

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

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).

Логическая схема базы данных демонстрирует логические ограничения, которые распространяются на хранимые данные. В ней отражаются условия целостности, представления и таблицы. Физическая база данных показывает, как хранятся данные в системе с точки зрения файлов и индексов.

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

2. 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).

3. 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.

4. Структура файлов базы данных

Хранение данных на диске: страницы, экстенты, 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.

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

Размер экстента составляет 8 последовательно расположенных страниц или 64 Кбайт.

Существует два следующих типа экстентов:

- однородные экстенты – содержат данные одной таблицы или индекса;
- смешанные экстенты – 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.

4. Создание файлов базы данных. Файловые группы. Создание таблиц в файловой группе. Дисковое хранение файлов базы данных. Страницы, экстенты.

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
```

5. 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.

6. 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 (мантицы). 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 столбце 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)).

6. 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 единственной 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)

14. 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 сессиями.

6. 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 типа 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.

6. 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 ='ЛЭиДВ'
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 = 'ЛВ-К';
update TEACHER set PULPIT = 'ИСиТ';
```

6. 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

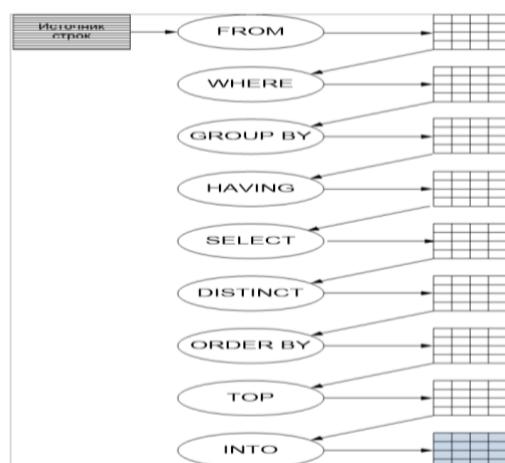
Для модификации существующих объектов БД или сервера СУБД применяется оператор **ALTER**. Структура оператора:

alter тип имя дополнение

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

drop тип имя

6. 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

7. 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 противоположном 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.

6. 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. родившихся	%
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.

6. 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;
```

8. 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)
```

6. 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]
```

6. 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)
```

6. 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.

6. 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

9. 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

10. 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

10. 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	Первая строка t1	1	Первая строка t2
2	Вторая строка t1	1	Первая строка t2
3	Третья строка t1	1	Первая строка t2
4	Четвертая строка t1	1	Первая строка t2
5	Первая строка t1	2	Вторая строка t2
6	Вторая строка t1	2	Вторая строка t2
7	Третья строка t1	2	Вторая строка t2
8	Четвертая строка t1	2	Вторая строка t2
9	Первая строка t1	3	Третья строка t2
10	Вторая строка t1	3	Третья строка t2
11	Третья строка t1	3	Третья строка t2
12	Четвертая строка t1	3	Третья строка t2

10. 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.

11. 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.

11. 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.

12. 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 'тъма'
      end 'Владимир'

Владимир
тъма

```

```

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

```

12. 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 '%'
        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';
```

11. 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».

13. 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 генерировать 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.

6. 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

```

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

```
print 'Подстрока          : '+ substring('1234567890', 3,2);
print 'Удалить пробелы слева : '+ 'X'+ 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
```

6. 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 exp   : '+ 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 'Радианы           : '+ 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));
```

15. 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;
```

15. Курсыры. Типы курсоров: global/local, static/dynamic

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

Курсыры 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....
```

Курсы могут быть статическими и динамическими.

- Статический курсор – данные выбраны один раз и произошедшие изменения не видны.

- Динамический курсор – изменения данных отображаются в динамике.

При объявлении статического курсора должен указываться атрибут STATIC. Открытие статического курсора приводит к выгрузке результирующего набора строк в динамически созданную временную таблицу системной БД TEMPDB, и все дальнейшие курсорные операции осуществляются с этой таблицей. После открытия курсора изменения в исходных таблицах, которые осуществляются в рамках этого или других сеансов, не будут отражаться в курсорном результирующем наборе.

6. Курсы. Опция scroll. Способы позиционирования в курсоре: relative/absolute, next/prior, first/last

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

По умолчанию для курсора установлен атрибут SCROLL, позволяющий применять оператор FETCH с дополнительными опциями позиционирования.

SCROLL

Курсы

Ид	Ном
1	НОМ1
2	НОМ2
3	НОМ3
4	НОМ4
5	НОМ5
6	НОМ6
7	НОМ7
8	НОМ8
9	НОМ9
10	НОМ10
11	НОМ11
12	НОМ12
13	НОМ13
14	НОМ14
15	НОМ15
16	НОМ16
17	НОМ17
18	НОМ18
19	НОМ19
20	НОМ20
21	НОМ21
22	НОМ22
23	НОМ23
24	НОМ24
25	НОМ25
26	НОМ26
27	НОМ27
28	НОМ28
29	НОМ29
30	НОМ30
31	НОМ31
32	НОМ32
33	НОМ33
34	НОМ34
35	НОМ35
36	НОМ36
37	НОМ37
38	НОМ38
39	НОМ39
40	НОМ40
41	НОМ41
42	НОМ42
43	НОМ43
44	НОМ44
45	НОМ45
46	НОМ46
47	НОМ47
48	НОМ48
49	НОМ49
50	НОМ50
51	НОМ51
52	НОМ52
53	НОМ53
54	НОМ54
55	НОМ55
56	НОМ56
57	НОМ57
58	НОМ58
59	НОМ59
60	НОМ60
61	НОМ61
62	НОМ62
63	НОМ63
64	НОМ64
65	НОМ65
66	НОМ66
67	НОМ67
68	НОМ68
69	НОМ69
70	НОМ70
71	НОМ71
72	НОМ72
73	НОМ73
74	НОМ74
75	НОМ75
76	НОМ76
77	НОМ77
78	НОМ78
79	НОМ79
80	НОМ80
81	НОМ81
82	НОМ82
83	НОМ83
84	НОМ84
85	НОМ85
86	НОМ86
87	НОМ87
88	НОМ88
89	НОМ89
90	НОМ90
91	НОМ91
92	НОМ92
93	НОМ93
94	НОМ94
95	НОМ95
96	НОМ96
97	НОМ97
98	НОМ98
99	НОМ99
100	НОМ100

Курсы – SCROLL

```
declare ccc cursor local scroll
      for select subject, pulpit from subject order by pulpit
      --read_only -- не совместен со scroll
declare @s char(10), @p char(10)
open ccc
fetch ccc into @s, @p
while @@FETCH_STATUS = 0
begin
    print @s+' '+@p
    fetch ccc into @s, @p
end
close ccc
go
```

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;

```

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).

6. Курсыры. 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

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

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. 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

```

Курсы – 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

```

16. 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

```

16. 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

(строк обработано: 4)

код возврата = 0

output = 29

```
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;
```

17. 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;
```

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

- DROP FUNCTION
- 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;

```

19. План запроса. Этапы обработки SELECT запроса. Понятие стоимости запроса. Понятия селективности и плотности. (18)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

18. 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 индексированном

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 NONCLUSTERED по умолчанию
 - Обычно *кластеризованные* индексы создаются автоматически при создании таблицы если в ней присутствует первичный ключ (ограничение PRIMARY KEY)

18. 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

19. 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

20. 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). Обрабатывают 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.

21. 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;
```

21. 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.

22. 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

22. 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 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).

22. 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

Неповторяющее чтение

Фантомное чтение

- READ COMMITTED

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

- Является уровнем изоляции по умолчанию

- Проблемы:

- Неповторяющееся 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

22. 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.

6. 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
<ul style="list-style-type: none"> • Гранулярность блокировки определяет, какой объект блокируется: <ul style="list-style-type: none"> ◦ строки ◦ страницы ◦ индексный ключ или диапазон индексных ключей ◦ таблицы ◦ экстент ◦ база данных • СУБД выбирает гранулярность блокировки автоматически 	<ul style="list-style-type: none"> • 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

6. 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 атрибутной 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 атрибутной и элементной форм
- EXPLICIT – расширенная форма РН

6. 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 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.

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 )
```

6. 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="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 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
);

```

6. 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;
```

The screenshot shows a SQL Server Management Studio window with the 'Results' tab selected. The query `SELECT USING_XML_INDEX_ID, SECONDARY_TYPE FROM SYS.XML_INDEXES;` is run, and the results are displayed in a table.

using_xml_index_id	secondary_type
NULL	N
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.

23. Операторы TCL. Привилегии. Роли. Назначение привилегий.

1. Операторы TCL:

COMMIT: Фиксирует текущую транзакцию, сохраняя все выполненные изменения в базе данных.

ROLLBACK: Откатывает текущую транзакцию, отменяя все выполненные изменения и возвращая базу данных в состояние до начала транзакции.

SAVEPOINT: Создает точку сохранения в рамках текущей транзакции, чтобы можно было выполнить откат только до этой точки.

Транзакции позволяют группировать операции в базе данных и обеспечивают атомарность, целостность и постоянство данных.

2. Привилегии: В Microsoft SQL Server привилегии определяют уровень доступа, который предоставляется пользователям или ролям для выполнения операций на базе данных или ее объектах. Привилегии включают права на выполнение операций, таких как SELECT, INSERT, UPDATE, DELETE и другие. Привилегии назначаются на уровне пользователя или роли, и могут быть установлены с использованием команды GRANT или отозваны с помощью команды REVOKE.

3. Роли: Роли представляют собой именованные группы привилегий, которые могут быть назначены пользователям. Они упрощают управление привилегиями, позволяя назначать и отзывать привилегии для целой группы пользователей, вместо назначения их для каждого пользователя отдельно. В Microsoft SQL Server есть предопределенные роли, такие как db_owner, db_datareader, db_datawriter и другие, а также возможность создания пользовательских ролей.

Роли могут быть назначены пользователям с использованием команды sp_addrolemember и отозваны с помощью команды sp_droprolemember.



Базы данных

1. Дистрибутивы СУБД Oracle. Установка СУБД Oracle 12с на Windows. Global Database Name и SID.

Дистрибутивы:

1. Oracle Database Standard Edition: это редакция Oracle Database начального уровня, предназначенная для малого и среднего бизнеса. Он включает в себя большинство функций Enterprise Edition, но ограничен одним сервером с максимумом двумя процессорами.
2. Oracle Database Enterprise Edition: это флагманская версия Oracle Database, разработанная для крупных предприятий и предприятий с высокими требованиями к рабочим нагрузкам. Он включает в себя все функции Standard Edition, а также расширенные возможности для обеспечения высокой производительности, масштабируемости и безопасности.
3. Oracle Database Express Edition: это бесплатная версия Oracle Database начального уровня, предназначенная для малого бизнеса, разработчиков и студентов. Он включает в себя подмножество функций, доступных в выпусках Standard и Enterprise, и ограничен одним ЦП и 1 ГБ ОЗУ.
4. Oracle Database Cloud Service: это облачная версия Oracle Database, предлагаемая по подписке через Oracle Cloud. Он доступен как в стандартной, так и в корпоративной версиях, и его можно легко увеличивать или уменьшать в соответствии с меняющимися требованиями к рабочей нагрузке.
5. Oracle Database Appliance: это предварительно настроенная, полностью интегрированная аппаратно-программная система, включающая Oracle Database, серверное оборудование и хранилище. Он предназначен для предприятий малого и среднего бизнеса, которым требуется простое в использовании готовое решение для работы с Oracle Database.

Установка:

1. почта на которую будут приходить сообщения о безопасности или обновлениях;

2. автоматически устанавливать обновления или нет;
3. создать и настроить бд / установить бд / обновить бд;
4. desktop / server class;
5. single instance / real application clusters / RAC one node;
6. typical / advanced;
7. создание пользователя Windows;
8. путь установки.

Системный идентификатор (SID) — это уникальное имя, идентифицирующее конкретный экземпляр Oracle.

Глобальное имя базы данных — это уникальное имя, которое идентифицирует базу данных в сети. Он состоит из имени службы базы данных и доменного имени.

Например, если имя службы базы данных — «sales», а имя домена — «example.com», то глобальное имя базы данных будет «sales.example.com».

2. Основные системные пользователи. Основные специальные привилегии. Роль DBA.

Основные пользователи:

1. SYS: предопределённый привилегированный пользователь ранга администратора базы данных, который является владельцем ключевых ресурсов БД. Этот пользователь создается автоматически при установке Oracle Database.
2. SYSTEM: предопределённый привилегированный пользователь, которому принадлежат ключевые ресурсы БД.

В Oracle Database системная привилегия SYSDBA позволяет пользователю выполнять определенные административные задачи, такие как запуск и остановка базы данных, создание и удаление баз данных, а также резервное копирование и восстановление базы данных. Другие системные привилегии:

1. SYSOPER: Эта привилегия позволяет пользователю выполнять определенные рабочие задачи, такие как запуск и остановка базы данных, но не позволяет ему выполнять такие задачи, как создание и удаление баз данных.
2. SYSDG: эта привилегия позволяет пользователю выполнять задачи, связанные с защитой данных, например создавать и поддерживать конфигурацию защиты данных.

3. SYSBACKUP: Эта привилегия позволяет пользователю выполнять задачи резервного копирования и восстановления, такие как создание резервных копий базы данных и управление ими.
4. SYSKM: Эта привилегия позволяет пользователю выполнять задачи, связанные с шифрованием базы данных, такие как создание ключей шифрования и управление ими.

DBA — предопределённая роль, которая автоматически создаётся для каждой базы данных Oracle, содержит все системные привилегии, кроме SYSDBA и SYSOPER.

3. Понятия базы данных и экземпляра базы данных.

В Oracle Database база данных представляет собой набор данных, организованных в логические структуры (например, таблицы, представления, процедуры), доступ к которым и управление которыми могут получать пользователи и приложения.

Экземпляр базы данных — это работающий экземпляр программного обеспечения Oracle Database, который обращается к данным в базе данных и управляет ими. Когда вы запускаете программное обеспечение Oracle Database, оно создает экземпляр базы данных и назначает ему уникальный системный идентификатор (SID). Экземпляр обрабатывает запросы пользователей и приложений на доступ и изменение данных в базе данных, а также управляет данными в файлах базы данных на диске.

База данных может иметь несколько экземпляров базы данных, каждый со своим собственным SID, работающих одновременно. Это может быть полезно для управления рабочей нагрузкой, аварийного переключения и других целей. Однако каждый экземпляр может одновременно обращаться только к одной базе данных.

В Oracle Database база данных и экземпляр базы данных — это два отдельных понятия. База данных — это логическая структура, в которой хранятся данные, а экземпляр базы данных — это работающее программное обеспечение, которое обращается к данным в базе данных и манипулирует ими.

4. Запуск и останов экземпляра базы данных Oracle.

Для запуска или остановки экземпляра должно использоваться подключение с разрешением SYSDBA или SYSOPER.

Запуск:

1. NOMOUNT: этот режим запускает экземпляр базы данных, но не монтирует базу данных. Это полезно для таких задач, как создание новой базы данных или изменение параметров инициализации базы данных.
2. MOUNT: этот режим запускает экземпляр базы данных и монтирует базу данных, но не открывает ее для использования. Это полезно для выполнения таких задач, как резервное копирование и восстановление, или для запуска сценариев, не требующих открытия базы данных.
3. OPEN: это режим запуска по умолчанию, при котором экземпляр базы данных запускается и монтируется, а сама база данных открывается для использования.
4. RECOVER: этот режим используется для запуска экземпляра базы данных в режиме восстановления, для восстановления базы данных из резервной копии или для применения файлов журнала повторного выполнения.
5. RESTRICTED: этот режим похож на OPEN, но позволяет входить в систему только пользователям с привилегией RESTRICTED SESSION. Это полезно для выполнения задач обслуживания, в то же время запрещая другим пользователям доступ к базе данных.

Пример: STARTUP NOMOUNT/MOUNT/OPEN/RECOVER/RESTRICTED.

(перевести из одного состояния в другое: ALTER DATABASE OPEN/...)

*Монтирование БД - это процесс, в ходе которого сервер базы данных Oracle сопоставляет файлы данных и индексов с экземпляром базы данных и готовит их к работе. При монтировании базы данных сервер базы данных проверяет целостность файлов данных и индексов, но не открывает их для доступа.)

Останов:

SHUTDOWN NORMAL Запрещено создавать новые сессии. Ожидается завершение работы всех пользователей. Самый безопасный и долгий способ останова. Никаких

- восстановительных работ при следующем старте не проводится.

SHUTDOWN TRANSACTIONAL Запрещено создавать новые сессии. Запрещено запускать новые транзакции. Сервер дожидается завершения уже начатых транзакций и отключает пользователей, не имеющих активных транзакций. Применяется в

- случаях, когда нет возможности применить NORMAL. Никаких восстановительных работ при следующем старте не проводится.

- SHUTDOWN IMMEDIATE Запрещено создавать новые сессии. Запрещено запускать новые транзакции. Все незафиксированные транзакции откатываются. Применяется в случаях, когда нет возможности ждать. Никаких восстановительных работ при следующем старте не проводится.
- SHUTDOWN ABORT Применяется в крайних случаях, когда остальные режимы останова не приводят к результату. Все действия прекращаются. Все транзакции не фиксируются и не откатываются. Пользователей отсоединяют от БД. При следующем старте будет выполнено возможное восстановление.

	ABORT	IMMEDIATE	TRANSACTIONAL	NORMAL
Новые соединения	E	-	-	-
Ждать конца сеанса	-	-	-	+
Ждать конца транзакций	-	-	+	+
Задачи по очистке	-	-	+	+
	-	+		

Пример: SHUTDOWN ABORT/IMMEDIATE/TRANSACTIONAL/NORMAL

5. Словарь базы данных: назначение, применение, основные представления.

Словарь базы данных представляет собой набор таблиц и представлений в базе данных Oracle, в которых хранятся метаданные о самой базе данных. Словарь базы данных используется ядром базы данных и другими инструментами для доступа и управления структурой и содержимым базы данных.

Цель словаря базы данных — предоставить согласованный и централизованный источник информации о базе данных, включая структуру объектов базы данных (например, таблицы, представления, индексы), привилегии и роли, предоставленные пользователям, а также производительность и так далее.

В Oracle Database представления, начинающиеся с «USER_», показывают объекты, принадлежащие текущему пользователю.

Представления, начинающиеся с «ALL_», показывают все объекты в базе данных, к которым имеет доступ текущий пользователь.

Представления, начинающиеся с «DBA_», отображают все объекты в базе данных, независимо от владельца. Эти представления могут быть доступны

только пользователям с ролью DBA или другим пользователям с достаточными привилегиями.

Представления, начинающиеся с «V\$», представляют собой динамические представления, отображающие текущее состояние и производительность экземпляра базы данных.

6. Мультиарендная архитектура Oracle Multitenant.

Oracle Multitenant - это архитектура, позволяющая размещать несколько баз данных в одной физической инстанции базы данных Oracle. Каждая база данных в этой архитектуре называется контейнером, а общие ресурсы экземпляра базы данных, такие как буферный кэш и процессы, разделяются между всеми контейнерами. Эта архитектура позволяет уменьшить накладные расходы на управление несколькими базами данных, а также упрощает обновление и обслуживание баз данных.

CDB (Container Database) — это база данных Oracle, которая содержит несколько контейнеров (PDB, Pluggable Database). CDB содержит также общую SGA и общий набор серверных процессов, которые разделяются между всеми контейнерами. CDB\$ROOT – главный контейнер, который содержит метаданные CDB. Этот контейнер копируется при создании PDB («юзается как изначальный шаблон короче» [©Какой-то чел из 6 группы](#)).

PDB (Pluggable Database) — это отдельная база данных, которая может быть создана в CDB и храниться в одном физическом файле с CDB. Каждый PDB имеет свой набор табличных пространств и набор схем; они изолированные между друг другом, не конфликтуют. Однако имеют общую SGA и один набор серверных процессов. Словарь разбивается на две части: общую часть и локальную(локальный очевидно для отдельной PDB)

Можно создавать несколько CDB. Одна CDB вмещает 252 PDB. PDB можно вложить между CDB (Елозить = перемещаться)

Если не понятно зачем CDB – Oracle дурачки и у них нельзя было создавать несколько БД в рамках одного инстанса, и чтобы не переписывать херову тучу кода решили костылем создавать БД внутри БД (мб не совсем так, но примерно)

7. Файлы экземпляра Oracle. Файл параметров, управляющие файлы, файлы паролей, файлы трассировки.

