ускорения выполнения будущих аналогичных запросов.

Выполнение - откомпилированный план выполняется сервером СУБД.

Селективность запроса – соотношение количества строк, удовлетворяющих условию, к общему количеству строк в таблице.

- Индекс успешно работает при $\leq 5\%$.
- Не нужен индекс при 80% или более

Плотность запроса – количество возвращаемых строк запроса.

Оптимизация запроса:

- анализ запроса
- выбор индекса
- выбор порядка выполнения операций соединения
- выбор метода выполнения операций соединения

46. Индексы. Назначение и применение индексов

Индекс представляет собой отдельную физическую структуру данных, которая позволяет получать быстрый доступ к одной или нескольким строкам данных.

Индексы сохраняются в страницах индексов. Для каждой индексируемой строки имеется элемент индекса, который сохраняется на странице индексов. Каждый элемент индекса состоит из ключа индекса и указателя. Индексы создаются по сбалансированному дереву B+. B+-дерево имеет древовидную структуру, в которой все листья находятся на расстоянии одинакового количества уровней от вершины дерева. Это свойство поддерживается при добавлении или удалении данных в индексируемом

столбце. Индекс всегда связан с таблицей или с подмножеством столбцов таблицы.

Свойства индексов:

- Индекс может иметь максимум 900 байтов и не более 16 столбцов.
- Разрешено максимум 249 некластеризованных индексов для таблицы.
- В UNIQUE составном индексе - однозначная комбинация значений всех столбцов каждой строки.
- Если UNIQUE не указывается, то повторяющиеся значения разрешаются.
- Параметр NONCLUSTED по умолчанию
- Обычно *кластеризованные* индексы создаются автоматически при создании таблицы если в ней присутствует первичный ключ (ограничение PRIMARY KEY)

47. Индексы. Виды индексов. Применение различных видов индексов.

Виды индексов:

- Кластеризованные и некластеризованные
- Уникальные и неуникальные
- Простые и составные
- XML-индексы
- Пространственные индексы
- Фильтрующие, покрытия (include)
- Кластеризованные

Определяет физический порядок данных в таблице. Может только один для одной таблицы. Таблица перестраивается в порядке индекса. Листья дерева индекса содержат страницы данных. Создается по умолчанию для каждой таблицы, для которой определен первичный ключ. Он уникальный – в столбце, для которого определен кластеризованный индекс, каждое значение данных может встречаться только один раз. Если кластеризованный индекс создается для столбца, содержащего повторяющиеся значения, СУБД принудительно добавляет четырехбайтовый идентификатор к строкам, содержащим дубликаты значений.

- Некластеризованные

Физически находится отдельно от таблицы. страницы листьев состоят из ключей индекса и закладок. Может быть несколько для одной таблицы. Не изменяет физическое упорядочивание строк таблицы. Если есть кластеризованный индекс, то закладка некластеризованного индекса показывает B+-дерево кластеризованного индекса таблицы. Если нет кластеризованного индекса, закладка идентична RID — Row Identifier, состоящего из: (Адреса файла, в котором хранится таблица; Адреса физического блока (страницы), в котором хранится строка; Смещения строки в странице).

Поиск данных с использованием некластеризованного индекса в зависимости от типа таблицы:

Куча — прохождение при поиске по структуре некластеризованного индекса, после чего строка извлекается, используя идентификатор строки.

Кластеризованная таблица — прохождение при поиске по структуре некластеризованного индекса, после чего следует прохождение по соответствующему кластеризованному индексу.

Применение индексов:

Индекс занимает определенный объем дискового пространства. Индекс используется для выборки данных. Для вставки и удаления данных необходимо обслуживания индекса. Чем больше индексов имеет таблица, тем больше объем работы по их реорганизации.

Правила:

- Выбирать индексы для частых запросов
- Затем оценивать их использование
- Не индексировать столбцы, по которым нет поиска

48. Индексы. Реорганизация, перестроение, включение и отключение индексов

Операции добавления и изменения строк базы данных могут повлечь образование неиспользуемых фрагментов в области памяти индекса. Процесс образования неиспользуемых фрагментов памяти называется фрагментацией.