Архитектура внешней памяти состоит из ЭКЗЕМПЛЯРА (ФАЙЛ ПАРАМЕТРОВ) и БАЗЫ ДАННЫХ (ФАЙЛЫ данных,

файлы журнала повторного выполнения, управляющие файлы, временные файлы, файлы паролей).

Файл параметров (parameter file) - это текстовый файл, содержащий параметры, необходимые для конфигурации экземпляра базы данных Oracle. Эти параметры включают в себя настройки памяти, размера файла трансформации, а также пути к файлам базы данных и журналам.

Параметры:

```
select * from v$parameter;
```

Изменение spfile:

```
alter system set open_cursors = 350 scope = spfile;
```

Создать текстовый файл из бинарного:

```
create pfile 'p1.ora' from spfile;
```

Сформировать бинарный файл SPFILE параметров из текстового файла PFILE:

```
create spfile = 'sp.ora' from pfile ='p1.ora'
```

Порядок поиска:

1. spfileORACLE_SID.ora;
2. spfile.ora;
3. initORACLE_SID.ora.

Управляющие файлы (control files) — файлы, содержащие имена и местоположение основных физических файлов БД и некоторых параметров. Обычно в экземпляре базы данных Oracle используется несколько управляющих файлов, что позволяет избежать потери информации в случае отказа одного из файлов. Все управляющие файлы расположены в папке datafiles (View-connections)

Местоположение:

Platform	Default Location
UNIX and Linux	<i>ORACLE_HOME/dbs</i>
Windows	<i>ORACLE_HOME\database</i>

Изменение управляющих файлов:

- Остановить Oracle
- скопировать один из управляющих файлов
- изменить параметр CONTROL_FILES В ФАЙЛЕ ПАРАМЕТРОВ
- стартовать ORACLE

```
select * from v$controlfile;
```

```
alter database backup controlfile to trace;
```

Файлы паролей (password files) — это специальные файлы, содержащие хэши паролей для пользователей с системными привилегиями. Эти файлы используются для проверки аутентичности пользователей, которые пытаются подключиться к экземпляру базы данных с использованием системных привилегий. Если этот файл отсутствует, то вы можете выполнять администрирование своей базы данных только локально.

```
select * from v$pwfile_users;
```

Файлы трассировки (trace files) — это специальные файлы, содержащие информацию об ошибках и предупреждениях, которые были обнаружены во время работы экземпляра базы данных Oracle. Файлы трассировки могут быть использованы для диагностики и устранения проблем с экземпляром базы данных. Информация, содержащаяся в файлах трассировки, может включать в себя SQL-запросы, параметры памяти, информацию об ошибках и т. д.

```
select * from v$diag_info; -- Там найти Diag Trace или Diag Alert
```

8. Файлы базы данных Oracle. Файлы данных, журналы, архивы.

Файлы данных (data files) — это файлы, содержащие данные базы данных. В экземпляре базы данных Oracle обычно используется несколько файлов данных, что позволяет разделить данные на несколько физических файлов. У любой БД как минимум один файл данных, определённый файл данных может относиться только к одной БД.

```
select * from dba_data_files;
```

Журналы повтора (redolog files) — это файлы, в которых фиксируются изменения, вносимые пользователями в базу данных. Журналы служат для резервного копирования и восстановления базы данных, а также для реализации транзакций. Каждая база данных должна иметь не менее двух журналов повтора. Текущий файл постепенно заполняется, после его заполнения (или переключения некоторыми командами), база данных приступает к записи в следующий файл. Эта операция называется переключением журналов — она происходит циклически.

(**ДЛЯ СПРАВКИ:** Когда файлы журнала используются циклически, это означает, что они перезаписываются снова и снова по мере того, как накапливается новая информация. Обычно это означает, что есть несколько файлов журнала, и один из них используется для записи новых данных, в то время как другие файлы журнала используются для хранения старых данных. Когда один файл журнала заполняется, система начинает использовать следующий файл журнала и так далее.)

```
select * from v$logfile;
```

Мультиплексирование журналов повтора (redo logs) — поддержка нескольких копий каждого журнала.

Мультиплексирование журналов повтора (redo logs) — это процесс разделения журналов повтора на несколько физических файлов, которые могут храниться на разных дисках. Мультиплексирование журналов повтора используется для увеличения производительности и уменьшения риска потери данных в случае отказа диска. Когда один из файлов журнала повтора заполнен, процесс переключается на следующий файл журнала повтора. Таким образом, журналы повтора постоянно циклически перезаписываются, пока экземпляр базы

данных работает. В случае отказа диска, система может использовать журналы повтора для восстановления данных, так как они содержат информацию о всех изменениях, внесенных в базу данных.

<<< Два стула

```
select * from v$log;

-- Переключение группы: alter system
switch logfile;

-- Удаление файла:
alter database drop logfile member 'name';

-- Удаление группы:
alter database drop logfile group 4;

-- При удалении текущая группа должна быть не current

-- Добавление группы:
alter database add logfile group 4 'name' size 50m blocksize 512;

-- Добавление файла:
alter database add logfile member 'name' to group 4;
```

Архивы (archive logs) — это файлы, содержащие сообщения об изменениях, внесенных в базу данных с момента последнего резервного копирования. Архивы служат для резервного копирования и восстановления базы данных, а также для обеспечения целостности.

Если необходимо сохранить историю изменений, нужно, чтобы после переключения журналов сохранялась их копия. Для этого достаточно перевести работу базы данных в режим работы ARCHIVELOG.

Архивные файлы журналов повтора жизненно важны при восстановлении. Если часть базы данных потеряна или повреждена, то для устранения повреждений обычно требуется несколько архивных журналов. Файлы журналов повтора должны применяться к базе данных последовательно. Если один из архивных файлов журналов повтора пропущен, то остальные архивные файлы журналов не могут использоваться.

```
-- Посмотреть архивирование
select name, log_mode from v$database;
select instance_name, archiver from v$instance;

-- Включение shutdown
immediate startup mount
alter database archivelog
```

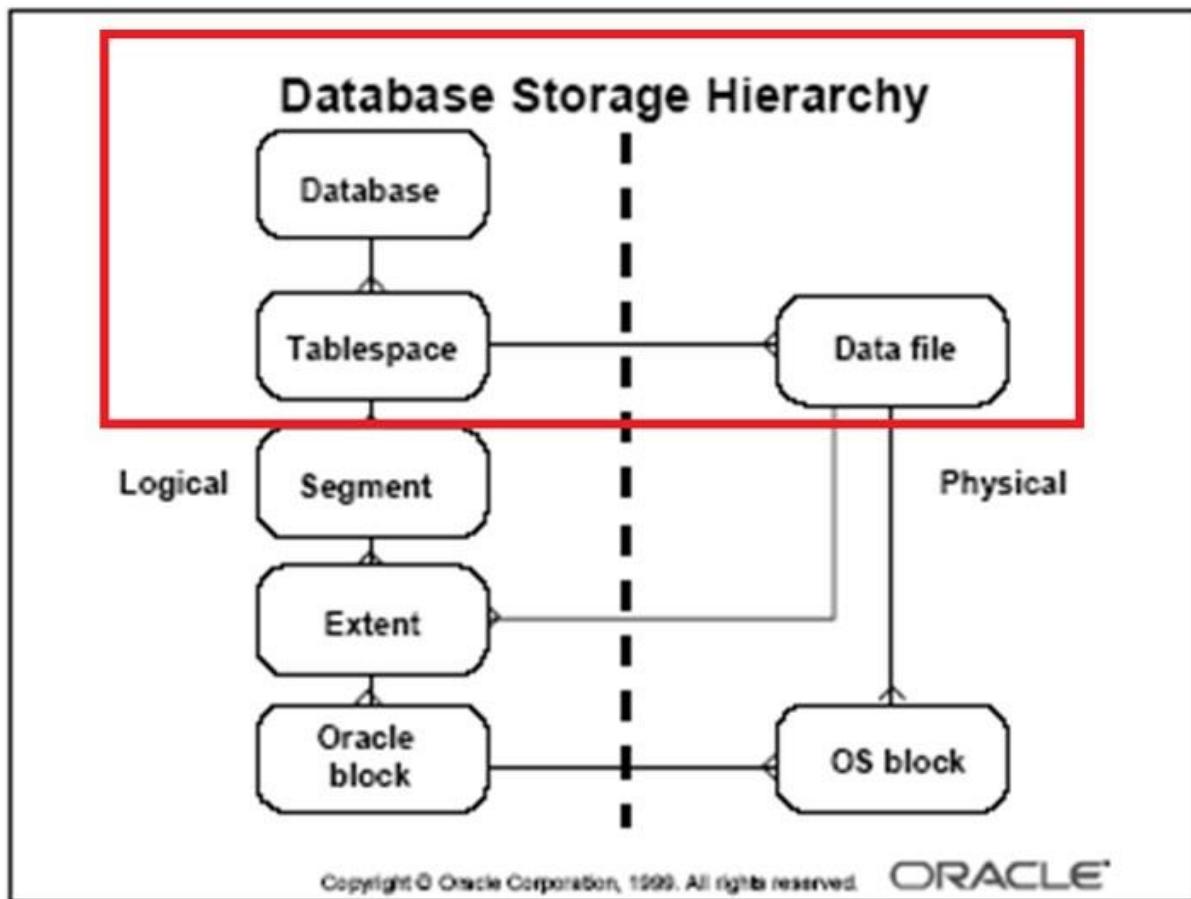
```
alter database open
```

```
-- Выключение такое же, но noarchivelog
```

Архивы создаются после переключения журналов повтора.

9. Абстрактная модель Oracle. Логическая структура внешней памяти.

В Oracle существует **двухуровневая организация базы данных**, в которой физическая структура базы данных отделена от логической структуры. Физическая структура базы данных состоит из файлов данных, журналов и других файлов, которые хранятся на диске. Логическая структура базы данных состоит из таблиц, индексов и других объектов, которые определяют, как данные будут храниться и использоваться.



Логическая структура состоит из :

Табличное пространство (tablespace) — это логическая структура, которая служит

- для хранения объектов базы данных, таких как таблицы, индексы и процедуры.

Каждое табличное пространство состоит из одного или

нескольких сегментов, которые физически хранят данные. С одним табличным пространством связаны один или несколько файлов, с каждым файлом связано только одно табличное пространство. Данные, временные данные, данные отката – организованы в виде табличных пространств.

- **Сегмент** (segment) — это логическая структура, которая содержит таблицы, индексы или другие объекты базы данных. Каждый сегмент состоит из экстентов, которые физически хранят данные. Сегменты типизируются в зависимости от типа данных, хранящихся в них – сегменты таблиц, сегменты индексов, сегменты кластеров и т.д.(всего 10 типов).
- **Экстенты** — это небольшие группы размещенных непрерывно физических блоков внешней памяти, которые используются для хранения таблиц и индексов. Экстент состоит из блоков данных. Является единицей выделения вторичной памяти (выделяется целым числом экстентов). Когда экстент заполняется выделяется следующий. Размер экстента варьируется от одного блока до 2 Гб.

Блоки данных — являются наименьшими размерными единицами хранения данных

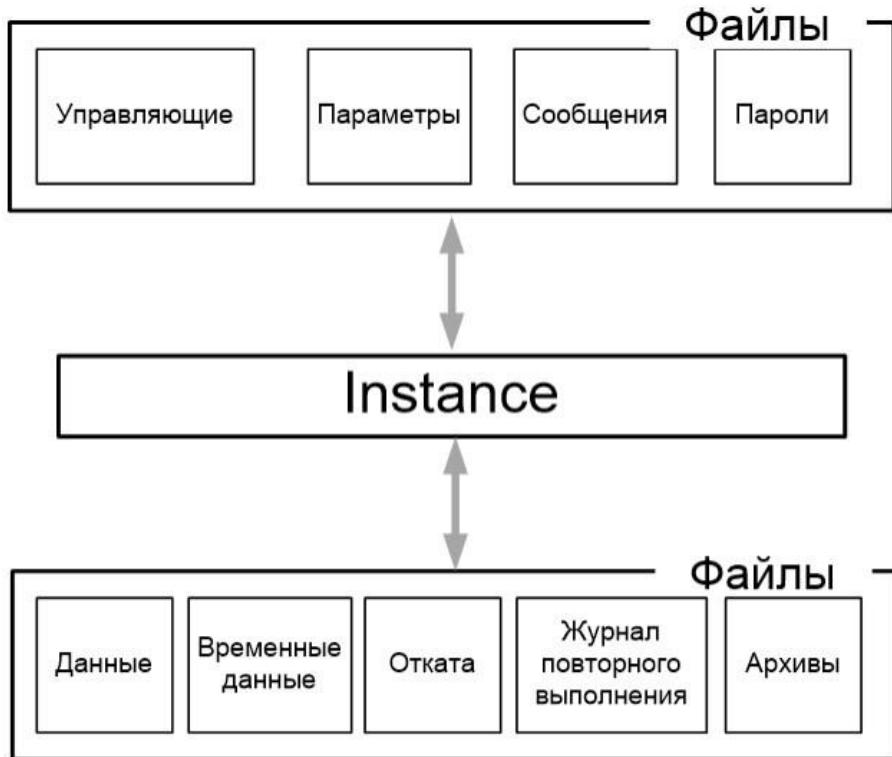
- в БД Oracle. Блоки данных содержат реальные данные и управляющую информацию, такую как указатели на следующий блок и т. д. Размер кратен 2К, и должен быть кратен величине блока операционной системы (2К, 4К, 8К, допустимы 16К, 32К). Устанавливается в файле параметров экземпляра при создании БД.

В табличном пространстве все блоки одного размера. Сегмент состоит из одного и более экстентов. Экстент состоит из идущих подряд блоков.

10. Абстрактная модель Oracle. Физическая структура внешней памяти.

Основные компоненты: файлы, образующие базу данных и поддерживающие экземпляр — файлы параметров, сообщений, данных, временных данных и журналов повторов;

Структуры памяти — системная глобальная область (System Global Area — SGA) и входящие в SGA пулы; Физические процессы или потоки — серверные процессы, фоновые процессы и подчиненные процессы.



Существуют две группы компонентов физического уровня :

1. Системные объекты — Spfile, Controlfile, файлы трассировки, архивы
2. Объекты юзера — файлы данных

11. Абстрактная модель Oracle. Структура SGA.

SGA (Глобальная область системы) — это область памяти в базе данных Oracle, которая используется для хранения данных и управляющей информации для экземпляра базы данных. Это область общей памяти, которая создается при запуске экземпляра Oracle и используется для хранения различных структур данных, включая следующие:

Буферный пул — область SGA, которая содержит образы блоков, считанные из

- файлов данные или созданные динамически, чтобы реализовать модель согласованного чтения. Может содержать грязные или чистые блоки данных.
- Грязные блоки записываются на диск в следующих случаях:

истечение тайм-аута (3 сек);

- контрольная точка;
- превышение длины грязных блоков заданного лимита;
- процесс не может обнаружить свободный блок.
-

Также он содержит следующие пулы:

- default (db_cache_size);
- keep (db_keep_cache_size);
- recycle (db_recycle_cache_size).

Просмотр размеров пулов: select * from v\$sga_dynamic_components where component like ‘%cache%’;

Помещение таблицы в определённый пул: create table xxx (k int) storage (buffer_pool [имя пула]) tablespace users;

Если после столбцов указать cache — то таблица будет помещена в конец lru списка.

- Буферы журналов повторов — предназначен для временного циклического хранения данных журналов повтора, содержимое сбрасывается на диск:
 - через каждые 3 секунды;
 - при фиксации транзакции;
 - при заполнении буфера на 1/3;
 - если в буфере более 1 мб данных.
- Фиксированная область SGA: хранит переменные, указывающие на другие области памяти, значения параметров.
- Разделяемый пул: используется для хранения кэша словаря данных, библиотечного кэша, скомпилированных запросов.
- Java pool: предназначен для работы JVM, в нем разворачиваются все необходимые объекты.
- Большой пул: это дополнительная область памяти, которая может использоваться для хранения больших объемов памяти, например, используемых для операций резервного копирования и восстановления.
- Пул потоков: Это дополнительная область памяти, которая может использоваться для хранения информации, связанной с потоками Oracle

LRU (Least Recently Used) - это алгоритм, который используется для управления кэшем в системах с ограниченным объемом памяти. Он основывается на идее, что чаще всего используемые данные будут повторно использоваться в будущем. В системах Oracle, алгоритм LRU используется для управления кэшем буфера

(Buffer Cache). Когда база данных Oracle требует данные, она сначала проверяет, есть ли они в кэше буфера. Если да, то Oracle использует данные из кэша, что увеличивает скорость доступа к данным. Если же данные отсутствуют в кэше, то Oracle их загружает с диска и помещает в кэш. При этом алгоритм LRU удаляет самые старые данные из кэша, чтобы освободить место для новых.

Размер SGA и отдельных областей памяти в нем можно настроить с помощью параметров инициализации. SGA является общим для всех серверных и фоновых процессов, и доступ к нему осуществляется ими с помощью указателей.

Файлик для конфигурации – init.ora

Память различным пулам в SGA выдел. блоками – **гранулами** (наименьшая единица выделения памяти)

12. Абстрактная модель Oracle. Серверные процессы Oracle.

Серверные процессы — процессы, выполняющиеся на основании клиентского запрос. База данных Oracle создает серверные процессы, чтобы обрабатывать запросы пользовательских процессов, соединенных с экземпляром. В некоторых ситуациях, когда приложение и БД Oracle работают на том же самом компьютере, возможно объединить пользовательский процесс и соответствующий серверный процесс в единственный процесс, чтобы уменьшить системные издержки. Однако, когда приложение и БД Oracle работают на различных компьютерах, пользовательский процесс всегда связывается с БД Oracle через отдельный серверный процесс.

При отправке sql-запроса серверный процесс:

1. производит синтаксический разбор;
2. помещает запрос в разделяемый пул;
3. создает план запроса и выполняет его;
4. при необходимости обращается в буферный пул за данными.

Для получения списка процессов:

```
select * from v$process;
```

13. Абстрактная модель Oracle. Фоновые процессы Oracle.

Фоновые процессы — специальная группа процессов для обеспечения производительности и поддержки работы большого числа пользователей.

В Oracle существует несколько фоновых процессов, включая следующие:

1. LREG (Listener Registration Process) — периодическая регистрация пользователей в процессе Listener;
2. DBWn (Database Writer) — фоновый процесс, записывающий по LRU изменённые блоки в файлы базы данных, проверяет с периодичностью 3 секунды и освобождает место в буферном кэше;
3. CKPT (Checkpoint) — выполняется при shutdown, alter system checkpoint, переключении REDO. Даёт команду DBW и LGWR на сброс буферов, а также изменяет SCN в управляющих файлах;
4. LGWR (Log Writer) — записывает блоки буфера журналов повтора в группы журналов;
5. ARCh (Archiver) — копирует файлы журнала повтора после переключения группы журналов повтора;
6. PMON (Process Monitor) — отвечает за очистку после ненормального закрытия подключений:
 - a. инициирует откат незафиксированных транзакций;
 - b. освобождает ресурсы SGA;
 - c. следит за работой других фоновых процессов, отвечает за их перезапуск;
 - d. восстанавливает работу dispatcher и shadow процессов.
7. SMON (System Monitor) — восстанавливает экземпляр для узла:
 - a. восстановление незавершенных транзакций;
 - b. очистка временных сегментов данных;
 - c. очистка временных табличных пространств;
 - d. очистка таблицы obj;
 - e. сжатие сегментов отката.
8. RECO (Recovery) — разрешение проблем, связанных с распределёнными транзакциями. Когда распределенная транзакция включает несколько баз данных, процесс RECO отвечает за координацию транзакции и обеспечение согласованности изменений, внесенных в данные, во всех базах данных. Это включает в себя отправку и получение сообщений в другие базы данных и из

них для подтверждения или отката транзакции, а также разрешение любых конфликтов, которые могут возникнуть.

Для получения списка процессов:

```
select * from v$bgprocess;
```

14. Процесс-слушатель Oracle и его основные параметры.

Oracle Net Listener – процесс на стороне сервера, прослушивающий входящие запросы клиента на соединение с экземпляром.

Listener – это программа-сервер, прослушивающая TCP-порт, принимающая запросы на соединение с Oracle экземпляром от программ-клиентов.

В результате успешной работы Listener устанавливается соединение между программой-клиентом и обработчиком запросов экземпляра.

Процесс прослушивания может быть настроен с использованием различных параметров, которые указаны в файле конфигурации listener.ora. Некоторые из основных параметров, которые можно настроить для процесса прослушивания, следующие:

- LISTENER_NAME: имя слушателя.
- PROTOCOL_ADDRESS: протокол и адрес прослушивателя. Например, TCP:1521 указывает прослушиватель, который прослушивает подключения на TCP-порту 1521.
- SERVICE_NAME: имя службы, которую обрабатывает слушатель. Это используется клиентами для подключения к экземпляру базы данных.
- SID_NAME: системный идентификатор (SID) экземпляра базы данных, который обрабатывает прослушиватель. Это используется клиентами для подключения к экземпляру базы данных.
- TIMEOUT: значение времени ожидания для слушателя в секундах. Это указывает количество времени, в течение которого прослушиватель будет ожидать ответа от экземпляра базы данных, прежде чем закрыть соединение.

15. Сетевые настройки Oracle. Установление соединения по сети.

В Oracle сетевые настройки определяют, как база данных взаимодействует с клиентами по сети. Сетевые настройки могут быть настроены с использованием

различных параметров, которые указаны в файле конфигурации sqlnet.ora.

Oracle Net Services – набор служб, которые устанавливают подключение между сервером БД и пользователями БД

Oracle Net – программный компонент, который инициализирует, устанавливает и поддерживает подключения между клиентом и сервером. Должен быть установлен и на клиенте, и на сервере.

Состоит из 2х компонентов:

- Oracle Network Foundation layer – отвечает за установку и поддержание подключений между клиентским приложением и сервером.
- Oracle Protocol Support – отвечает за отображение функциональности TNS (Transparent Network Substrate) на стандартные протоколы, используемые при подключении.

Дескриптор подключения - объединенная спецификация двух обязательных компонентов подключения к базе данных:

- 1) Имени службы базы данных
- 2) Местоположения адреса базы данных. **Идентификатор подключения** – именованный дескриптор.

Некоторые из основных сетевых параметров, которые можно настроить в Oracle, следующие:

- NAMES.DEFAULT_DOMAIN : доменное имя по умолчанию, которое используется для разрешения неквалифицированных имен хостов.
- NAMES.DIRECTORY_PATH : путь к каталогу, который используется для разрешения глобальных имен баз данных.
- TCP.VALIDNODE_CHECKING : параметр, который определяет, должен ли клиент проверять имя узла сервера перед подключением.
- SQLNET.EXPIRE_TIME: количество секунд, в течение которых соединение может бездействовать до его завершения.

Oracle поддерживает 2 режима соединений **выделенный(dedicated)** и **общий(shared)**:

1. В режиме выделенного сервера каждое клиентское подключение к базе данных обслуживается выделенным серверным процессом (также известным как "теневой процесс"). Это означает, что каждое клиентское соединение имеет свой собственный серверный процесс, который отвечает за обслуживание этого соединения и выполнение инструкций SQL от имени клиента.

Режим выделенного сервера хорошо подходит для сред с большим количеством клиентских подключений и высоким уровнем использования ЦП и памяти, поскольку он позволяет базе данных масштабироваться и эффективно использовать системные ресурсы.

2. В режиме общего сервера несколько клиентских подключений обслуживаются пулем общих серверных процессов. Когда клиент подключается к базе данных, он назначается общему серверному процессу из пула, который затем отвечает за обслуживание этого соединения и выполнение инструкций SQL от имени клиента.

Режим общего сервера хорошо подходит для сред с меньшим количеством клиентских подключений и более низким уровнем использования процессора и памяти

Также имеются различные **типы** соединений:

Basic : Базовое соединение - это прямое подключение к базе данных с использованием дескриптора подключения. Дескриптор подключения - это набор

- сведений о соединении, который определяет протокол, используемый для связи, номер порта для подключения и сетевой адрес сервера.

CONNECT имя/пароль@[/]хост[:порт][/имя_службы]

TNS . Это подключение к базе данных с использованием сетевых сервисов Oracle,

которые представляют собой набор сетевых инструментов и протоколов,

- позволяющих клиентам подключаться к базе данных по сети. Чтобы подключиться к базе данных с помощью TNS, клиентское приложение должно иметь доступ к файлу TNSnames, который содержит сведения о подключении к базе данных.

ttnames.ora — Блокнот

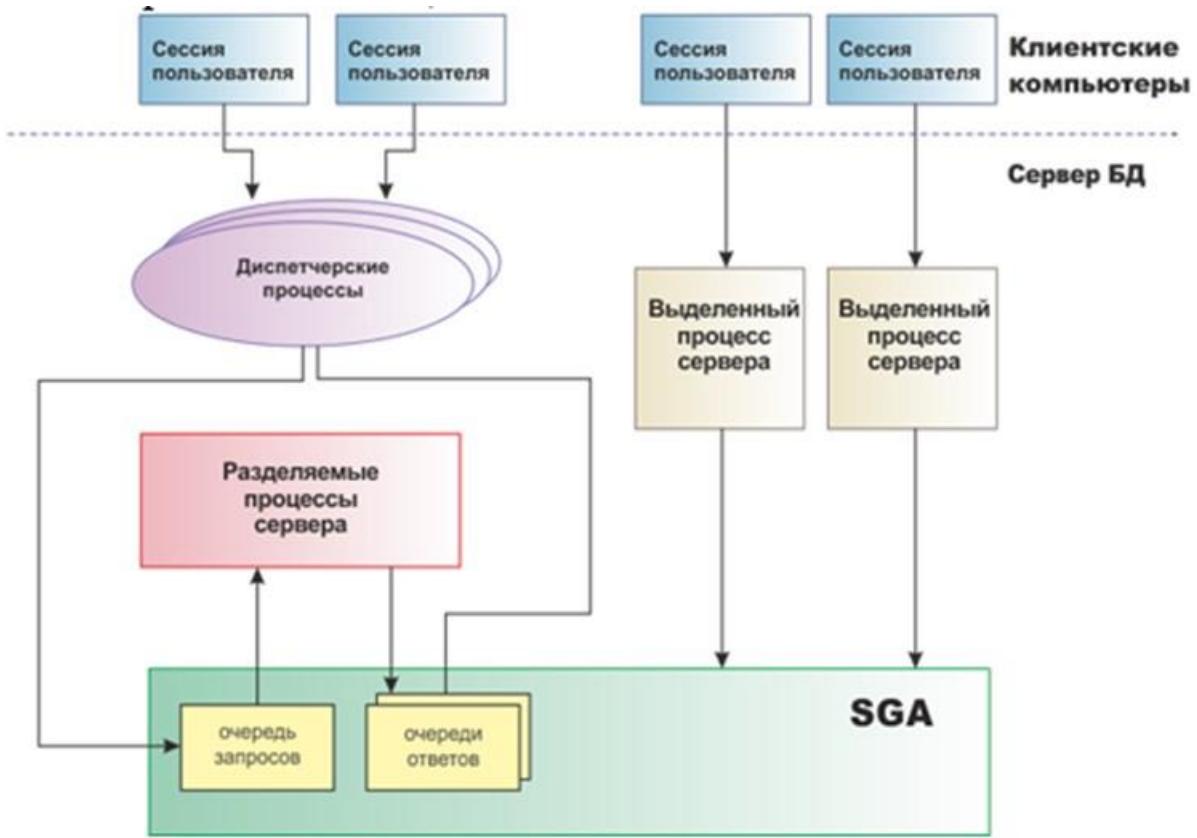
Файл Правка Формат Вид Справка

```
# ttnames.ora Network Configuration File: C:\app\client\blinova\product
# Generated by Oracle configuration tools.

PDB_B =
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST = bor.be.by)(PORT = 1521))
  )
  (CONNECT_DATA =
    (SERVICE_NAME = pdb_b.be.by)
  )
)
```

SQL> connect system/Pa\$\$w0rd@pdb_b
Connected.

- **LDAP** - это подключение к базе данных с использованием сервера LDAP (Lightweight Directory Access Protocol) в качестве посредника. Сервер LDAP -это служба каталогов, которая хранит информацию о сетевых ресурсах, включая сведения о подключении к базам данных. Чтобы подключиться к базе данных с помощью LDAP, клиентское приложение должно иметь доступ к серверу LDAP и соответствующие учетные данные.
- **Local/bequeath** - это подключение к базе данных из клиентского приложения, которое выполняется на том же компьютере, что и сервер базы данных. Чтобы подключиться к базе данных с помощью локального/наследуемого соединения, клиентскому приложению не нужно использовать прослушиватель или Oracle Net Services. Вместо этого он может напрямую подключаться к базе данных, используя протокол bequeath, который является проприетарным протоколом для обмена данными между процессами на одном компьютере.



16. Табличные пространства СУБД Oracle и их основные параметры.

В Oracle табличное пространство — это логическая единица хранения, которая используется для управления объектами данных (такими как таблицы и индексы) в базе данных. Табличные пространства могут использоваться для организации данных в базе данных и выделения места для данных на диске.

В Oracle существует несколько типов табличных пространств, включая :

- Permanent — для хранения постоянных объектов БД, может быть несколько, может быть по умолчанию;
- Temporary — для хранения временных объектов БД, может быть несколько;
- Undo — хранение сегментов отката, может быть несколько, но используется всегда одно, которое указано в файле параметров.

Главные параметры которые можно указать при создании табличных пространств :

TABLESPACE_NAME: это имя создаваемого табличного пространства.

- DATAFILE: это имя файла данных, который будет использоваться для хранения данных для табличного пространства. Вы можете указать несколько

файлов данных, разделив их запятыми. Определённый файл может использоваться только для одного табличного пространства.

- SIZE: это начальный размер файла данных в байтах, килобайтах, мегабайтах или гигабайтах. Вы можете указать необязательный параметр MAXSIZE, чтобы указать максимальный размер, до которого может увеличиться файл данных.

AUTOEXTEND ON/OFF: указывает, должен ли файл данных автоматически

- увеличивать свой размер, когда он заполняется. Если включено АВТОМАТИЧЕСКОЕ РАСШИРЕНИЕ, файл данных автоматически увеличится в размере на указанное значение ПРИРАЩЕНИЯ, когда он заполнится.

EXTENT MANAGEMENT LOCAL/DICTIONARY: это указывает, следует ли

- управлять распределением экстентов для табличного пространства локально или с использованием словаря данных. Локальное управление экстентами может повысить производительность, но для этого требуется удалить файл данных и создать его заново, если необходимо изменить размер экстента.

SEGMENT SPACE MANAGEMENT AUTO/MANUAL: Это указывает, должно ли

- распределение пространства для сегментов в табличном пространстве управляться автоматически или вручную..

DEFAULT STORAGE: задает параметры хранилища по умолчанию для

- объектов, созданных в табличном пространстве, такие как начальный и следующий размеры экстента, а также значения pctfree и pctused.

17. Роли и привилегии СУБД Oracle и их основные параметры.

В Oracle **привилегия** — это право выполнять конкретный тип предложений SQL, или право доступа к объекту другого пользователя.

Чтобы предоставить привилегию или роль пользователю или роли в Oracle, вы можете использовать инструкцию GRANT. Например:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON my_table TO my_user;
```

Чтобы отозвать привилегию или роль у пользователя или роли в Oracle, вы можете использовать инструкцию REVOKE. Например:

```
REVOKE SELECT, INSERT, UPDATE, DELETE ON my_table FROM my_user;
```

В Oracle существует два типа привилегий:

- Системные привилегии: эти привилегии позволяют пользователю или роли выполнять определенные действия, влияющие на базу данных в целом, такие как создание табличных пространств или пользователей.

Группы системных привилегий - примеры

Группа	Примеры привилегий
PROCEDURE	CREATE
PROFILE	CREATE ANY
ROLE	ALTER
ROLLBACK SEGMENT	ALTER ANY
SESSION	DROP
SEQUENCE	
SYSTEM	
TABLE	
TABLESPACE	
TRIGGER	
USER	
VIEW	

- Привилегии объекта: эти привилегии позволяют пользователю или роли выполнять определенные действия с определенным объектом базы данных, таким как таблица или представление.

Объектные привилегии - примеры

Привилегия	TABLE	VIEW	SEQUENCE	PROCEDURE
ALTER	+		+	
DELETE	+	+		
EXECUTE				+
INDEX	+			
INSERT	+	+		
REFERENCES	+			
SELECT	+	+	+	
UPDATE	+	+		

В Oracle предложение **WITH ADMIN OPTION** используется для предоставления привилегии предоставлять определенные привилегии другим пользователям или ролям.

Предложение **WITH GRANT OPTION** аналогично предложению **WITH ADMIN OPTION**, но оно применяется к привилегиям объекта, а не к системным привилегиям.

В Oracle **роль** - это именованная группа привилегий, которые могут быть предоставлены пользователям или другим ролям. Роли могут использоваться для управления привилегиями пользователей в базе данных и упрощения процесса предоставления и отзыва привилегий.

Создание ролей:

```
CREATE ROLE NAME_ROLE;
```

Посмотреть все роли:

```
SELECT * FROM DBA_ROLES;
```

Посмотреть что доступно интересующей роли:

```
SELECT * FROM DBA_SYS_PRIVS WHERE GRANTEE = 'NAME_ROLE'
```

18. Пользователь СУБД Oracle и его основные параметры.

В Oracle **пользователь** - это учетная запись базы данных, которая используется для аутентификации и авторизации пользователя или приложения для доступа к базе данных. Пользователям могут быть предоставлены привилегии и роли для управления их доступом к базе данных и действиями, которые они могут выполнять.

Некоторые из основных параметров, которые могут быть настроены для пользователей в Oracle, следующие:

USERNAME: это имя создаваемого пользователя.

- **IDENTIFIED BY:** здесь указывается пароль для пользователя. Пароль может быть указан напрямую или он может быть указан с помощью предложения EXTERNAL IDENTIFIED, которое позволяет хранить пароль во внешней службе аутентификации.

DEFAULT TABLESPACE : задает табличное пространство по умолчанию для

- пользователя. Все объекты, созданные пользователем, будут храниться в этом табличном пространстве до тех пор, пока не будет указано другое табличное пространство.

TEMPORARY TABLESPACE : указывает временное табличное пространство для

- пользователя. Временное табличное пространство используется для хранения временных объектов, таких как буферы сортировки и глобальные временные таблицы.

QUOTA: задает квоту для пользователя в указанном табличном

- пространстве. Квота ограничивает объем пространства, которое пользователь может использовать в табличном пространстве.

PROFILE : здесь указывается профиль пользователя. Профиль - это набор

- ограничений ресурсов и политик паролей, которые могут быть назначены пользователю.

- **ROLE** : здесь указываются роли, которые должны быть предоставлены пользователю. Роли - это именованные группы привилегий, которые могут быть предоставлены пользователям.
- **ACCOUNT LOCK | UNLOCK**, (заблокированный или разблокированный пользователь, может использоваться для блокировки пользователя на время или навсегда)
- **PASSWORD EXPIRE**, (пароль пользователя должен быть изменен до того, как пользователь попытается войти в базу данных, то есть при первом входе, будет предложено сменить текущий пароль, старый пароль только для первого входа).

19. Профиль безопасности СУБД Oracle и его основные параметры.

В Oracle **профиль безопасности** - это набор параметров безопасности, которые могут быть назначены пользователю для управления его доступом к базе данных и действиями, которые он может выполнять. Профили безопасности можно использовать для обеспечения соблюдения политик безопасности и упрощения процесса управления привилегиями пользователей и ограничениями ресурсов.

Некоторые из основных параметров, которые можно настроить для профилей безопасности в Oracle, следующие:

- PASSWORD_LIFE_TIME: количество дней, в течение которых пароль
 - пользователя действителен до истечения срока его действия.
- FAILED_LOGIN_ATTEMPTS: The number of failed login attempts that are allowed before the user's account is locked.
- PASSWORD_REUSE_TIME: количество дней, которые пользователь должен
 - подождать, прежде чем повторно использовать пароль.
- PASSWORD_LOCK_TIME: количество дней, в течение которых учетная запись
 - пользователя будет заблокирована после достижения максимального количества неудачных попыток входа в систему.
- PASSWORD_GRACE_TIME: количество дней, в течение которых
 - пользователь должен сменить свой пароль после истечения срока его действия.
- IDLE_TIME: максимальное количество секунд, в течение которых сеанс
 - пользователя может находиться в режиме ожидания до его завершения.

- CONNECT_TIME: максимальное количество секунд, в течение которых сеанс пользователя может быть подключен до его завершения.
- SESSIONS_PER_USER : параметр для ограничения количества одновременных сеансов, которые разрешено иметь пользователю.

Создание профиля безопасности

```
CREATE PROFILE PFEACORE LIMIT
  PASSWORD_LIFE_TIME 180 -- количество дней жизни пароля
  SESSIONS_PER_USER 3 -- количество сессий для пользователя
  FAILED_LOGIN_ATTEMPTS 7 -- количество попыток входа
  PASSWORD_LOCK_TIME 1 -- количество дней блокирования после ошибок
  PASSWORD_REUSE_TIME 10 -- через сколько дней можно повторить пароль
  PASSWORD_GRACE_TIME DEFAULT -- количество дней предупреждений о смене пароля
  CONNECT_TIME 180 -- время соединения, минут
  IDLE_TIME 30 -- количество минут простоя
```

Профиль с именем **DEFAULT**: профиль с неограниченными ресурсами. По умолчанию назначается всем пользователям и не имеет ограничений. Если пользователю не назначен профиль безопасности, либо в назначенному профиле отсутствуют некоторые параметры, то они берутся из профиля по умолчанию:

```
SELECT *
FROM DBA_PROFILES
WHERE PROFILE = 'DEFAULT';
```

PROFILE	RESOURCE_NAME	RESOURCE_TYPE	LIMIT	COMMON
1 DEFAULT	COMPOSITE_LIMIT	KERNEL	UNLIMITED	NO
2 DEFAULT	SESSIONS_PER_USER	KERNEL	UNLIMITED	NO
3 DEFAULT	CPU_PER_SESSION	KERNEL	UNLIMITED	NO
4 DEFAULT	CPU_PER_CALL	KERNEL	UNLIMITED	NO
5 DEFAULT	LOGICAL_READS_PER_SESSION	KERNEL	UNLIMITED	NO
6 DEFAULT	LOGICAL_READS_PER_CALL	KERNEL	UNLIMITED	NO
7 DEFAULT	IDLE_TIME	KERNEL	UNLIMITED	NO
8 DEFAULT	CONNECT_TIME	KERNEL	UNLIMITED	NO
9 DEFAULT	PRIVATE_SGA	KERNEL	UNLIMITED	NO
10 DEFAULT	FAILED_LOGIN_ATTEMPTS	PASSWORD	10	NO
11 DEFAULT	PASSWORD_LIFE_TIME	PASSWORD	180	NO
12 DEFAULT	PASSWORD_REUSE_TIME	PASSWORD	UNLIMITED	NO
13 DEFAULT	PASSWORD_REUSE_MAX	PASSWORD	UNLIMITED	NO

20. Язык SQL. Основные операторы и их назначение.

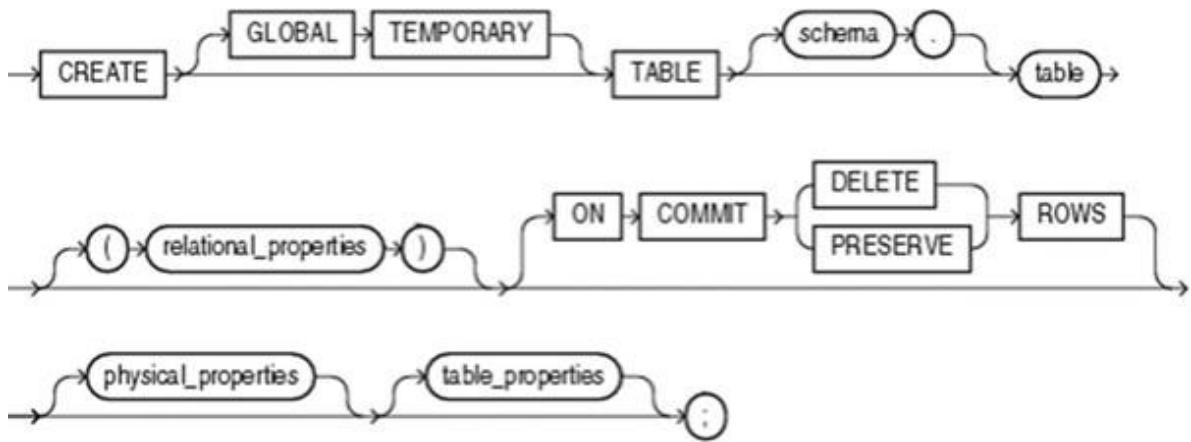
SQL (Structured Query Language) - это язык программирования, который используется для управления данными в системах управления реляционными базами данных (СУБД), таких как Oracle, MySQL и Microsoft SQL Server.

Существует 4 группы операторов sql :

1. **Data definition language (DDL)** operators: Эти операторы используются для определения структуры и организации базы данных. CREATE, ALTER , ,DROP.
2. **Data manipulation language (DML)** operators: Эти операторы используются для манипулирования и изменения данных в базе данных.SELECT, INSERT, UPDATE, DELETE.
3. **Data control language (DCL)** operators: Эти операторы используются для управления доступом к базе данных и для управления привилегиями. GRANT, REVOKE
4. **Операторы языка управления транзакциями (TCL)**: Эти операторы используются для управления транзакциями в базе данных, которые представляют собой группы инструкций SQL, которые рассматриваются как единое целое.
COMMIT,ROLLBACK,SAVEPOINT

21. Таблица и ее основные параметры.

Таблицы — основная структура сохранения информации в БД. Таблица — это логическая сущность, которая делает чтение и манипуляции данных интуитивно понятными для пользователя. Таблица состоит из столбцов и строк, причем строка соответствует одиночной записи, которая состоит из набора полей. Когда вы создаете таблицу, то присваиваете ей имя и определяете набор столбцов, относящихся к ней. Каждый столбец имеет имя и определенный тип данных. Для определенных столбцов можно специфицировать ширину или точность и масштаб (scale), а некоторым из них могут быть установлены значения по умолчанию.



Типы таблиц:

- Традиционные таблицы (heap organized table)
- Индекс-таблицы (index organized table)
- Кластеризованные индекс-таблицы (index clustered table)
- Кластеризованные хэш-таблицы (hash clustered table)
- Отсортированные кластеризованные хэш-таблицы (sorted hash clustered table)

Вложенные таблицы (nested table)

- Временные таблицы (temporary table)
- Объектные таблицы
- Внешние таблицы

Может иметь до 1000 столбцов (<254)

Может иметь неограниченное число строк

Может иметь неограниченное число индексов

Нет ограничения на число таблиц

Create table Employee(

```

        emp_id char(12),
        emp_name nvarchar2(100),
        hire_date date,
        constraint pk_emp_id primary_key (emp_id)
    )

```

organization index — определяет организацию в таблице, может повысить производительность

nomonitoring — отключает мониторинг таблицы, не собирает статистику

logging — добавление dml операций в журнал

pctthreshold 20 — после выполнения операций должно оставаться минимум 20% свободного места, предотвращает фрагментацию, повышает производительность

Параметры

Параметры PCTFREE и PCTUSED

Параметр PCTFREE – процент памяти блока, резервируемой для возможных обновлений строк, уже содержащихся в блоке

Параметр PCTUSED – процент занятой части памяти блока

Управление параметрами заполнения блока

Automatic segment space management - только PCTFREE

Manual segment space management - PCTUSED, PCTTHRESHOLD, FREELISTS и др.

44. Иерархические запросы в СУБД Oracle.

22. Временные таблицы.

Временные таблицы – механизм хранения данных в БД. Состоит из столбцов и строк, как и обычная таблица. Временные таблицы – глобальны.

Привилегии для создания временной таблицы CREATE TABLE

Можно разместить временную таблицу в заданном табличном пространстве.

Временные таблицы – это шаблон, хранящийся в словаре базы данных, для нее выделяется временный сегмент в (по умолчанию) TEMPORARY-табличном пространстве и для каждого пользователя свой.

Каждый пользователь видит только свои данные (свой сегмент данных).

Статичны: временные таблицы создаются (CREATE) один раз и существуют, пока их не удалят (DROP). DROP не получится, если таблица в этот момент используется другим пользователем.

•
Временные таблицы бывают:

- ON COMMIT PRESERVE ROWS – на время сеанса, данные существуют только на время сеанса, возможны все DML-операторы, TCL-операторы

ON COMMIT DELETE ROWS – на время транзакции, данные существуют только на время транзакции, возможны все DML-операторы, после выполнения COMMIT или ROLLBACK таблица становится пустой

В начале сеанса временная таблица всегда пуста!

Для временных таблиц можно создавать триггеры, указать констрайны (ограничения), создавать индексы

Не могут быть индексно-организованными, нельзя секционировать, размещать в кластере!

Данные повторного выполнения генерируются, но их количество пренебрежительно мало.

Отличие от обычных: нет flashback (в доке есть, но я не уверен в правдивости этого факта. В лекциях ничего нет. Есть инфа про приватные временные таблицы, но 18 оракла и выше, и там действительно дропается все сразу мимо корзины)

23. Ограничения целостности в таблицах.

Ограничения целостности в реляционных базах данных позволяют легко и автоматически применять важные бизнес-правила к таблицам базы данных.

datatype тип данных Предотвращает появление в столбце значений, не соответствующих типу данных

notnull запрет значений null Предотвращает появление в столбце значений null

default значение по умолчанию Устанавливает значение в столбце по умолчанию при выполнении операции INSERT

primary key первичный ключ Предотвращает появление в столбце (группе столбцов) повторяющихся значений (комбинации значений) и пустого значения (комбинации пустых значений)

foreign key внешний ключ. Устанавливает связь между таблицей со столбцом, имеющим свойство foreignkey (FK) и таблицей, имеющей столбец со свойством primarykey (PK); предотвращает несогласованные операции между PK и FK

unique уникальное значение Аналогично primarykey, но допускает пустые значения и не может быть использован для связи с foreignkey

check проверка значений Предотвращается появление в столбце значения, не удовлетворяющего логическому условию

Для ограничений целостности PRIMARYKEY, FOREIGNKEY, UNIQUE и CHECK может быть задано имя, которое при возникновении ошибки, связанной с этим ограничением, будет указано в сообщении сервера. В том случае, если это имя не задано, при создании таблицы сервер назначает ограничениям этих типов собственные имена.

пример

```
CREATE TABLE employees ( employee_id  
NUMBER(10) PRIMARY KEY, first_name  
VARCHAR2(50) NOT NULL, email  
VARCHAR2(255) UNIQUE , hire_date DATE  
DEFAULT SYSDATE, department_id  
NUMBER(10) NOT NULL,  
FOREIGN KEY (department_id) REFERENCES departments(department_id) );
```

24. Типы данных базы данных.

Тип данных связан с конкретным форматом хранения и ограничениями диапазона. В Oracle каждому значению или константе присваивается тип данных.

Типы данных ORACLE – символьные

CHAR	Символьное поле фиксированной длины до 2000 байт
NCHAR	Поле фиксированной длины для набора символов, состоящих из нескольких байт. Максимальный размер – 2000 символов или 2000 байт в зависимости от набора символов.
VARCHAR2	Символьное поле переменной длины до 4000 байт
NVARCHAR2	Поле переменной длины для набора символов, состоящих из нескольких байт. Максимальный размер – 4000 символов или 4000 байт в зависимости от набора символов.
LONG	Символьный, переменной длины, до 2GB, оставлен для совместимости
RAW(n)	Переменной длины, для бинарных данных n <= 2000 byte оставлен для совместимости
LONG RAW	Бинарные данные до 2GB
CLOB	Символьный тип большой объект до 4GB
NLOB	CLOB для многобайтных символов
BLOB	Большой двоичный объект до 4GB
BFILE	Указатель на двоичный файл операционной системы

Типы данных ORACLE – дата/время

DATE	7 байтовое поле фиксированной длины, используемое для хранения даты и времени
INTERVAL DAY TO SECOND	11 байтовое поле фиксированной длины для интервала времени: Дни, часы, минуты, секунды

INTERVAL YEAR TO MONTH	5 байтовое поле фиксированной длины для интервала времени: Годы и месяцы
TIMESTAMP WITH TIME ZONE	13 байтовое поле фиксированной длины Дата, время и настройки, связанные с часовым поясом.
TIMESTAMP WITH LOCAL TIME	7-11 байтовое поле переменной длины Дата и время, приведенные к часовому поясу базы данных

Типы данных ORACLE – числовые

NUMBER(n, s)	Числовой тип переменной длины Точность n <= 38, общее количество цифр Масштаб s = [-84,127], количество цифр после запятой
--------------	--

Неявные преобразования типов данных

VARCHAR2	CHAR	DATE
DATE		VARCHAR2
VARCHAR2	CHAR	ROWID
ROWID		VARCHAR2
VARCHAR2	CHAR	NUMBER
NUMBER		VARCHAR2

25. Индексы базы данных. Виды и особенности применения индексов.

Индекс – структура базы данных, используемая сервером для быстрого поиска строки в таблице.

Виды индексов Oracle Database

Индексы Oracle могут относиться к нескольким видам, наиболее важные из которых перечислены ниже.

- **Уникальные и неуникальные индексы.** Уникальные индексы основаны на уникальном столбце. Уникальные индексы можно создавать явно, но Oracle не рекомендует это делать. Вместо этого следует использовать уникальные ограничения. Когда накладывается ограничение уникальности на столбец таблицы, Oracle автоматически создает уникальные индексы по этим столбцам.

- **Первичные и вторичные индексы.** Первичные индексы — это уникальные индексы в таблице, которые всегда должны иметь какое-то значение и не могут быть равны null. Вторичные индексы — это прочие индексы таблицы, которые могут и не быть уникальными.
- **Составные индексы.** Составные индексы — это индексы, содержащие два или более столбца из одной и той же таблицы. Они также известны как сцепленные индексы (concatenated index). Составные индексы особенно полезны для обеспечения уникальности сочетания столбцов таблицы в тех случаях, когда нет уникального столбца, однозначно идентифицирующего строку.

Создание индекса

Индекс создается с помощью оператора *CREATE INDEX*:

```
CREATE INDEX employee_id ON employee(employee_id) TABLESPACE
emp_index_01;
```

Типы индексов:

— Табличный (B*Tree) индекс

- Битовый индекс
- Функциональный индекс
-

- ▶ Кардинальность — количество строк, возвращаемых после каждой операции плана выполнения запроса
- ▶ Значение кардинальности равно произведению селективности на количество обработанных строк

- ▶ Селективность таблицы — значение, представляющее долю строк таблицы, удовлетворяющих определенному условию выбора
- ▶ Селективность таблицы связана с условием выбора строк
- ▶ Селективность индекса — значение, представляющее отношение количества уникальных значений индексируемых столбцов к общему числу строк таблицы
- ▶ Селективность индекса показывает долю строк от общего числа строк в таблице, которое приходится на одно значение индекса

1. Табличный индекс (**B*Tree**) структурирован в виде сбалансированного дерева

Листовой блок содержит индексированные значения столбца и соответствующий ему идентификатор строки (RowId)

Предназначен для индексирования уникальных столбцов или столбцов с высокой селективностью

2. Битовый индекс создает битовые карты для каждого возможного значения столбца, где каждому биту соответствует строка, а значение бита 1 (0) означает, что соответствующая строка содержит (не содержит) индексируемое значение

Предназначен для индексирования столбцов с низкой селективностью

Не подходит для таблиц с частым обновлением

Хорошо подходят для хранилищ данных

3. Функциональный индекс – предварительно вычисляют значения функции по заданному столбцу и сохраняют результат в индексе

LOWER(NAME) / UPPER (NAME)

Руководство по созданию индексов:

Индексация имеет смысл, если нужно обеспечить доступ одновременно не более чем к 4–5% данных таблицы. Альтернативой использованию индекса

-

для доступа к данным строки является полное последовательное чтение таблицы от начала до конца, что называется полным сканированием таблицы. Полное сканирование таблицы больше подходит для запросов, которые требуют извлечения большего процента данных таблицы. Помните, что применение индексов для извлечения строк требует двух операций чтения: индекса и затем таблицы.

Избегайте создания индексов для сравнительно небольших таблиц. Для таких

- таблиц больше подходит полное сканирование. В случае маленьких таблиц нет необходимости в хранении данных и таблиц, и индексов.

Создавайте первичные ключи для всех таблиц. При назначении столбца в

- качестве первичного ключа Oracle автоматически создает индекс по этому столбцу.

Индексируйте столбцы, участвующие в многотабличных операциях

- соединения.

Индексируйте столбцы, которые часто используются в конструкциях *WHERE*.

- Индексируйте столбцы, участвующие в операциях *ORDER BY* и *GROUP BY* или
- других операциях, таких как *UNION* и *DISTINCT*, включающих сортировку.

Поскольку индексы уже отсортированы, объем работы по выполнению необходимой сортировки данных для упомянутых операций будет существенно сокращен.

Столбцы, состоящие из длинносимвольных строк, обычно плохие кандидаты на индексацию.

- Столбцы, которые часто обновляются, в идеале не должны быть индексированы из-за связанных с этим накладных расходов.
- Индексируйте таблицы только с высокой селективностью. То есть индексируйте таблицы, в которых мало строк имеют одинаковые значения.
- Сохраняйте количество индексов небольшим.

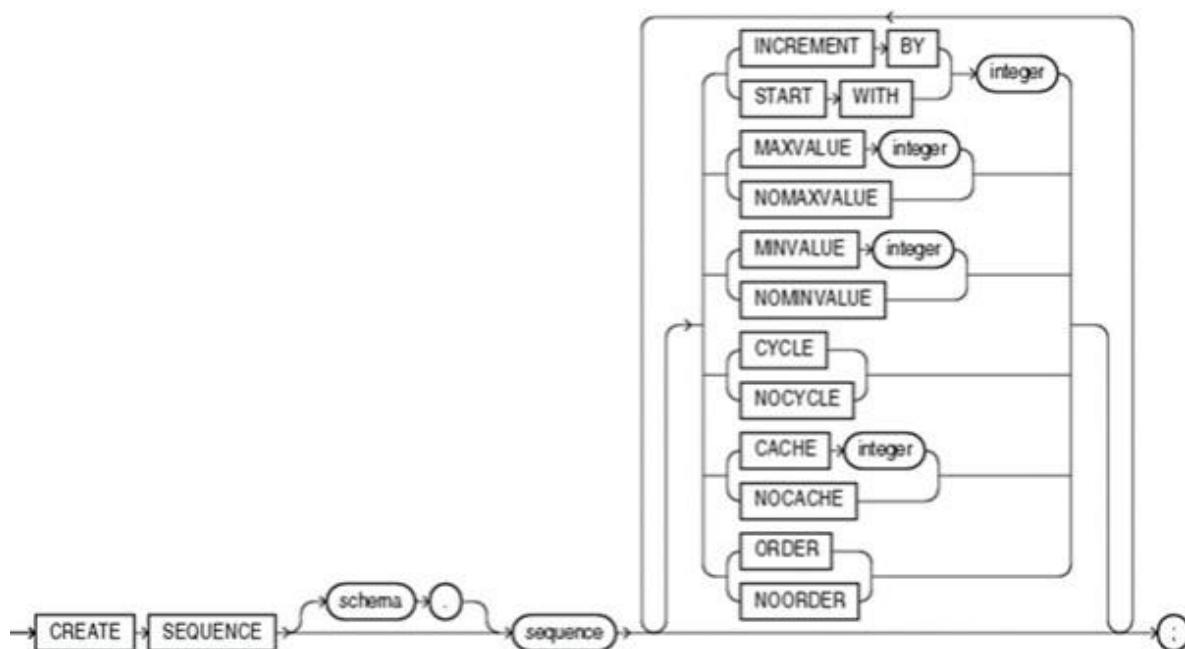
Составные индексы могут понадобиться там, где одностолбцовые значения сами по

- себе не уникальны. В составных индексах первым столбцом ключа должен быть
- столбец с максимальной селективностью.

26. Последовательность СУБД Oracle и ее параметры.

Последовательность – объект базы данных, предназначенный для генерации числовой последовательности. Oracle использует генератор последовательностей для автоматической генерации уникальной

последовательности чисел, которые пользователь может применять в своих операциях. Обычно последовательности используются для создания уникальных чисел для столбца первичного ключа.



При создании последовательности, что означают опции cache и nocache?

Что касается последовательности, опция `cache` определяет, сколько значений последовательности будут сохранены в памяти для быстрого доступа.

Недостатком создания последовательности с `cache`, что если происходит отказ системы, все кэшированные значения последовательности, которые не были использованы, будут потеряны. Это приведет к разрывам в значениях, назначенной последовательности. Когда в система восстановится, Oracle будет кэшировать новые номера, с того места, где была прервана последовательность, игнорируя потерянные значения последовательности.

Как установить значение lastvalue в последовательность Oracle?

Например, если последнее значение используемой последовательности Oracle был 100, и вы хотите, чтобы следующее значение было 225. Вы должны выполнить следующие команды.