Фрагментация индексов снижает эффект от их применения.

Для ликвидации фрагментации используются:

Для избавления от фрагментации индекса предусмотрены две специальные операции: реорганизация и перестройка индекса.

Реорганизация (REORGANIZE) выполняется быстро, но после нее фрагментация будет убрана только на самом нижнем уровне. Пусть выполнена реорганизация с помощью оператора ALTER для индекса #EX_TKEY.

```
ALTER index #EX_TKEY on #EX reorganize;
```

Тогда выполнение соответствующего запроса покажет, что уровень фрагментации значительно снизился, но не до конца.

Операция перестройки (REBUILD) затрагивает все узлы дерева, поэтому после ее выполнения степень фрагментации равна нулю. Пусть выполнена перестройка с помощью оператора ALTER для индекса #EX_TKEY в режиме OFFLINE.

```
ALTER index #EX_TKEY on #EX rebuild with (online = off);
```

Выполнением запроса о фрагментации можно оценить ее уровень.

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.

Пусть индекс пересоздан со значением параметра FILLFACTOR равным 65:

```
DROP index #EX_TKEY on #EX;  
CREATE index #EX_TKEY on #EX(TKEY)  
with (fillfactor = 65);
```

После добавления строк в таблицу #EX можно оценить уровень фрагментации:

```
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(N'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;
```

PAD_INDEX : используется для применения процента свободного пространства, указанного FillFactor, к страницам промежуточного уровня индекса во время создания индекса.

DISABLE - отключение индекса

Отключенный индекс недоступен, пока он не будет снова включен

reorganize - быстро, но фрагментация убирается только на нижнем уровне

rebuild - затрагивает все узлы дерева, поэтому фрагментация = 0

49. Триггеры. Типы триггеров. Создание и назначение after-триггера.

Триггер - специальный вид хранимых процедур, выполняющихся при событиях базы данных.

- DML-триггеры - Создаются для таблицы или представления. Реагируют на события INSERT, DELETE, UPDATE. Before, after, instead of.

```
create trigger AUD_AFTER_INSERT
on AUDITORIUM after INSERT
as
  print 'AUD_AFTER_INSERT';
  return;
go

create trigger AUD_AFTER_DELETE
on AUDITORIUM after DELETE
as
  print 'AUD_AFTER_DELETE';
  return;
go

create trigger AUD_AFTER_UPDATE
on AUDITORIUM after UPDATE
as
  print 'AUD_AFTER_UPDATE';
  return;
go
```

Две специальные виртуальные таблицы

deleted — содержит копии строк, удаленных из таблицы

inserted — содержит копии строк, вставленных в таблицу

Структура этих таблиц эквивалентна структуре таблицы, для которой определен триггер.

- DDL-триггеры - триггеры уровня сервера (ALLSERVER). Обработывают события сервера СУБД (Создание объектов сервера, Изменение объектов сервера, Удаление объектов сервера, Подключение к серверу)

- триггеры уровня базы данных (DATABASE). Обработка событий, происходящих в рамках базы данных

Триггеры AFTER можно создавать только для базовых таблиц. Можно использовать для создания журнала аудита действий в таблицах базы данных, реализации бизнес-логики, принудительного обеспечения ссылочной целостности.

AFTER-триггеры - триггеры уровня оператора. Выполняются по одному разу для каждого оператора. Выполняются после наступления события. AFTER-триггер вызывается после выполнения активизирующего его оператора. Если оператор нарушает ограничение целостности, то возникшая ошибка не допускает выполнения этого оператора и соответствующих триггеров.

Ссылочная целостность — корректность значений внешних ключей реляционной базы данных.

Триггеры типа AFTER исполняются после выполнения оператора.