```
ALTER SEQUENCE seq_name  
INCREMENT BY 124;  
SELECT seq_name.nextval FROM dual;  
ALTER SEQUENCE seq_name
```

INCREMENT BY 1;

Следующее значение последовательности, для использования, теперь будет 225.

Привилегия

create sequence

increment by 10

start with 100

nomaxvalue

nominvalue

nocycle

cache 20

order --если клиенту подкл в определенном порядке, сервер следит

Представления словаря: sys.dba_sequences, sys.all_sequences,

sys.user_sequences

27. Кластер и его параметры

Кластер – объект БД, который хранит значения общих столбцов нескольких таблиц

Создание CREATE CLUSTER

Привилегия CREATE CLUSTER

Таблицы, с которыми часто работают совместно, можно физически хранить совместно.

⇒ Для этого создается кластер, который будет их содержать

⇒ Строки из отдельных таблиц сохраняются в одних и тех же блоках, поэтому объединяющие запросы выполняются быстрее

Уменьшается количество операций ввода-вывода

Производительность операций вставки, обновления и удаления может быть ниже, чем для обычных таблиц

Связанные столбцы называются кластерным ключом

Хэш-кластеры используют функции хэширования кластерного ключа строки для определения физической локализации места, где строку следует хранить

Наибольшие преимущества – в запросах, использующих операции равенства:

```
select Name from STUDENT where Id = 999;
```

Пример: create cluster CJACORE.cl1 (x number(2,5) hash case 100)...

Параметры:

1. HASHKEYS: количество хеш-ключей, которые кластер использует для хранения данных. Хэш-ключи определяют распределение данных между блоками данных в кластере.
2. SIZE: размер блоков данных в кластере в байтах. Это определяет количество данных, которые могут быть сохранены в каждом блоке данных.
3. HASH IS: функция, которую кластер использует для генерации хеш-ключей. Функция должна принимать значения столбцов в ключе кластера в качестве входных данных и возвращать хэш-ключ в качестве выходных данных.
4. PCTFREE: процент свободного места, который должен оставаться в блоках данных кластера. Это определяет объем пространства, доступного для вставки или обновления строк в кластере.
5. PCTUSED: Процент используемого пространства, при котором кластер должен освободить свободное пространство. Когда используемое пространство в блоках данных достигает этого порога, кластер уплотняет данные, чтобы освободить свободное пространство.

28. Представление и его параметры.

Представление – хранимый запрос

- Можно обращаться, как к обычной таблице
- Данные хранятся в таблице
- Добавляют уровень защиты данных
- Скрывают сложность данных
- Скрывают имена столбцов таблиц

Привилегия – CREATE VIEW

Создание – CREATE (OR REPLACE) VIEW

Параметры:

- FORCE – это то же самое, что "create or replace" для хранимых процедур. Предложение "force" указывает Oracle заменить представление о том, что оно уже существует

- NOFORCE – по умолчанию
- WITH CHECK OPTION – запрещает вставку в представления данных, которые не будут отображаться в этом представлении
- WITH READ ONLY - это параметр, который может быть указан при создании view в Oracle. Он означает, что представление является только для чтения и не может быть использовано для вставки, обновления или удаления данных.
- WITH NO DATA - это параметр, который может быть указан при создании view в Oracle. Этот параметр означает, что представление будет создано без данных, то есть, будет пустым. Это может быть полезно, когда вам нужно создать представление с определенной структурой, но без данных, которые должны быть добавлены позже.

29. Материализованное представление и его параметры.

Материализованное представление — физический объект базы данных, содержащий результат выполнения запроса

Привилегия – CREATE MATERIALIZED VIEW

Создание – CREATE MATERIALIZED VIEW

BUILD IMMEDIATE – создает представление в момент выполнения оператора

START WITH – показывает, когда выполнится в первый раз (если не был построен сразу)

NEXT – показывает, когда выполнится в следующий раз

Далее – в разницу времени между START WITH и NEXT

(start with – next = периодичность) **COMMIT !!!! Отличие:** Хранят не только запрос, но и какие-то данные, которые обновляются на некотором временном промежутке или их можно обновлять вручную

REFRESH COMPLETE — полное обновление данных из базовых таблиц

REFRESH FAST – используются журналы фиксации изменений базовых таблиц

```
CREATE MATERIALIZED VIEW LOG ON EMPLOYEE  
WITH ROWID, PRIMARY KEY  
INCLUDING NEW VALUES;
```

```
CREATE MATERIALIZED VIEW EMP_SALARY_LIST  
REFRESH FAST ON COMMIT  
ENABLE QUERY REWRITE  
AS SELECT EMP_ID, SALARY FROM EMPLOYEE;
```

```
CREATE MATERIALIZED VIEW EMP_SUMMARY  
BUILD IMMEDIATE  
REFRESH COMPLETE  
ENABLE QUERY REWRITE  
AS SELECT COUNT(EMP_ID) FROM EMPLOYEE;
```

REFRESH FORCE – попытка быстрого обновления; если быстрое обновление невозможно, то выполняется полное обновление

Обновление явное

Обновление неявное по расписанию

ENABLE QUERY REWRITE — возможность перезаписывать план запроса на получения данных из представления, а не из таблицы, повышает производительность, усложняет структуру

REFRESH:

- ON COMMIT – обновление при COMMIT
- ON DEMAND – обновление по требованию

Просмотр USER_MVIEWS, USER_MVIEW_LOGS

Удалить DROP MATERIALIZED VIEW

30. Частные и публичные синонимы СУБД Oracle.

Синоним – способ обращаться к объекту базы данных без указания обязательной полной идентификации объекта (хост – экземпляр – владелец – объект).

Частный синоним принадлежит пользователю, который его создал.

Публичный синоним используется совместно всеми пользователями базы данных.

Привилегия – CREATE (PUBLIC) SYNONYM

Создание – CREATE (PUBLIC) SYNONYM

Допустимость синонима не проверяется сервером при создании!

Представление словаря dba.synonyms

Может указывать на: таблицы, процедуры, функции, последовательности, представления, пакеты, объекты в локальной или удаленной базе данных

пример: *create synonym T1 for svvcore.teacher;*

31. Основные характеристики языка PL/SQL.

PL/SQL - Procedure Language extensions to SQL

Основной язык для программирования хранимых процедур (stored procedures);

Интегрирован с базой данных Oracle;

Производительность серверных модулей;

Приложение может быть проще в реализации при написании бизнес-логики на основе хранимых процедур;

Отсутствие накладных расходов на приведение типов;

Может выполняться независимо от пользователя;

PL/SQL-функции можно вызывать из SELECT запросов

Взаимодействие с пользователем (user interaction);

Внутренний язык (proprietary for Oracle);

Содержит элементы объектно-ориентированного программирования;

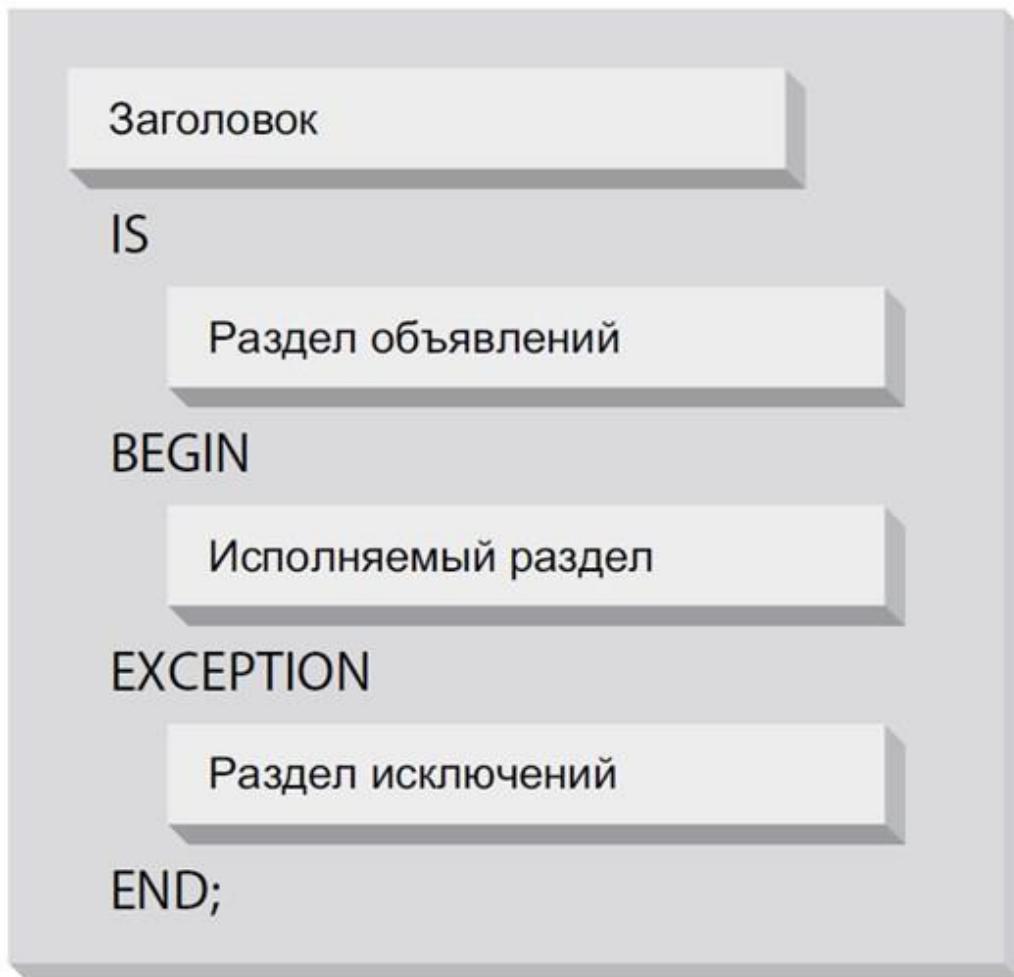
Позволяет использовать объектные типы;

Интерпретация (режим по умолчанию);

Компиляция (промежуточный код на С и конечный объектный код процессора);

Среда выполнения: SQL*Plus, SQL Developer, TOAD.

32. Структура программы языка PL/SQL. Анонимные и именованные блоки.



Нет имени (заголовка) – анонимный блок

DECLARE

Раздел объявлений – не обязательно – объявление переменных

BEGIN

Исполняемый раздел – обязательно, хоть один

EXCEPTION

Раздел исключений – не обязательно

END;

Анонимный блок – не может быть вызван из другого блока

- Используется как скрипт для выполнения PL/SQL выражений
- Начинается с DECLARE или BEGIN

- Простейший состоит из команды *null* (не делает ничего, используется когда специально хотим показать, что гасим всякую деятельность)
- не чувствителен к регистру (лучше маленьк, большими только sql-операторы)

Именованные блоки – процедуры и функции

when others – обработка любого исключения

sqlerrm – функция, которая возвращает сообщение об ошибке

sqlcode – функция, которая возвращает № ошибки (обычно 5-значный)

блоков **when** сколько угодно, самый последний – **when others**

33. Типы данных, основные операции, константы языка PL/SQL.

Символьные типы

CHAR	Символьное поле фиксированной длины до 2000 байт
NCHAR	Поле фиксированной длины для набора символов, состоящих из нескольких байт. Максимальный размер – 2000 символов или 2000 байт в зависимости от набора символов.
VARCHAR2	Символьное поле переменной длины до 4000 байт
NVARCHAR2	Поле переменной длины для набора символов, состоящих из нескольких байт. Максимальный размер – 4000 символов или 4000 байт в зависимости от набора символов.
LONG	Символьный, переменной длины, до 2GB, оставлен для совместимости
RAW(n)	Переменной длины, для бинарных данных n <= 2000 byte, оставлен для совместимости
LONG RAW	Бинарные данные до 2GB
CLOB	Символьный тип большой объект до 4GB
NLOB	CLOB для многобайтных символов
BLOB	Большой двоичный объект до 4GB
BFILE	Указатель на двоичный файл операционной системы

Дата / время

DATE	7 байтовое поле фиксированной длины, используемое для хранения даты и времени
INTERVAL DAY TO SECOND	11 байтовое поле фиксированной длины для интервала времени: Дни, часы, минуты, секунды
INTERVAL YEAR	5 байтовое поле фиксированной длины для интервала времени: Годы

TO MONTH TIMESTAMP	и месяцы
TIMESTAMP WITH TIME ZONE	13 байтовое поле фиксированной длины Дата, время и настройки, связанные с часовым поясом.
TIMESTAMP WITH LOCAL TIME	7-11 байтовое поле переменной длины Дата и время, приведенные к часовому поясу базы данных

Числовые

NUMBER(n,s)	Числовой тип переменной длины Точность n <= 38, общее количество цифр Масштаб s = [-84,127], количество цифр после запятой
-------------	--

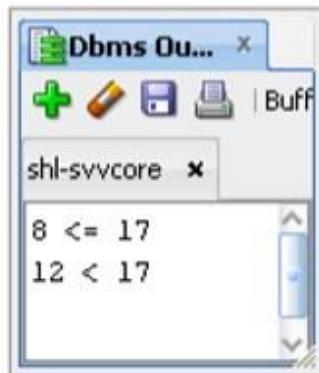
Основные операции:

- Оператор IF

```

declare
    x pls_integer := 17;
begin
    if 8 > x
    then
        dbms_output.put_line('8 > '||x);
    end if;
    -----
    if 8 > x
    then
        dbms_output.put_line('8 > '||x);
    else
        dbms_output.put_line('8 <= '||x);
    end if;
    -----
    if 8 > x
    then
        dbms_output.put_line('8 > '||x);
    elsif 8 = x
    then
        dbms_output.put_line('8 = '||x);
    elsif 12 > x
    then
        dbms_output.put_line('12 > '||x);
    elsif 12 = x
    then
        dbms_output.put_line('12 = '||x);
    else
        dbms_output.put_line('12 < '||x);
    end if;
end;

```

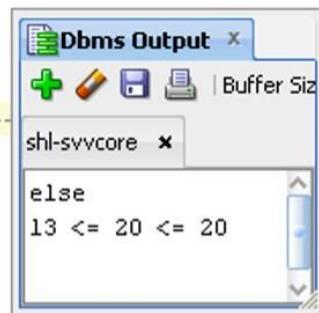


- Оператор CASE

```

declare
    x pls_integer := 17;
begin
-----
    case x
        when 1 then    dbms_output.put_line('1');
        when 2 then    dbms_output.put_line('2');
        when 3 then    dbms_output.put_line('3');
        else          dbms_output.put_line('else');
    end case;
-----
    case
        when 8 > x  then  dbms_output.put_line('8 > '||x);
        when 8 = x   then  dbms_output.put_line('8 = '||x);
        when 12 = x  then  dbms_output.put_line('12 = '||x);
        when x between 13 and 20 then dbms_output.put_line('13 <= '||x||' <= 20');
        else          dbms_output.put_line('else');
    end case;
-----
end;

```



- Циклы loop, for, while

```

declare
    x pls_integer := 0;
begin
-----
    loop
        x := x + 1;
        dbms_output.put_line(x);
        exit when x > 5;
    end loop;
-----
    for k in 1..5
    loop
        dbms_output.put_line(k);
    end loop;
-----
    while (x > 0)
    loop
        x := x - 1;
        dbms_output.put_line(x);
    end loop;
-----
end;

```

The screenshot shows the Oracle SQL Developer interface with two windows. The top window is titled 'Dbms Out...' and contains icons for file, edit, and buffer. Below it is a tab labeled 'shl-svvcore'. The main area displays the output of the PL/SQL code. It shows the sequence of numbers 1, 2, 3, 4, 5 repeated once, then 4, 5, 5, 4, 3, 2, 1, and finally 0, which corresponds to the execution of the three loops defined in the code.

Константы: с помощью constant

Пример: n1 constant number (5) := 5;

При попытке изменить значение будет выдана ошибка.

34. Поддержка национальных языков в СУБД Oracle. Наборы символов. Байтовая и символьная семантика символов.

NLS – National Language Support

- может хранить данные множества национальных языков, используя Unicode или специальные кодировки - наборы символов (character set)
- символы хранятся как коды символов, зависящие от выбранного набора символов

- в одной БД могут быть 2 набора символов: основной (database character set) и дополнительный (national character set)
- устанавливаются при создании БД
- изменяются: alter database (national) character set
- кроме символов алфавита в набор входит знаки препинания, числа, символы денежных единиц и пр.

основной набор символов для:

- хранения символьных типов char, varchar2, clob, long
- описание имен объектов, переменных
- ввод и хранения pl/sql модулей

дополнительный набор символов для:

- хранения символьных типов nchar, nvarchar2, nlob

переменная окружения **NLS_LANG = lang_territory.charset**

lang – имена месяцев, дней, направление текста : по умолч. american

territory – настройка календаря, формат даты, денег

charset – отображение символов, заглавных букв,

 NLS_LANG	REG_SZ	AMERICAN_AMERICA.WE8MSWIN1252
--	--------	-------------------------------

словари: nls_[session | instance | database]_parameters

Семантика символов:

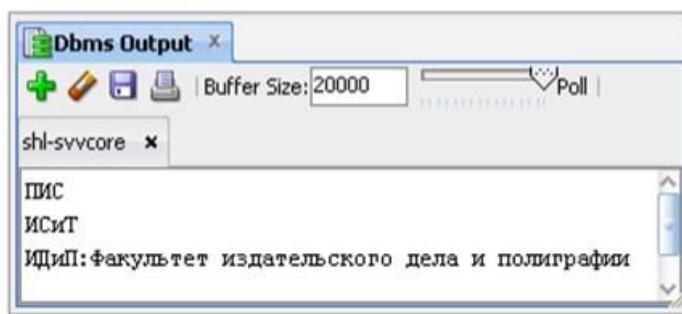
- байтоваая семантика – рассматривает строки как последовательность байтов (По умолчанию - BYTE)
- символьная семантика – рассматривает строки как последовательность символов
- задается nls_length_semantics
- Можно задавать семантику для столбца: VARCHAR2(20 BYTE) VARCHAR2(10 CHAR)

35. Связанные объявления переменных: инструкция %TYPE, инструкция %ROWTYPE.

Тип переменной основан на известной структуре данных.

Скалярная ссылка %TYPE для определения переменной на основе другой переменной или поля в таблице

Ссылка на запись %ROWTYPE для определения структуры записи на основе таблицы или курсор

```
-- 11/11.sql
declare
    subject      svvcore.subject.subject%type;
    pulpit       svvcore.pulpit.pulpit%type;
    faculty_rec svvcore.faculty%rowtype;
begin
    subject := 'ПИС';
    pulpit := 'ИСиТ';
    faculty_rec.faculty := 'ИДиП';
    faculty_rec.faculty_name := 'Факультет издательского дела и полиграфии';
    dbms_output.put_line(subject);
    dbms_output.put_line(pulpit);
    dbms_output.put_line(rtrim(faculty_rec.faculty)||':'||faculty_rec.faculty_name);
exception
    when others
        then dbms_output.put_line('error = '|| sqlerrm);
end;
/

```

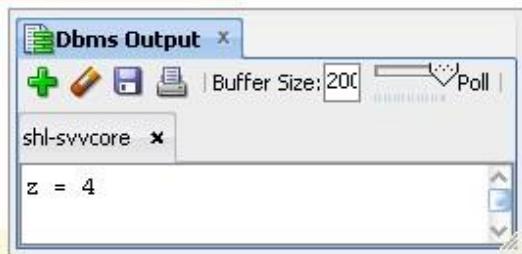
36. Локальные процедуры и функции языка PL/SQL.

Локальный программный модуль – это процедура или функция, определенная в секции декларации PL/SQL блока

- Объявление локальных процедур и функций должно размещаться в конце секции декларации после всех типов, записей, курсоров, переменных и исключений
могут быть использованы только в рамках блока, в котором они объявлены
- могут быть перегружены

Локальная процедура:

```
-- 13/08.sql
declare
    x number(3):= 4;
    y number(3):= 5;
    z number(3);
    procedure summod5 (x1 number, x2 number, x3 out number)
    is
        z number(3):= 5;
    begin
        x3 := mod(x1+ x2,z);
    end summod5;
begin
    summod5(x,y,z);
    dbms_output.put_line('z = '||z);
exception
    when others then dbms_output.put_line(sqlerrm);
end;
/
```



Локальная функция:

```
-- 13/09.sql
declare
    x number(3):= 4;
    y number(3):= 5;
    z number(3);
    s number(5);
    function summod5 (x1 number, x2 number, x3 out number)
    return number is
        z number(3):= 5;
    begin
        x3 := mod(x1+ x2,z);
        return (x1+x2);
    end summod5;
begin
    s := summod5(x,y,z);
    dbms_output.put_line('s = '||s);
    dbms_output.put_line('z = '||z);

exception
    when others then dbms_output.put_line(sqlerrm);
end;
/
```



37. Использование записей в PL/SQL. Вложенные записи.

Запись – структура данных, составленная из нескольких частей информации, называемых полями. Для объявления записи вначале надо определить как тип, а потом объявить переменную типа «запись»

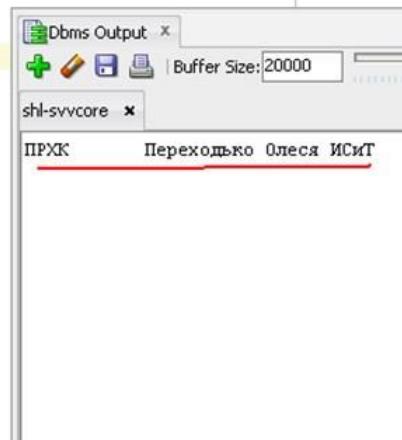
Типы записей:

*табличные

*курсорные

```
-- 13/04.sql
declare
    recl teacher%rowtype;
begin
    recl.teacher := 'ПРЖК';
    recl.teacher_name := 'Переходько Олеся';
    recl.pulpit := 'ИСиТ';
    dbms_output.put_line(recl.teacher||' '||recl.teacher_name||' '||recl.pulpit);

exception
    when others then dbms_output.put_line(sqlerrm);
end;
/
```



```
Dbms Output x
+ - Buffer Size: 20000
shl-svvcore x
ПРЖК Переходько Олеся ИСиТ
```

*программно-определенные

```
-- 13/05.sql
declare
    recl teacher%rowtype;
    type person is record
    (
        code char(10),
        name varchar2(100)
    );
    rec2 person;
begin
    select * into recl from teacher where teacher = 'УРБ';
    select teacher, teacher_name into rec2 from teacher where teacher = 'ДДК';
    dbms_output.put_line(recl.teacher||' '||recl.teacher_name||' '||recl.pulpit);
    dbms_output.put_line(rec2.code||' '||rec2.name);
exception
    when others then dbms_output.put_line(sqlerrm);
end;
/

```

УРБ	Урбанович Павел Павлович ИСИТ
ДДК	Дедко Александр Аркальевич

Объявление записей:

- на осн.таблицы (%rowtype)

```
declare
```

```
    one_book books%rowtype;
```

- на основе курсора (cursor + %rowtype)

```
declare cursor myc is
```

```
select * from books where author like '%Fadeev%';
```

```
one_curs mycc%rowtype;
```

- запись определяемая программистом:

```
-- 10/00.sql
declare
    recl teacher%rowtype;
    type person is record
        (
            code teacher.teacher%type,
            name teacher.teacher name%type
        );
    rec2 person;
begin
    select * into recl from teacher where teacher = 'УРБ';
    select teacher, teacher_name into rec2 from teacher where teacher = 'ДДК';
    dbms_output.put_line(recl.teacher||' '||recl.teacher_name||' '||recl.pulpit);
    dbms_output.put_line(rec2.code||' '||rec2.name);
exception
    when others then dbms_output.put_line(sqlerrm);
end;
```

УРБ	Урбанович Павел Павлович ИСиТ
ДДК	Дедко Александр Аркальевич

Вложенные записи – одно из полей внешней записи в действительности является полем другой записи.

```

-- 13/07.sql
declare
    recl teacher%rowtype;
    type address is record
    (
        address1 varchar2(100),
        address2 varchar2(100),
        address3 varchar2(100)
    );
    type person is record
    (
        code teacher.teacher%type,
        name teacher.teacher_name%type,
        homeaddress address
    );
    rec2 person;
begin
    rec2.code := 'ПРЖК';
    rec2.name := 'Переходько Олеся';
    rec2.homeaddress.address1 := 'Беларусь';
    rec2.homeaddress.address2 := 'Минск, Брестская обл.';
    rec2.homeaddress.address3 := 'Набережная 7, кв.77';
exception
    when others then dbms_output.put_line(sqlerrm);
end;
/

```

38. Операторы управления, операторы цикла языка PL/SQL.

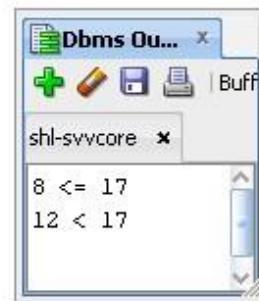
Оператор IF

- if ... then ... end if;
- if ... then ... else ... end if;
- if ... then ... elseif ... then ... (elseif ... then ...) ... else ... end if;

```

declare
    x pls_integer := 17;
begin
    if 8 > x
    then
        dbms_output.put_line('8 > '||x);
    end if;
    -----
    if 8 > x
    then
        dbms_output.put_line('8 > '||x);
    else
        dbms_output.put_line('8 <= '||x);
    end if;
    -----
    if 8 > x
    then
        dbms_output.put_line('8 > '||x);
    elsif 8 = x
    then
        dbms_output.put_line('8 = '||x);
    elsif 12 > x
    then
        dbms_output.put_line('12 > '||x);
    elsif 12 = x
    then
        dbms_output.put_line('12 = '||x);
    else
        dbms_output.put_line('12 < '||x);
    end if;
end;
/

```

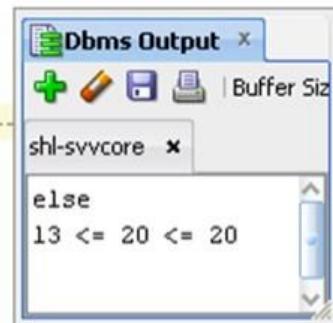


Оператор CASE

```
-- 11/19.sql
declare
    x pls_integer := 17;
begin
    case x
        when 1 then dbms_output.put_line('1');
        when 2 then dbms_output.put_line('2');
        when 3 then dbms_output.put_line('3');
        else dbms_output.put_line('else');
    end case;

    case
        when 8 > x then dbms_output.put_line('8 > ||x||');
        when 8 = x then dbms_output.put_line('8 = ||x||');
        when 12 = x then dbms_output.put_line('12 = ||x||');
        when x between 13 and 20 then dbms_output.put_line('13 <= ||x|| <= 20');
        else dbms_output.put_line('else');
    end case;

end;
```



```
case x
when ... then ...;
when ... then ...;
when x between 13 and 20 then ....;
else ....;
end case;
```

Отличие: в отличие от if выбираем значение и сравниваем его с чем-то. Во 2 – проверяем условия.

Циклы loop, for, while

```
-- 11/14.sql
declare
    x pls_integer := 0;
begin
    -----
    loop
        x := x + 1;
        dbms_output.put_line(x);
        exit when x > 5;
    end loop;

    -----
    for k in 1..5
    loop
        dbms_output.put_line(k);
    end loop;

    -----
    while (x > 0)
    loop
        x := x - 1;
        dbms_output.put_line(x);
    end loop;
    -----
end;
/
```

```
1
2
3
4
5
6
1
2
3
4
5
5
4
3
2
1
0
```

- for – нельзя измен.знач.в цикле
- loop – обяз.указ.когда вых. из цикла
- while – вып. пока вып.усл.цикла

Выход из цикла:

- exit – безусл.выход
- exit when – выход при вып.усл.
- goto – выход во внешн.контекст

39. Курсоры. Виды курсоров. Схемы обработки курсора. Курсор

– объект БД, позв работать с записями построчно

Курсор Oracle – указатель на область в PGA, в кот. хран: строки запроса, число строк, ук-ль на разобранный запрос в общем пуле

Виды:

- явный (объяв разработчиком), неявный (нет объяв, open fetch close)
- стат (выражение опр при компил), динам (выраж опр при выполнении)

- может возвр 1/0/неск строк
- для повторного созд. рез набора для др. значений параметров, курсор надо закрыть и открыть

```
-- 11/39.sql
declare
    cursor curs_subject is select subject, subject_name, pulpิต
                           from subject;
    m_subject      svvcore.subject.subject%type;
    m_subject_name svvcore.subject.subject_name%type;
    m_pulpit       svvcore.subject.pulpit%type;
begin
    open curs_subject;
    dbms_output.put_line('rowcount = '|| curs_subject%rowcount);
    loop
        fetch curs_subject into m_subject, m_subject_name, m_pulpit ;
        exit when curs_subject%notfound;
        dbms_output.put_line(''||curs_subject%rowcount||' '
                            ||m_subject||'
                            ||m_subject_name||'
                            ||m_pulpit);
    end loop;
    dbms_output.put_line('rowcount = '|| curs_subject%rowcount);
    close curs_subject;
exception
    when others
        then dbms_output.put_line(sqlerrm);
end;
/

```

rowcount = 0

1 ТП	Технология про
2 УД	Управление дан
3 ИНФ	Информатика ИС
4 КПИЯП	Конструировани
5 ПЗ	Представление
6 ОПИП	Основы педагог.
7 МСОИ	Моделирование
8 ПИС	Проектирование
9 КГиГ	Компьютерная г
10 ЗС	Энергосбережение
11 КМС	Компьютерные с
12 ОТ	Охрана труда
13 ПИ	Политология И
14 ОПП	Организация п
15 ОИ	Основы идеоло
16 КЧИСИТ	Классный час И
17 ФК	Физкультура Ф
18 ПМАПЛ	Полиграфическ

rowcount = 18

Операторы управления курсором:

- DECLARE — выполняет объявление явного курсора.
- OPEN — открывает курсор, создавая новый результирующий набор на базе указанного запроса.
- FETCH — выполняет последовательное извлечение строк из результирующего набора от начала до конца.
- CLOSE — закрывает курсор и освобождает занимаемые им ресурсы.

Ошибки неявного курсора:

- no_data_found – не возвр строк вообще
- too_many_rows – более 1 строки

select into чтобы вернуть ровно 1 строку – *точную выборку!*

40. Курсоры. Атрибуты курсора. Курсоры с параметрами.

Атрибуты:

%ISOPEN – открыт ли (у неявного всегда false)

%FOUND – true, если строки были встав/уд/выбраны

%NOTFOUND – наоборот

%ROWCOUNT - № тек строки

явные курсоры с парам:

```
cursor cur (capacity int) is
    select * from aud where cap > capacity begin
        for aum in cur(80)
            loop ...output... end loop;
```

41. Курсоры. Курсорные переменные. Параметры инстанса, связанные с курсорами.

Курсорные переменные – структуры д-х, указ. на курсорный объект

- для передачи курсора в качестве параметра
- чтобы отложить связь курсора с SELECT-запросом до выполнения OPEN

Если курсор переменной объявлен с помощью REF CURSOR без RETURN – она мб связана с любым запросом, иначе – только с запросом, возвращающим результат точно соотв. числу и ТД в записи после фразы return

```
declare
    type tlesson type is ref cursor return tlesson%rowtype;
    xcurs tlesson_type;
    rec_tlesson tlesson%rowtype;
begin
    open xcurs for select * from svvcore.tlesson;
    fetch xcurs into rec_tlesson;
```

sys_refcursor - является краткой записью создание курсорной переменной (невозможно накладывать ограничения RETURN)

Параметры Oracle, связанные с курсорами

cursor_space_for_time = {TRUE|FALSE} – больший объем памяти для курсоров и никогда не освобождается. Применяется для увеличения скорости работы курсоров при наличии памяти для разделяемого пула.

cursor_sharing = {EXACT|SIMILAR|FORCE}

open_cursors - максимальное количество открытых курсоров.

session_cached_cursors – максимальное количество кэшируемых курсоров для сессии.

42. Курсоры. Курсорные подзапросы.

```
declare
    cursor curs aut
    is select auditorium_type,
        cursor (
            select auditorium
            from auditorium aum
            where aut.auditorium_type = aum.auditorium_type
        )
    from auditorium_type aut;
    curs_aum sys_refcursor;
    aut auditorium_type.auditorium_type%type;
    txt varchar2(1000);
    aum auditorium.auditorium%type;
begin
    open curs_aut;
    fetch curs_aut into aut, curs_aum;
    while (curs_aut%found)
    loop
        txt := rtrim(aut)||':';
        loop
            fetch curs_aum into aum;
            exit when curs_aum%notfound;
            txt := txt||','||rtrim(aum);
        end loop;
        dbms_output.put_line(txt);
        fetch curs_aut into aut, curs_aum;
    end loop;
    close curs_aut;
exception
    when others
        then dbms_output.put_line(sqlerrm);
end;
```

ЛК:,236-1,313-1,324-1,408-2,103-4,105-4,107-4,110-4,111-4,132-4,02Б-4,229-4,314-4,
ЛБ-К:,206-1,301-1,413-1,423-1,304-4
ЛК-К:,114-4
ЛБ-Х:
ЛБ-СК:

43. Курсоры. Использование конструкции CURRENT OF в курсорах.

Если планируете обновить/удалить записи, на которые ссылается SELECT FOR UPDATE:

```
UPDATE имя_таблицы
SET set_clause
WHERE CURRENT OF имя_курсора;
ИЛИ
DELETE FROM имя_таблицы WHERE
CURRENT OF имя_курсора;
```

```

-- 12/50.sql
declare
    cursor curs_auditorium (capacity svvcore.auditorium.auditorium%type)
        is select auditorium, auditorium_capacity
            from auditorium
            where auditorium_capacity >= capacity for update;
    aum  svvcore.auditorium.auditorium%type;
    cty  svvcore.auditorium.auditorium_capacity%type;
begin
    open curs_auditorium(80);
    fetch curs_auditorium into aum, cty;
    while (curs_auditorium%found)
    loop
        if cty >= 90
        then
            cty := cty * 1.1;
            update auditorium
                set auditorium_capacity = cty
                where current of curs_auditorium;
        end if;
        dbms_output.put_line(''||aum||'||cty);
        fetch curs_auditorium into aum, cty;
    end loop;
    close curs_auditorium;
    rollback;
exception
    when others
        then dbms_output.put_line(sqlerrm);
end;
/

```

Dbms Output x

shl-svvcore x

236-1	80
313-1	80
408-2	80
103-4	80
105-4	80
107-4	80
110-4	80
111-4	80
114-4	110
132-4	80
02Б-4	80
229-4	80
314-4	80
320-4	80
429-4	80
?	80
301-4	99

44. Курсоры. Динамические курсоры.

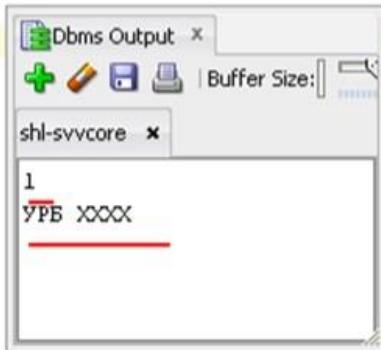
execute immediate – одностроч запросы и ddl-команды open

for, fetch, close – динам многострочные запросы

* для улучш производительности выполнения sql выражений можно использовать динамические курсоры со связанными переменными

* позволяет повторно использовать разобранные SQL выражения из разделяем. пула

```
-- 14/08.sql
declare
    stml varchar2(150) := 'update teacher set teacher_name = :1 where teacher = :2';
    t teacher.teacher%type;
    tn teacher.teacher_name%type;
    rec teacher%rowtype;
begin
    t := 'УРБ'; tn := 'XXXX';
    execute immediate stml using tn, t;
    dbms_output.put_line(sql%rowcount);
    select * into rec from teacher where teacher = 'УРБ';
    dbms_output.put_line(rtrim(rec.teacher) || ' ' || rec.teacher_name);
exception
    when others then
        dbms_output.put_line(sqlerrm);
end;
```



45. Применение псевдостолбцов ROWID, ROWNUM в PL/SQL.

ROWID – псевдостолбец, являющий уникальный идентификатор строки.

*уникальный не только в рамках таблицы, но и в рамках БД.

*упрощает работу с БД, т.к. позволяет однозначно идентифицировать любую строку таблицы, что позволяет удалять/изменять строки таблицы без первичного ключа.

*поиск по rowid – самый быстрый

!нельзя применять при разработке приложений, расчет. на работу с БД других типов.

```
-- 12/53.sql
declare
    cursor curs_auditorium (capacity svvcore.auditorium.auditorium%type)
        is select auditorium, auditorium_capacity, rowid
            from auditorium
            where auditorium_capacity >= capacity for update;
    aum svvcore.auditorium.auditorium%type;
    cty svvcore.auditorium.auditorium_capacity%type;
begin

    for xxx in curs_auditorium(80)
    loop
        case
            when xxx.auditorium_capacity >= 90
                then delete auditorium
                    where rowid = xxx.rowid;
            when xxx.auditorium_capacity >= 80
                then update auditorium
                    set auditorium_capacity = auditorium_capacity+3
                    where rowid = xxx.rowid;
        end case;
    end loop;
    for yyy in curs_auditorium(80)
    loop
        dbms_output.put_line(''||yyy.auditorium||
                            '||yyy.auditorium_capacity);
    end loop;
    rollback;
exception
    when others
        then dbms_output.put_line(sqlerrm);
end;
/

```

Dbms Output	
shl-svvcore	
236-1	83
313-1	83
408-2	83
103-4	83
105-4	83
107-4	83
110-4	83
111-4	83
132-4	83
02Б-4	83
229-4	83
314-4	83
320-4	83
429-4	83
?	83

ROWNUM – псевдостолбец, кот. умеет нумеровать строки в возвращ. рез-тах.

! нельзя напрямую использовать в запросе (вернет ошибку)

* позволяет вводить ограничение на количество выводимых записей

```
select * from students where rownum < 10;
```

46. Обработка исключений в PL/SQL, стандартные исключения, генерация и обработка исключений.

Исключительная ситуация – событие, возникающее в программе и требующее незамедлительной обработки

Два типа исключительных ситуаций:

- программно-определяемые исключения

- предопределенные (стандартные) исключения

```
-- 14/05.sql
declare
begin
null;
-- DUP_VAL_ON_INDEX           нарушена уникальность
-- TIMEOUT_ON_RESOURCE        истекло время ожидания ресурса
-- TRANSACTION_BACKED_OUT    запрещенная операция с курсором
-- INVALID_CURSOR             отсутствует подключение к Oracle
-- NOT_LOGGED_ON               данные не найдены
-- NO_DATA_FOUND                не может быть получен ROWID
-- SYS_INVALID_ROWID          не точная выборка в SELECT
-- TOO_MANY_ROWS                 деление на нуль
-- ZERO_DIVIDE                  ошибка USERENV('COMMIT SCN')
-- USERENV_COMMIT SCN_ERROR     ошибка преобразования в NUMBER
-- INVALID_NUMBER                PL/SQL недостаточно памяти
-- STORAGE_ERROR                 внутренняя ошибка PL/SQL
-- PROGRAMM_ERROR                ошибка преобразование или усечение точности
-- VALUE_ERROR                   переменная PL/SQL и курсорная переназначение не совместимы
-- ROUTETYPE_MISMATCH           попытка открыть открытый курсор
-- CURSOR_ALREADY_OPEN          попытка присвоить значение атрибуту NULL-объекта
-- ACCESS_INTO_NULL              нет подходящей фразы WHEN в операторе CASE
-- CASE_NOT_FOUND                попытка вызвать метод NULL-объекта
-- SELF_IS_NULL
-- INVALID_PATH
-- INVALID_MODE
-- INVALID_FILEHANDLE
-- INVALID_OPERATION
-- READ_ERROR
-- WRITE_ERROR
-- INTERNAL_ERROR
-- INVALID_MAXLINESIZE
-- INVALID_FILENAME
-- ACCESS_DENIED
-- INVALID_OFFSET
-- DELETE_FAILED
-- RENAME_FAILED
-- NO_DATA_NEEDED
-- collection exceptions
end; -- син. типы исключений для коллекций
/
```

Генерация исключений:

- ошибка, сгенерированная сервером
- ошибка в результате действий пользователя
- ошибка, сгенерированная приложением пользователю

sqlerrm – функция, возвр сообщ об ошибке

sqlcode – функция, возвр № ошибки

ошибка, сгенер сервером / приложением / в рез. д-вий юзера

Обработка исключений – перехват ошибки в секции исключений

47. Принцип распространения исключений в PL/SQL. Инструкция RAISE_APPLICATION_ERROR.

Распространение исключения – процесс передачи исключений от одного блока другому, если исключение не было обработано

raise_application_error – команда, перехват. выполнение тек. блока

- определена в пакете dbms_standart
- может присвоить сообщ об ошибке

При вып процедуры:

- вып блока прерыв
- измен в аргументах in out / out откат
- изменения в глобальной структуру не откатываются – надо явно rollback

48. Встроенные функции языка PL/SQL. Функции работы с датами, текстом и числами.

Для строк:

- ascii - код символа по таблице аски
- insrt - номера , где вхождение подстроки замечено
- concat - обединение строк
- unistr - символ по unicode символу

```
-- 12/55.sql
declare
  vvv varchar(200);
begin
dbms_output.put_line('----- ascii -----');
  dbms_output.put_line('ascii(''a'') = ''||ascii(''a''));
  dbms_output.put_line('ascii(''A'') = ''||ascii(''A''));
  dbms_output.put_line('ascii(''3'') = ''||ascii(''3''));
  dbms_output.put_line('ascii(''B'') = ''||ascii(''B''));
  dbms_output.put_line('ascii(''я'') = ''||ascii(''я'')");
dbms_output.put_line('----- ascistr-----');
  dbms_output.put_line('ascistr(''ABCЭMa'') = ''||ascistr(''ABCЭMa''));
dbms_output.put_line('----- chr-----');
  dbms_output.put_line('chr(97)= ''||chr(97)');
  dbms_output.put_line('chr(255)= ''||chr(255)');
dbms_output.put_line('----- unistr-----');
  dbms_output.put_line('unistr(''\042D'') = ''||unistr(''\042D''));
  dbms_output.put_line('unistr(''\044F'') = ''||unistr(''\044F''));
  dbms_output.put_line('unistr(''\0061'') = ''||unistr(''\0061''));
  dbms_output.put_line('unistr(''\0041'') = ''||unistr(''\0041''));
dbms_output.put_line('----- concat-----');
  dbms_output.put_line('concat(''12345'', ''54321'') = ''||concat(''12345'', ''54321''));
dbms_output.put_line('----- initcap-----');
  dbms_output.put_line('initcap(''abc эма Тмм'') = ''||initcap(''abc эма Тмм''));
dbms_output.put_line('----- instr-----');
  dbms_output.put_line('instr(''123456789'', ''34'') = ''||instr(''123456789'', ''34''));
  dbms_output.put_line('instr(''123456789'', ''34'', 5) = ''||instr(''123456789'', ''34'', 5));
  dbms_output.put_line('instr(''12345678912345678'', ''34'', 5) = ''||instr(''12345678912345678'', ''34'', 5));
  dbms_output.put_line('instr(''12345678912345678'', ''34'', 1,2) = ''||instr(''12345678912345678'', ''34'', 1,2));

```

```
exception
```

```

----- ascii -----
ascii('a') = 97
ascii('A') = 65
ascii('3') = 221
ascii('B') = 222
ascii('я') = 255