50. Триггеры. Создание и назначение instead of-триггеров

Триггеры уровня оператора. Выполняются по одному разу для каждого оператора. Выполняются вместо операции - сама операция не выполняется.

Всегда использует таблицы inserted и deleted. Выполняется после создания таблиц inserted и deleted.

Выполняется перед выполнением проверки ограничений целостности или каких-либо других действий.

INSTEAD OF можно создавать для таблиц и для представлений - выполняется вместо выполнения любых действий с любой таблицей.

Не могут вызываться рекурсивно (если в триггере сработает операция, снова вызвавшая работу триггера). Если образуется рекурсия вызовов триггеров, то будет сделана попытка выполнить оператор.

```
use BSTU
go
create trigger AUDTYPE_INSTED
on AUDITORIUM_TYPE instead of INSERT, DELETE, UPDATE
as
    raiserror (N'изменение данных запрещено!!!', 10, 1);
return;
```

51. Триггеры. Использование таблиц inserted, deleted

Две специальные виртуальные таблицы:

- deleted — содержит копии строк, удаленных из таблицы
- inserted — содержит копии строк, вставленных в таблицу

Структура этих таблиц эквивалентна структуре таблицы, для которой определен триггер.

Таблица deleted – в инструкции CREATE TRIGGER указывается DELETE или UPDATE.

Таблица inserted – в инструкции CREATE TRIGGER указывается INSERT или UPDATE.

52. Транзакции. Явные и неявные транзакции

(опр1) Одна или несколько команд SQL, которые либо успешно выполняются как единое целое, либо отменяются как единое целое.

(опр2) Логическая единица работы, обеспечивающая переход базы данных из одного согласованного состояния в другое согласованное состояние.

Бывают явные и неявные.

Неявная транзакция — задает любую отдельную инструкцию INSERT, UPDATE или DELETE как единицу транзакции.

Явная транзакция — группа инструкций, начало и конец которой обозначаются инструкциями:

- BEGIN TRANSACTION
- COMMIT
- ROLLBACK

53. Транзакции. Свойства ACID

Транзакция — это механизм базы данных, позволяющий таким образом объединять несколько операторов, изменяющих базу данных, чтобы при выполнении этой совокупности операторов они или все выполнились или все не выполнились.

ACID

- Atomicity - Атомарность
- Consistency - Согласованность
- Isolation - Изолированность
- Durability - Долговечность

Проще говоря, ACID - это основные свойства транзакции.

Атомарность (операторы изменения БД, включенные в транзакцию, либо выполняются все, либо не выполняются 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).

54. Транзакции. Уровни изолированности транзакций

Уровень изолированности задает степень защищенности данных в транзакции от возможности изменения другими транзакциями.

Уровни изоляции:

- READ UNCOMMITTED

- Не изолирует операции чтения других транзакций

- Транзакция не задает и не признает блокировок

- Допускает проблемы:

- Грязное чтение

- Неповторяемое чтение

- Фантомное чтение

- READ COMMITTED

- Транзакция выполняет проверку только на наличие монопольной блокировки для данной строки

- Является уровнем изоляции по умолчанию

- Проблемы:

- Неповторяемое чтение

- Фантомное чтение

- REPEATABLE READ

- Устанавливает разделяемые блокировки на все считываемые данные и удерживает эти блокировки до тех пор, пока транзакция не будет подтверждена или отменена

- Не препятствует другим инструкциям вставлять новые строки

- Проблема:

- Фантомное чтение

- SERIALIZABLE

- Устанавливает блокировку на всю область данных, считываемых соответствующей транзакцией

- Предотвращает вставку новых строк другой транзакцией до тех пор, пока первая транзакция не будет подтверждена или отменена

- Реализуется с использованием метода блокировки диапазона ключа

- Блокировка диапазона ключа блокирует элементы индексов

- SNAPSHOT

Уровень изоляции SNAPSHOT - отдельный уровень изоляции транзакций, который нужно явно указывать в коде. Этот уровень обеспечивает согласованность данных на уровне транзакции. Это значит, что запрос обращается к версии данных, которая была зафиксирована на момент начала транзакции.