----- ascistr-----
ascistr('ABCЭMa') = ABC\042D\042E\044F

----- chr-----
chr(97)= а
chr(255)= я

----- unistr-----
unistr(''\042D'') = Э
unistr(''\044F'') = м
unistr(''\0061'') = я
unistr(''\0041'') = А

----- concat-----
concat(''12345'', ''54321'') = 1234554321

----- initcap-----
initcap(''abc эма Тмм'') = Аbc Эma Тmm

----- instr-----
instr(''123456789'', ''34'') = 3
instr(''123456789'', ''34'', 5) = 0
instr(''12345678912345678'', ''34'', 5) = 12
instr(''12345678912345678'', ''34'', 1,2) = 12

```

Для даты:

- trunc (d, [формат]) – возвращ.дату d, усеч.до ед.указ.в формате
- sysdate – тек.дата и время
- current_date - текущая дата
- dbtimezone -таймзона
- extract (year from date , ‘2003-08-22’) - 2003
- to_date

```

declare
  v varchar2(50);
begin
dbms_output.put_line('current_date = ''|| current_date');
dbms_output.put_line('current_timestamp = ''|| current_timestamp');
dbms_output.put_line(sysdate);
dbms_output.put_line(localtimestamp);
dbms_output.put_line(sys_extract_utc(timestamp '2000-03-28 11:30:00.00 -08:00'));
dbms_output.put_line(next_day('01-12-10', 'суббота'));
dbms_output.put_line(last_day(to_date('2003/03/15', 'YYYY/MM/DD')));
dbms_output.put_line(last_day(to_date('2003/02/03', 'YYYY/MM/DD')));
dbms_output.put_line(last_day(to_date('2004/02/03', 'YYYY/MM/DD')));
dbms_output.put_line(dbtimezone); -- CREATE/ALTER DATABASE
dbms_output.put_line(sessiontimezone); -- CREATE/ALTER SESSION
dbms_output.put_line(tz_offset('Europe/Minsk'));
dbms_output.put_line(extract(year from date '2003-08-22'));
dbms_output.put_line(extract(month from date '2003-08-22'));
dbms_output.put_line(extract(day from date '2003-08-22'));
dbms_output.put_line(months_between(sysdate, sysdate+100));
dbms_output.put_line(months_between(sysdate+100, sysdate));
dbms_output.put_line(round(to_date ('01-12-10'), 'YEAR'));
dbms_output.put_line(round(to_date ('02-12-10'), 'MONTH'));
dbms_output.put_line(round(to_date ('02-12-10'), 'DAY'));
dbms_output.put_line(round(to_date ('02-12-10'), 'Q'));
dbms_output.put_line(trunc(to_date ('01-12-10'), 'YEAR'));
dbms_output.put_line(trunc(to_date ('02-12-10'), 'Q'));
dbms_output.put_line(new_time (to_date ('2003/11/01 01:45', 'YYYY/MM/DD HH24:MI'), 'AST', 'MST'));

```

```

current_date = 01.12.10
current_timestamp = 01-ДЕК-10 11.09.59,10
01.12.10
01-ДЕК-10 11.09.59,109000000
28-МАР-00 07.30.00,000000000 PM
04.12.10
31.03.03
28.02.03
29.02.04
+00:00
Europe/Minsk
+02:00
2003
8
22
-3,32258064516129032258064516129032258065
3,32258064516129032258064516129032258065
01.01.11
01.12.10
29.11.10
01.01.11
01.01.10
01.10.10
31.10.03

```

Для чисел:

abs - модуль ,
ceil,round,floor - округление
to_number
sqrt - корень
sin,cos

```
begin
  dbms_output.put_line('abs(-10.493) ='||abs(-10.493));
  dbms_output.put_line('ceil(-10.493) ='||ceil(-10.493));
  dbms_output.put_line('round(-10.493) ='||round(-10.493));
  dbms_output.put_line('round(-10.493,1) ='||round(-10.493,1));
  dbms_output.put_line('trunc(-10.493,1) ='||trunc(-10.493,1));
  dbms_output.put_line('floor(-10.493) ='||floor(-10.493));
  dbms_output.put_line('floor(10.493) ='||floor(10.493));
  dbms_output.put_line('remainder(10,3) ='||remainder(10,3)); -- 10%3
  dbms_output.put_line('remainder(-10,3) ='||remainder(-10,3)); -- 10%3
  dbms_output.put_line('mod(-10,3) ='||mod(-10,3));
  dbms_output.put_line('bitand(0,1) ='||to_number(bitand(0,1)));
  dbms_output.put_line('bitand(15,7) ='||to_number(bitand(15,7)));
  dbms_output.put_line('bitand(5,3) ='||to_number(bitand(5,3)));
-- dbms_output.put_line('width_bucket(21, 0,100,10) ='||WIDTH_BUCKET (21,
dbms_output.put_line('cos(3.14/180*60) ='||cos(3.14/180*60));
dbms_output.put_line('acos(0.5) ='||acos(0.5)/3.14*180);
dbms_output.put_line('sin(3.14/180*60) ='||sin(3.14/180*60));
dbms_output.put_line('asin(0.5) ='||asin(0.5)/3.14*180);
dbms_output.put_line('tan(3.14/180*60) ='||tan(3.14/180*60));
dbms_output.put_line('atan(0.5) ='||atan(0.5)/3.14*180);
dbms_output.put_line('exp(1) ='||exp(1));
dbms_output.put_line('exp(0) ='||exp(0));
dbms_output.put_line('power(5,2) ='||power(5,2));
dbms_output.put_line('power(25,1/2) ='||power(25,1/2));
dbms_output.put_line('power(5,-2) ='||power(5,-2));
dbms_output.put_line('sqrt(16) ='||sqrt(16));
dbms_output.put_line('log(100,10) ='||log(100,10));
dbms_output.put_line('log(10,100) ='||log(10,100));
dbms_output.put_line('log(2,16) ='||log(2,16));
dbms_output.put_line('sign(-25.7) ='||sign(-25.7));
dbms_output.put_line('sign(25.7) ='||sign(25.7));
dbms_output.put_line('sign(0) ='||sign(0));
dbms_output.put_line('greatest(-1,3,45,9,1) ='||greatest(-1,3,45,9,1));
```

```
abs(-10.493) = 10,493
ceil(-10.493) = -10
round(-10.493) = -10
round(-10.493,1) = -10,5
trunc(-10.493,1) = -10,4
floor(-10.493) = -11
floor(10.493) = 10
remainder(10,3) = 1
remainder(-10,3) = -1
mod(-10,3) = -1
bitand(0,1) = 0
bitand(15,7) = 7
bitand(5,3) = 1
cos(3.14/180*60) = ,5004596890082057
acos(0.5) = 60,030432871142545957884
sin(3.14/180*60) = ,8657598394923444
asin(0.5) = 30,015216435571272978942
tan(3.14/180*60) = 1,729929220089790
atan(0.5) = 26,578525356734108572791
exp(1) = 2,7182818284590452353602874
exp(0) = 1
power(5,2) = 25
power(25,1/2) = 5,0000000000000000000000000000
power(5,-2) = ,04
sqrt(16) = 4
log(100,10) = ,5
log(10,100) = 2
log(2,16) = 3,9999999999999999999999999999999
sign(-25.7) = -1
sign(25.7) = 1
sign(0) = 0
greatest(-1,3,45,9,1) = 45
```

49. Встроенные функции языка PL/SQL. Функции регулярных выражений

Регулярные выражения – формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов.

- REGEXP_LIKE выбирает все строки, соответствующие заданному шаблону
- REGEXP_INSTR определяет местоположение вхождения шаблона в строку

- **REGEXP_REPLACE** заменяет шаблон выражения на заданный
- **REGEXP_SUBSTR** выделяет из строки шаблон
- **REGEXP_COUNT** определяет количество вхождений

Метасимволы привязки:

метасимвол	описание	пример	Результат
^	К началу строки	REGEXP_LIKE(str,'^t')	test11 => true 11123345 => f
\$	К концу строки	REGEXP_LIKE(str,'\$5')	test11 => false 11123345 => true
Квантификатора + операторы повтора			
*	Встреч.0+	REGEXP_REPLACE(str, '11*', '1')	test11 => test1
?	0/1		
+	1+	REGEXP_LIKE(str,'5+')	test11 => false
{m}	m раз	REGEXP_LIKE(str,'3{2}')	11123345 => true
{m,}	по крайне мере m От m до n		
{m,n}			
.....			

50. Коллекции. Массивы переменной длины.

Коллекция – структура данных, содержащая элементы одного типа

Элементом коллекции может быть как скалярная величина, так и композитные данные

Элементы коллекций можно сравнивать между собой на эквивалентность

Можно передавать параметром

Одномерная, но можно создавать коллекции коллекций

Виды коллекций: Массив переменной длины VARRAY, Вложенная таблица (nested tables), Ассоциативный массив (associative array).

Коллекция называется **ограниченной**, если заранее определены границы возможных значений индексов ее элементов, иначе **неограниченной**

Коллекции типа VARRAY всегда ограничены

Вложенные таблицы и ассоциативные массивы не ограничены.

Коллекция называется плотной, если все ее элементы, определены и каждому из них присвоено некоторое значение (таковым может быть и NULL)

Массивы VARRAY всегда являются плотными

Вложенные таблицы первоначально всегда плотные, но по мере удаления некоторых элементов становятся разреженными

Ассоциативные массивы могут быть как разреженными, так и плотными в зависимости от способа их заполнения

Работа с коллекциями:

1. Объявление коллекций

2. Инициализация коллекций

2.1. Явно с помощью конструктора

2.2. Неявно при выборке из базы данных

2.3. Прямыми присвоением переменной с другой коллекции такого же типа

1. Добавление и удаление элементов

3.1. Вложенные таблицы и массивы переменной длины – сначала увеличить размер при помощи функции EXTEND, а затем присвоить значения новым элементам

3.2. Ассоциативный массив – присвоение значения новому элементу

Массивы переменной длины – одномерные, связанные коллекции однотипных – элементов

Доступны в рамках PL/SQL и в БД

Являются плотными

```

declare
    type myarraytype  is varray(3) of number;
    va myarraytype;

begin
    for i in 1..3 loop
        va(i) := 1;
    end loop;

exception
    when others then dbms_output.put_line(sqlerrm);
end;

```

51. Коллекции. Вложенные таблицы.

- Коллекция – структура данных, содержащая элементы одного типа.
Элементом коллекции может быть как скалярная величина, так и композитные данные
- Коллекция состоит из **набора** элементов, причем каждый элемент находится в определенной позиции (имеется **индекс** элемента)
- Необходимо объявить **тип коллекции** – командой TYPE
- Необходимо объявить **коллекцию** – переменную этого типа для дальнейшего использования
- Коллекция называется плотной, если все ее элементы, определены и каждому из них присвоено некоторое значение
- Вложенные таблицы первоначально всегда плотные, но по мере удаления некоторых элементов становятся разреженными

Вложенные таблицы (Nested tables) – одномерные, несвязанные коллекции однотипных элементов. Вложенную таблицу можно рассматривать как одномерный массив, в котором индексами служат значения целочисленного типа.

Свойства:

Могут использоваться как в SQL (тип столбца в таблице), так и в PL\SQL коде

Содержат однородные данные, т.е. все строки имеют одинаковую структуру

- данных
- Требуется инициализация, при использовании в PL\SQL
- Порядок элементов не зафиксирован
-

- Используется память PGA (для всех 3 типов коллекций)
- При попытке чтения элемента с несуществующим индексом -> исключение NO_DATA_FOUND

Использование вложенных таблиц в SQL

CREATE TYPE XX_TYPE_CHAIR IS TABLE OF VARCHAR2(100);

Методы для вложенных таблиц:

Exists(n)- Возвращает TRUE, если элемент существует; FALSE — если элемент не существует

Count - Возвращает текущее количество элементов

First/Last - Возвращает индекс первого и последнего элемента

Prior(n) - Возвращает индекс предыдущего элемента

Next(n) - Возвращает индекс следующего элемента

Extend - Добавляет один пустой элемент в коллекцию

Delete - Удаляет все элементы в коллекции, не вызывает исключение при несуществующем индексе

```
-- 14/01.sql
declare
    type tperson is record
    (
        name teacher.teacher_name%type,
        pulpit teacher.pulpit%type
    );
    type mytable is table of tperson;
    tb mytable := mytable();
    rec tperson; -- := tperson('xxx','xxxxxx');
begin
    dbms_output.put_line('tb.count = '||tb.count);

    tb.extend;
    dbms_output.put_line('tb.count = '||tb.count);
    tb(1).name := 'Урбанович Павел Павлович';
    tb(1).pulpit := 'ИСиТ';
    tb.extend;
    dbms_output.put_line('tb.count = '||tb.count);
    tb(2).name := 'Дятко Александр Аркадьевич';
    tb(2).pulpit := 'ИСиТ';
    tb.extend;
    dbms_output.put_line('tb.count = '||tb.count);
    tb(3).name := 'Кабайло Александр Сергеевич';
    tb(3).pulpit := 'ИСиТ';
    for i in 1..tb.count
    loop
        dbms_output.put_line( ' '||i||' '||tb(i).name||' '||tb(i).pulpit);
    end loop;
exception
    when others then dbms_output.put_line(sqlerrm);
end;
/

```

```
Dbms Output
+ - Buffer
shl-svvcore x
set serveroutput on
tb.count = 0
tb.count = 1
tb.count = 2
tb.count = 3
1 Урбанович Павел Павлович
2 Дятко Александр Аркадьевич
3 Кабайло Александр Сергеевич
```

52. Коллекции. Ассоциативные массивы.

Ассоциативные массивы могут быть как разреженными, так и плотными в зависимости от способа их заполнения

Работа с коллекциями:

1. Объявление коллекций
2. Инициализация коллекций
 - 2.1. Явно с помощью конструктора
 - 2.2. Неявно при выборке из базы данных
 - 2.3. Прямыми присвоением переменной с другой коллекции такого же типа
 1. Добавление и удаление элементов

3.1. Вложенные таблицы и массивы переменной длины – сначала увеличить размер при помощи функции EXTEND, а затем присвоить значения новым элементам

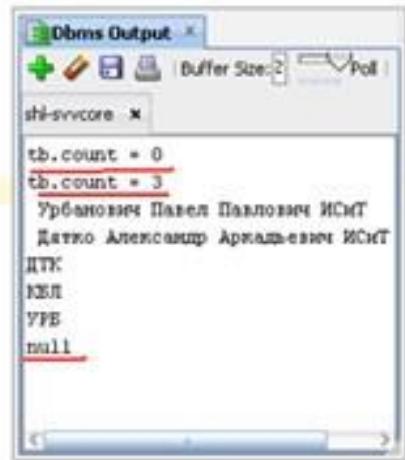
3.2. Ассоциативный массив – присвоение значения новому элементу

Ассоциативные массивы – одномерные, неограниченные (по максимальному количеству элементов при создании) коллекции элементов

Доступны только в рамках PL/SQL

Изначально являются разреженными, индекс может принимать непоследовательные значения

```
declare
    type tperson is record (name teacher.teacher_name%type, pulpit teacher.pulpit%type );
    type mytable is table of tperson index by teacher.teacher_name%type;
    tb mytable;
    ntx teacher.teacher_name%type;
    rec tperson;
begin
    dbms_output.put_line('tb.count ='||tb.count);
    tb('УРБ').name := 'Урбанович Павел Павлович';
    tb('УРБ').pulpit := 'ИСИТ';
    tb('ДТК').name := 'Датко Александр Аркадьевич';
    tb('ДТК').pulpit := 'ИСИТ';
    tb('МЕЛ').name := 'Мабаево Александра Сергеевна';
    tb('МЕЛ').pulpit := 'ИСИТ';
    dbms_output.put_line('tb.count ='||tb.count);
    dbms_output.put_line(''||tb('УРБ').name||'||tb('УРБ').pulpit);
    rec := tb('ДТК');
    dbms_output.put_line(''||rec.name||'||rec.pulpit);
    ntx := tb.first;
    if ntx is not null then dbms_output.put_line(ntx);
    else dbms_output.put_line('null');
    end if;
    ntx := tb.next(ntx);
    if ntx is not null then dbms_output.put_line(ntx);
    else dbms_output.put_line('null');
    end if;
    ntx := tb.next(ntx);
    if ntx is not null then dbms_output.put_line(ntx);
    else dbms_output.put_line('null');
    end if;
    ntx := tb.next(ntx);
    if ntx is not null then dbms_output.put_line(ntx);
    else dbms_output.put_line('null');
    end if;
exception
    when others then dbms_output.put_line(sqlerrm);
end;
```



Index	Name	Pulpit
1	Урбанович Павел Павлович	ИСИТ
2	Датко Александр Аркадьевич	ИСИТ
3	Мабаево Александра Сергеевна	ИСИТ

53. Процедурные объекты. Хранимые процедуры. Вызов процедур. Входные и выходные параметры, позиционный и параметрический форматы передачи фактических параметров. Значения параметров по умолчанию.

Процедура – именованный модуль, который выполняет одно или несколько выражений и может принимать или возвращать значения через список параметров.

Для создания процедур необходима привилегия create procedure

Типы данных параметров: PL/SQL или программно-определенный, Не может быть ограничен по размеру, Размер определяется через вызывающую программу или через связанное объявление переменной

Типы параметров: IN, OUT, IN OUT.

При выполнении: Значения OUT устанавливаются в NULL ; Значения IN OUT остаются неизменными; При ошибке присвоения для параметров откатываются, кроме NOCOPY.
IN, IN OUT можно не задавать при вызове.

```
create or replace procedure svvcore.xsum(
    min_cy in svvcore.auditorium.auditorium_capacity%type, -- минимальная вместимость
    max_cy in out svvcore.auditorium.auditorium_capacity%type, -- максимальная вместимость
    n_aud out number -- количество
)
is
    m_max_cy svvcore.auditorium.auditorium_capacity%type;
    m_n_aud number := 0;
begin
    select count(*), max(auditorium_capacity) into m_n_aud, m_max_cy from svvcore.auditorium
    where auditorium_capacity >= min_cy and auditorium_capacity <= max_cy;
exception
    when others then dbms_output.put_line(sqlerrm);
end xsum;
```

Передача параметров:

Позиционный – каждое значение в списке аргументов вызова ставится в соответствие формальному параметру по порядку. *Empid_to_name(23, name, surname);*

Именованный – явно связывает аргументы при вызове с параметрами по именам.

Empid_to_name(in_id =>23, out_name=> name, out_surname =>surname);

Можно комбинировать оба метода, пока позиционные аргументы стоят слева.

Empid_to_name(23, name, out_surname =>surname);

Вызов процедур:

```

SVVCORE@sh1> exec :max_cy:=10;
Процедура PL/SQL успешно завершена.

SVVCORE@sh1> exec xsum(20,:max_cy,:n_aud);
Процедура PL/SQL успешно завершена.

SVVCORE@sh1> select :max_cy, :n_aud from dual;
          :MAX_CY      :N_AUD
----- -----
          0
-----
```

```

SVVQUEST@sh1>
SVVQUEST@sh1>
SVVQUEST@sh1> CALL svvcore.xsum(20,:max_cy,:n_aud);
Вызов завершен.