При использовании уровня изоляции SNAPSHOT операции записи не блокируют друг друга, за исключением тех случаев, когда они меняют одни и те же строки. Это приводит либо к блокировке, либо к ошибке 3960.

Уровни изоляции

- Пессимистическая модель:
 - READ UNCOMMITTED
 - REPEATABLE READ
 - SERIALIZABLE
- Оптимистическая модель:
 - SNAPSHOT
- Обе модели:
 - READ COMMITTED

55. Транзакции. Вложенные транзакции. Функция TRANCOUNT

Транзакция, выполняющаяся в рамках другой транзакции, называется вложенной.

При работе с вложенными транзакциями нужно учитывать следующее:

- 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.

56. Транзакции. Блокировки. Эскалация блокировок. Взаимные блокировки

Блокировки

- Блокировки – механизм обеспечения согласованности данных в случае одновременного обращения к данным нескольких пользователей
- Свойства:
 - Длительность блокировки
 - Режим блокировки
 - Гранулярность блокировки

Длительность блокировки — это период времени, в течение которого ресурс удерживает определенную блокировку.

Режим блокировки

- Разделяемая (shared lock)
- Монопольная (exclusive lock)
- Обновления (update lock)

- СУБД автоматически выбирает соответствующий режим блокировки, в зависимости от типа операции (чтение или запись)

Разделяемая блокировка

- Разделяемая блокировка резервирует ресурс только для чтения
- Другие процессы не могут изменять заблокированный ресурс
- Может быть несколько разделяемых блокировок

Монопольная блокировка

- Монопольная блокировка резервирует страницу или строку для монопольного использования одной транзакции
- Применяется при INSERT, UPDATE и DELETE
- Монопольную блокировку нельзя установить, если на ресурс уже установлена какая-либо блокировка

Блокировка обновления

- Можно устанавливать на объекты с разделяемой блокировкой, накладывается еще одна разделяемая блокировка
- Нельзя устанавливать при наличии на нем другой блокировки обновления или монопольной блокировки
- При COMMIT транзакции обновления, блокировка обновления преобразовывается в монопольную блокировку
- У объекта может быть только одна блокировка обновления

Эскалация блокировок — это процесс, при котором множество блокировок с маленькой гранулярностью, конвертируются в одну блокировку на более высоком уровне иерархии с большей *гранулярностью*.

Гранулярность блокировки

- Гранулярность блокировки определяет, какой объект блокируется:
 - строки
 - страницы
 - индексный ключ или диапазон индексных ключей
 - таблицы
 - экстенд
 - база данных
- СУБД выбирает гранулярность блокировки автоматически

Гранулярность блокировки

- Процесс преобразования большого числа блокировок уровня строки, страницы или индекса в одну блокировку уровня таблицы называется эскалацией блокировок (lock escalation)
- ALTER TABLE
- SET (LOCK_ESCALATION = {TABLE | AUTO | DISABLE})
- Подсказки блокировок (locking hints)
- SET LOCK_TIMEOUT - период в миллисекундах, в течение которого транзакция будет ожидать снятия блокировки с объекта (-1 по умолчанию, не установлен)

Взаимоблокировка (deadlock) — это особая проблема одновременного конкурентного доступа, в которой две транзакции блокируют друг друга

57. XML в SQL Server. Секция for XML в SELECT. Директивы PATH, AUTO, RAW. Директивы TYPE, ELEMENTS, ROOT

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.

Для преобразования результата SELECT-запроса в формат XML в операторе SELECT применяется секция FOR XML. При этом могут использоваться режимы RAW, AUTO, PATH.

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.

```
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;
```

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;

```

PATH – сочетание атрибутивной и элементной форм. 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;

```

Директивы:

TYPE - сохранять результат реляционного запроса как XML-документ или фрагмент типа данных XML.

ELEMENTS - она всё бахает как элементы. Атрибутов нет.

ROOT - Добавление к результирующему набору XML одного элемента верхнего уровня.

Представление данных в XML

- RAW – каждая строка РН в строку XML
- AUTO – каждая строка РН в XML-элемент с подчиненными
- PATH – сочетание атрибутивной и элементной форм
- EXPLICIT – расширенная форма РН

58. XML в SQL Server. Применение процедур sp_xml_preparedocument, sp_xml_removedocument. Применение OPENXML

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 ХРАТН 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 ХРАТН предназначено для выбора требуемых данных из исходного XML-документа.

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).

С помощью выражения WITH должна быть указана структура формируемого результата.

Для того, чтобы извлечь данные о товарах из XML-документа и добавить их в таблицу Товары, надо заменить оператор

```
select * from openxml(@h, '/товары/товар', 0)
with([товар] nvarchar(20), [цена] real, [количество] int )
```

на

```
insert Товары select [товар], [цена], [количество]
from openxml(@h, '/товары/товар', 0)
with([товар] nvarchar(20), [цена] real, [количество] int )
```

59. XML в SQL Server. Схема XML-документа. Коллекция XML SCHEMA

Сложность XML-данных требует иного подхода к механизму ограничения целостности для этого типа данных. В семействе XML-технологий существует и активно используется технология, основанная на языке XML-Schema.

XML-Schema – это одна из реализаций языка XML, поддерживаемая консорциумом W3C и предназначенная для описания структуры XML-документа. С помощью языка XML-Schema можно описать правила, которым должен подчиниться XML-документ. Файл, содержащий XML-Schema, обычно имеет расширение XSD (XML Schema definition). Большинство современных систем программирования предусматривают встроенные механизмы, позволяющие с помощью заданного XSD-файла проверять на корректность XML-документы.

Для хранения документов XML-Schema в БД MSS предусмотрен специальный объект – XML SCHEMA COLLECTION. Каждый такой объект может содержать один или более XML-SCHEMA-документов.

На рис. приведен пример создания объекта XML SCHEMACOLLECTION (далее коллекция схем) с именем **Student**, содержащего один документ.


```

use BSTU
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="data" 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 тег **element**).

При создании таблицы для XML-столбца можно указать имя коллекции схем. Такой столбец называется типизированным столбцом. Типизированный XML-столбец должен удовлетворять хотя бы одной схеме из связанной с ним коллекции.

```

use BSTU
go
drop table STUDENT;
go
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 (STUDENT),          -- типизированный столбец XML-типа
    FOTO         varbinary
);

```

60. XML в SQL Server. Индексирование XML. Инструкции xPath. Методы типа данных xml

Первичный XML-индекс:

- Индексируются все теги, значения и пути

- Используется для возвращения скалярных значений или поддеревьев

```
CREATE PRIMARY XML INDEX index_xml_column ON xmltab(xml_column);
```

Три типа вторичных типа XML-индексов:

- FOR PATH — по структуре
- FOR VALUE — по значениям элементов и атрибутов
- FOR PROPERTY — по свойствам

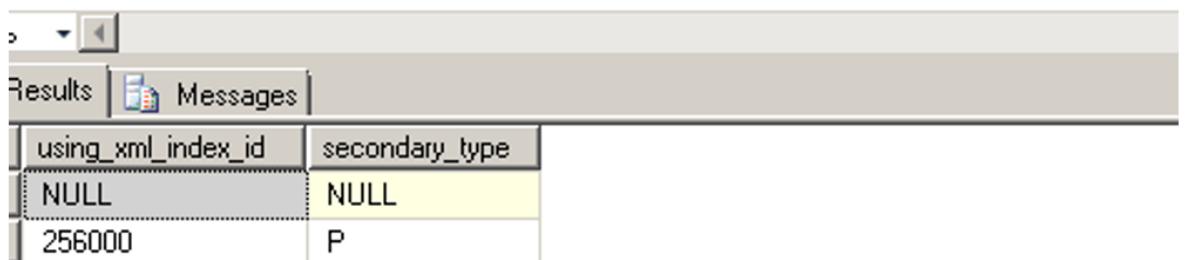
```
CREATE XML INDEX i_xmlcolumn_path ON xmltab(xml_column)
USING XML INDEX index_xml_column FOR PATH;
```

Индексы xml:

- Не могут быть составными
- Не могут быть кластеризованными

Хранятся в таблице sys.xml_indexes

```
SELECT USING_XML_INDEX_ID, SECONDARY_TYPE FROM SYS.XML_INDEXES;
```



using_xml_index_id	secondary_type
NULL	NULL
256000	P

Xpath

Язык запросов Xpath

- XML Path Language — язык запросов к элементам XML-документа

Язык запросов Xquery

- XQuery — язык запросов, разработанный для обработки данных в формате XML

Запрос данных XPath:

для доступа к элементам и атрибутам XML-документа

- Дочерние элементы узла /customer/*
- Все атрибуты узла /customer/!?*
- Чтобы вернуть только покупателей из региона Dallas /customer[@region = " Dallas "]

Оси XPath:

Имя	Описание
ancestor	Содержит родительский узел и все вышестоящие родительские узлы вплоть до корневого узла
ancestor-or-self	Содержит узлы-предки вместе с самим контекстным узлом, вплоть до корневого узла
attribute	Содержит атрибуты контекстного узла, если контекстный узел — узел элемента
child	Содержит дочерние узлы
descendant	Содержит дочерние и дальнейшие узлы
descendant-or-self	Содержит сам контекстный узел и все его дочерние и дальнейшие узлы
following	Содержит все узлы того же документа, к которому принадлежит контекстный узел, которые находятся после текущего контекстного узла в порядке документа, но не включает никаких потомков, пространства имен или узлов атрибутов
following-sibling	То же, что и following, но содержит все узлы, которые имеют того же родителя, что и контекстный узел
namespace	Содержит пространство имен контекстного узла, до тех пор, пока контекстный узел является элементом
parent	Содержит родительский узел контекстного узла Корневой элемент не имеет родителя

Методы типа данных XML

Метод	Описание
query	Исполняет выражения XPATH и XQUERY и возвращает XML-фрагмент

value	Исполняет выражения XPATH и XQUERY и возвращает скалярное значение, которое может быть преобразовано в тип SQL
exist	Исполняет выражения XPATH и XQUERY и возвращает 1, если узел, заданный выражением, найден
modify	Изменение XML-содержимого
nodes	Исполняет выражения XPATH и XQUERY и возвращает XML-фрагмент

```

use BSTU
go
select IDSTUDENT,
       INFO.value(' (/студент/паспорт/@серия) [1]', 'varchar(10)') [серия паспорта],
       INFO.value(' (/студент/паспорт/@номер) [1]', 'varchar(10)') [номер паспорта],
       INFO.value(' (/студент/телефон) [1]', 'varchar(10)') [телефон],
       INFO.query('/студент/адрес') [адрес]
from STUDENT

```

IDSTUDENT	серия паспорта	номер паспорта	телефон	адрес
1001	MP	22223333	44445555	<адрес><страна>Беларусь</страна><город>Минск</го...

Обратите внимание: 1) XML-тип является объектным типом (имеет методы и свойства);

2) в первых двух случаях применения метода **value** извлекаются значения атрибутов **серия** и **номер**;

3) в третьем случае применения **value** извлекается текстовое значение элемента **телефон**;

4) во всех трех случаях применения метода **value** в конце заданного выражения стоит число в квадратных скобках, обозначающее номер выбираемого экземпляра; дело в том, что выражения, указанные в качестве параметров, позволяют выбрать более чем одно значение; указанная в квадратных скобках единица позволяет получить только первый экземпляр;

5) второй параметр метода **value** указывает на тип данных, к которому должно быть преобразовано выбранное значение;

6) метод **query**, применяемый в четвертом элементе SELECT-списка, позволяет выбрать XML-фрагмент.