SVVQUEST@sh1> select :max_cy, :n_aud from dual;
          :MAX_CY      :N_AUD
----- -----
          60          5
-----
```

```

-- 16/03.sql
declare
    max_cy number(3) := 80;
    n_aud  number(3):= 0;
begin
    svvcore.xsum(20, max_cy, n_aud);
    dbms_output.put_line(max_cy || ', ' || n_aud);

    xsum(n_aud=>n_aud, min_cy =>20, max_cy=>max_cy);
    dbms_output.put_line(max_cy || ', ' || n_aud);

    xsum(20, n_aud=>n_aud, max_cy=>max_cy);
    dbms_output.put_line(max_cy || ', ' || n_aud);

end;
```

Значения по умолчанию - DEFAULT

```
-- 16/04.sql
create or replace procedure svvcore.xsum(
    min_cy in svvcore.auditorium.auditorium_capacity%type default 20, -- минимальная вместимость
    max_cy in out svvcore.auditorium.auditorium_capacity%type, -- максимальная вместимость
    n_aud out number -- количество
)
is
    no_max_cy exception;
begin
    select count(*), max(auditorium_capacity) into n_aud, max_cy from svvcore.auditorium
    where auditorium_capacity >= min_cy and auditorium_capacity <= max_cy;
    if n_aud is null
        then raise no_max_cy;
    end if;
exception
    when no_max_cy then return;
    when others then dbms_output.put_line(sqlerrm);
end xsum;
```

54. Процедурные объекты. Хранимые функции. Параметры функции. Вызов функций. Понятие детерминированной функции. Понятие pipeline функции. Значения параметров по умолчанию.

Функции — эти подпрограммы возвращают одно значение; в основном используется для вычисления и возврата значения.

- 1) Автономная функция создается с помощью оператора **CREATE FUNCTION**
- 2) Список необязательных параметров содержит имя, режим и типы параметров. IN представляет значение, которое будет передано извне, а OUT представляет параметр, который будет использоваться для возврата значения вне процедуры
- 3) Функция должна содержать инструкцию **возврата**
- 4) Предложение *RETURN* указывает тип данных, который вы собираетесь вернуть из функции
- 5) *Функция body* содержит исполняемую часть.
- 6) Ключевое слово AS используется вместо ключевого слова IS для создания отдельной функции.
- 7) ПРИМЕР

```
CREATE OR REPLACE FUNCTION totalCustomers
```

```
RETURN number IS
```

```
total number(2) := 0;
```

```

BEGIN
SELECT count(*) into total
FROM customers;
RETURN total;
END;

```

Функция называется детерминированной, если при одном наборе параметров IN и IN OUT она всегда возвращает одно и то же значение. Функция должна быть объявлена с ключевым словом DETERMINISTIC.

Суть в том, что у нас например есть параметры param_1 и param_2 и каждый раз когда мы будем их передавать мы всегда будем получать одно и тоже, поэтому с ключевым словом deterministic мы получим детерминированную функцию, которая гораздо оптимизированнее !

- Pipeline function- Конвейерная обработка позволяет табличной функции быстрее возвращать строки и может уменьшить объем памяти, необходимый для кэширования результатов табличной функции. Конвейерные табличные функции включают фразу **PIPELINED** и используют вызов **PIPE ROW**, чтобы вытолкнуть строки из функции, как только они создадутся, вместо построения табличной коллекции. Заметим, что вызов **RETURN** пустой, поскольку нет никакой коллекции, возвращаемой из функции.
- - Построение конвейерной табличной функции.

```

CREATE OR REPLACE FUNCTION get_tab_ptf (p_rows IN NUMBER) RETURN
t_tf_tab PIPELINED AS

```

```

BEGIN
FOR i IN 1 .. p_rows LOOP
PIPE ROW(t_tf_row(i, 'Description for ' || i));
END LOOP;
RETURN;
END;

```

- - Тестирование

```

SELECT *
FROM TABLE(get_tab_ptf(10))

```

ORDER BY id DESC;

- Параметры по умолчанию – ключевое слово default – задаем какое-то дефолтное значение для переменной, которую передаем в функцию

Параметры бывают 1) IN параметр Этот параметр используется для ввода данных в подпрограммы В вызывающем операторе эти параметры могут быть переменной, литеральным значением или выражением, например, это может быть арифметическое выражение типа «5 * 8» или «a / b», где «a» и «b» являются переменными По умолчанию параметры имеют тип IN. 2) Выходной параметр Этот параметр используется для получения выходных данных из подпрограмм. В операторе вызова эти параметры всегда должны быть переменной для хранения значения из текущих подпрограмм. 3) Параметр IN OUT Этот параметр используется как для ввода, так и для получения выходных данных из подпрограмм. Это переменная чтения-записи внутри подпрограмм. Их значения могут быть изменены внутри подпрограмм.

55. Процедурные объекты. Пакеты. Спецификация и реализация пакета.

Пакеты – коллекция pl/sql объектов, сгруппированных вместе.

Нажера-Что делает: 1) скрытие инфы, 2) объектно-ориентированный дизайн, 3) постоянство объектов в транзакциях, 4) улучшенная производительность.

Можно включать: процедуры, функции, константы, исключения, курсоры, переменные, TYPE выражения, записи, REF курсоры

Спецификация пакета (package) – обязательна, содержит список объектов для общего доступа из других модулей или приложения

```
CREATE OR REPLACE PACKAGE time_pkg IS
    FUNCTION GetTimestamp RETURN DATE;
    PROCEDURE ResetTimestamp(new_time DATE DEFAULT SYSDATE);
END time_pkg;
```

Реализация пакета (package body) – содержит реализацию процедур и функций из спецификации; приватные объекты; секцию инициализации.

```


CREATE OR REPLACE PACKAGE BODY time_pkg IS
    StartTimeStamp DATE := SYSDATE;
    -- StartTimeStamp is package data.
-- Function
FUNCTION GetTimestamp RETURN DATE IS
BEGIN
    RETURN StartTimeStamp;
END GetTimestamp;
-- Procedures
PROCEDURE ResetTimestamp(new_time DATE DEFAULT SYSDATE)
IS
BEGIN
    StartTimeStamp := new_time;
END ResetTimestamp;
-- Initialization section
BEGIN
    null;
EXCEPTION
    WHEN NO_DATA_FOUND
        THEN dbms_output.put_line('not initialized');
END time_pkg;


```

Вызов пакета: *Package_name.package_element;*

Пакетные данные – структуры данных в пакете.

Пакетные переменные являются глобальными данными (сохраняют своё состояние от одной транзакции к другой)

56. Процедурные объекты. Триггеры. Виды триггеров.

Классификация, порядок выполнения и предикаты триггеров.

Триггеры замещения. Привилегии для создания триггеров.

Включение/отключение триггеров. Псевдозаписи old и new.

Триггер – особый вид процедур, кот. срабатывают по запускающему их событию

Применение триггеров:

- для реализации сложных ограничений целостности базы данных;
- для аудита (контроля хранимой и изменяемой информации);
- для автоматического оповещения программ о произошедших событиях;

Виды триггеров:

- *по привязанному объекту*: на таблицу, на представлении (instead of trigger)
- *по событиям запуска*: insert, update, delete
- *по области действия*: уровень оператора, ур. записи, составные

- по времени срабатыванию: before (до записи в журнал), after

Уровни триггеров:

- for each row – сраб для каждой измененной строки
- по умолч (операторный ур) – сраб 1 раз на триггерное событие

Порядок выполнения:

- операторные BEFORE
- для каждой строки BEFORE
- выполняется оператор
- для каждой строки AFTER
- операторный AFTER

Предикаты триггеров:

Чтобы различать DML команды и события, которые выполняют триггер, используются триггерные предикаты INSERTING, UPDATING, and DELETING в условиях IF

create or replace trigger ..

after insert or update or delete on ..

begin

if inserting then ...

elsif updating then ...

elsif deleting then ...

Триггеры замещения: INSTEAD OF – только для предст! только уровня строки!

Привилегии:

- create (any) trigger
- alter any trigger
- drop any trigger
- administer database trigger – созд/изм/уд системные триггеры

Вкл / откл триггеров:

- alter trigger { disable | enable }
- всех для таблиц : alter table .. { enable | disable } all triggers;

- компиляция триггера : alter trigger .. compile
- переименование триггера

Псевдозаписи new, old:

операция срабатывания	OLD.column	NEW.column
insert	null	новое знач
update	старое знач	новое знач
delete	старое знач	null

57. Секционирование таблиц. Виды секционирования.

Секционирование – метод хранения сегмента данных (напр. таблица) физически в виде нескольких сегментов (логически – монолитная структура).

Секция – отдельный сегмент. Могут находиться в разных ТП (сл-но на разных дисках). Отдельные ТП могут находиться в OFFLINE, не нарушая работоспособности всей таблицы.

Секции имеют: 1) общие логические атрибуты – имена, типы, ограничения столбцов, ROW_MOVEMENT; 2) собственные физические атрибуты – например атрибуты ТП.

Операции над секциями:

1. **RENAME, DROP, MOVE, ADD**
2. **SPLIT** – разбиваем секцию.

```
ALTER TABLE DEACore.sales SPLIT PARTITION sales_llq4
AT ('01-JUL-2012') INTO
(PARTITION sales_sp1 TABLESPACE data01,
PARTITION sales_sp2 TABLESPACE data02)
```

3. **MERGE** – объединяем секции.

```
ALTER TABLE DEACore.sales MERGE PARTITIONS
sales_llq1, sales_llq2 INTO sales_ml;
```

4. **EXCHANGE** – пихаем данные секции в какую-то таблицу (вроде бы).

```
ALTER TABLE DEACore.sales EXCHANGE PARTITION sales_llq2
WITH TABLE DEACore.my_sales WITHOUT VALIDATION;
```

Мы можем читать данные прямо из секции:

```
SELECT * FROM DEACore.sales PARTITION (sales_llq1);
```

Если читаем так, то называется ассоциативная ссылка (хз почему):

```
SELECT * FROM DEACore.sales PARTITION FOR ('02-FEB-2011');
```

Виды секционирования:

1. Диапазонное (RANGE) – определяем секции ручками. Обязательно указываем MAXVALUE секцию. Проблема: При загрузке новых данных в таблицу нужно постоянно расщеплять секцию MAXVALUE

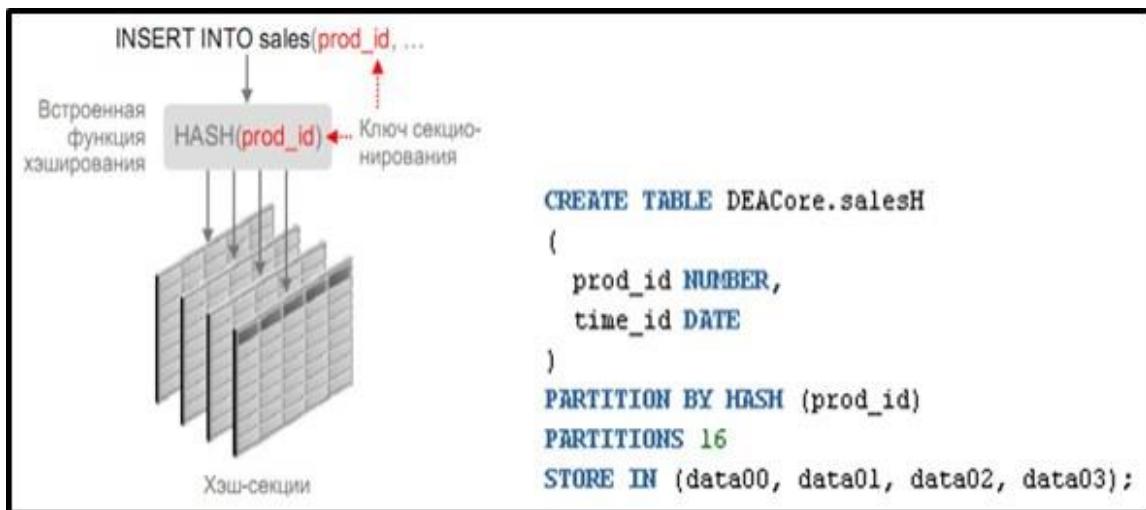
```
CREATE TABLE DEACore.sales
(
    prod_id NUMBER,
    time_id DATE
)
PARTITION BY RANGE (time_id)
(
    PARTITION sales_llq1 VALUES LESS THAN ('01-APR-2011') TABLESPACE data00,
    PARTITION sales_llq2 VALUES LESS THAN ('01-JUL-2011') TABLESPACE data01,
    PARTITION sales_llq3 VALUES LESS THAN ('01-OCT-2011') TABLESPACE data02,
    PARTITION sales_llq4 VALUES LESS THAN ('01-JAN-2012') TABLESPACE data03,
    PARTITION sales_max VALUES LESS THAN (maxvalue) TABLESPACE data04
);
```

2. Интервальное – Создается единственная диапазонная секция без MAXVALUE.

Новые секции будут создаваться автоматически (по 1ой операции INSERT, не попадающей в диапазоны существующих секций). Предустановленный интервал можно изменить через alter table set interval

```
create table t3 (x1 nvarchar2(50), x2 number(3))
partition by range (x2)
interval (100) store in (users)
(
    partition t11 values less than (101)
);
```

3. **Хэш-секционирование** – на основе заполняемых данных (например, primary key) вычисляется хэш, который потом используется для быстрого доступа.



4. **Списочное** – разбиваем по списку дискретных значений

```
CREATE TABLE DEACore.salesL
(
    prod_id NUMBER,
    time_id DATE,
    group_id NUMBER
)
PARTITION BY LIST (group_id)
(
    PARTITION sales_11 VALUES(3, 9),
    PARTITION sales_12 VALUES(5, 4),
    PARTITION sales_other VALUES (DEFAULT)
);
```

5. **Композитное** – Если данных много настолько, что сами секции большие, то их тоже можно секционировать. Такие секции могут секционироваться по другому критерию секционирования. Допускается 2 уровня секционирования (секции и подсекции).

```

CREATE TABLE DEACore.salesC
(
    prod_id NUMBER,
    time_id DATE,
    cust_id NUMBER
)
PARTITION BY RANGE (time_id)
SUBPARTITION BY HASH(cust_id)
SUBPARTITIONS 4 STORE IN (data00, data01, data02, data03)
(
    PARTITION sales_11q1 VALUES LESS THAN ('01-APR-2011'),
    PARTITION sales_11q2 VALUES LESS THAN ('01-JUL-2011')
);

```

Зачем секционирование:

1. **Отказоустойчивость** – данные могут располагаться на разных дисках
2. **Производительность** – тут по двум причинам. 1) Читаем из разных секций параллельно. 2) Например, у нас два клиента – Microsoft и Google. Мы храним их инфу в одной таблице. Следовательно, когда нам потребуется какая-то выборка данных по Google'у, мы будем ещё читать и данные Microsoft. При секционировании мы можем разделить эти данные на разные секции и сразу читать данные только Google'а, хотя логически всё будет в одной куче.

58. Транзакции. Виды транзакций. Понятие автономной транзакции.

Транзакция – это логическая единица работы в базе данных Oracle, состоящая из одного или более операторов SQL. Транзакция начинается с первого исполняемого оператора SQL и завершается, когда вы фиксируете или отказываете транзакцию. Фиксация (committing) транзакции закрепляет проведенные вами изменения, а откат (roll back) – отменяет их.

Как только вы зафиксировали транзакцию, все прочие транзакции других

- пользователей, которые начались после нее, смогут видеть изменения, проведенные вашими транзакциями

Когда транзакция вообще не может выполниться (скажем, из-за отключения

- электропитания), то она вся целиком должна быть отменена. Oracle откатывает все изменения, проведенные предшествующими операторами SQL, возвращая данные в исходное состояние

ACID properties- основные правила которых подчиняются транзакции:

1. Atomicity- Выполняются все задачи транзакции или ни одна из них
2. Consistency - Транзакция переводит базу данных из одного согласованного состояния в другое согласованное состояние(последовательность)
3. Isolation - Эффект транзакции не виден другим транзакциям, пока транзакция не будет зафиксирована
4. Durability - Изменения, внесенные совершенными транзакциями, являются постоянными(долговечность).

Автономная транзакция является независимой сделкой , которая может быть вызвана из другой транзакции, которая является основной транзакцией.

Автономные транзакции полезны для действий, которые должны выполняться независимо, независимо от того, фиксирует или откатывает вызывающая транзакция. Например, в транзакции покупки акций вы хотите зафиксировать данные о клиентах независимо от того, проходит ли покупка акций в целом. Кроме того, вы хотите зарегистрировать сообщения об ошибках в таблице отладки, даже если вся транзакция откатывается.

Автономные транзакции имеют следующие характеристики:

1. Автономная транзакция не видит незафиксированных изменений
2. Изменения в автономной транзакции видны другим транзакциям после фиксации автономных транзакций
3. Автономные транзакции могут запускать другие автономные транзакции.

РАСПРЕДЕЛЕННЫЕ транзакции - В отличие от транзакции в локальной базе данных, распределенная транзакция изменяет данные в нескольких базах данных.

59. Обработка заданий. Системные пакеты обработки заданий в Oracle.

DBMS_JOB – поддержка управления заданиями

Задание – процедура, PL-SQL блок, внешняя процедура

*вып-ся в фон.реж., надо задать кол-во одноврем.вып-мых процессов

```
ALTER SYSTEM SET JOB_QUEUE_PROCESSES=9;
```

SUBMIT – создание задания

```
declare job_number user_jobs.job%type;
begin
dbms_job.submit(job_number, 'begin up_students_grades_JOB; end;',
    SYSDATE, 'SYSDATE + 60/86400'); -- 24 * 60 * 60 = 86 400
COMMIT;
SYS.dbms_output.put_line(job_number);
end;
```

ISUBMIT – создание задания с номером

```
--isubmit
declare job_number user_jobs.job%type;
begin
dbms_job.isubmit(43, 'begin up_students_grades_JOB; end;',
    SYSDATE, 'SYSDATE + 60/86400'); -- 24 * 60 * 60 = 86 400
COMMIT;
select job into job_number from user_jobs;
SYS.dbms_output.put_line(job_number); --43
end;
```

REMOVE – удаление задания из очереди

RUN – немедленное выполнение задания в пользовательском сеансе

```
-- run
declare job_number user_jobs.job%type;
begin
dbms_job.run(43);
COMMIT;
select job into job_number from user_jobs;
SYS.dbms_output.put_line(job_number); --43
end;
```

BROKEN – разрушение задания (16)

```
begin
dbms_job.broken(43,true, null);
end;
```

INSTANCE – указание экземпляра

NEXT_DATE – изменение времени выполнения

INTERVAL – изменение интервала выполнения

```
-- next_date -- interval
begin
dbms_job.next_date(43,SYSDATE + 2/24); -- через 2 часа
dbms_job.interval(43,SYSDATE + 3); -- через 31 дня
COMMIT;
end;
```

CHANGE – изменение параметров задания

```
-- change
declare job_number user_jobs.job%type;
begin
dbms_job.change(43, -- номер задания
                null, --what
                null, -- следующее исполнение
                'SYSDATE + 360/86400'); --интервал
COMMIT;
end;
```

WHAT – изменение задания

DBMS_SCHEDULER

Schedule – расписание

Расписание — это спецификация того, когда и как часто база данных должна выполнять задание.

```
--scheduler
begin
dbms_scheduler.create_schedule(
    schedule_name =>'Sch_1',
    start_date =>'01/01/2017 12:00:00',
    repeat_interval =>'FREQ=DAILY',
    comments => 'Sch_1 DAILY at 12:00');
|
end;
```

Program – программа

Программа содержит метаданные о задании Scheduler. Программа включает имя, тип (например, код PL/SQL или сценарий оболочки UNIX) и действие программы, которое представляет собой действительное имя процедуры или исполняемого сценария.

```
--program
begin
dbms_scheduler.create_program(
    program_name =>'Pr_1',
    program_type =>'STORED_PROCEDURE',
    program_action =>'up_students_grades_JOB',
    number_of_arguments =>0,
    enabled => false,
    comments => 'up_students_grades_JOB');
end;
```

Job – плановая программа

Job— это задача, которая планируется для однократного или многократного автоматического запуска. Задание содержит спецификацию того, что и когда должно быть выполнено.

Одним из ограничений пакета *DBMS_JOB* является то, что он может выполнять только задания на базе PL/SQL, и его нельзя использовать для планирования запуска системных сценариев либо исполняемых файлов. Scheduler позволяет использовать сценарии PL/SQL, сценарии оболочки операционной системы, программы Java, и родные двоичные исполняемые файлы для выполнения запланированных заданий.

```
--job
begin
dbms_scheduler.create_job(
    job_name =>'J_1',
    program_name =>'Pr_1',
    schedule_name =>'Sch_1',
    enabled => true);
end;
```

Представления словаря: DBA_JOBS, USER_JOBS,
USER_SCHEDULER_SCHEDULES,
USER_SCHEDULER_PROGRAMS, USER_SCHEDULER_JOBS,
USER_SCHEDULER_JOB_LOG.

▶ **Права:**

- ▶ Роль SCHEDULERADMIN
- ▶ Права на объекты

60. Системные пакеты Oracle.

Системные пакеты Oracle:

- устанавливается во время установки Oracle
- используется для расширения функциональных возможностей Oracle
- владелец пакетов – SYS
- написаны на C или plsql

```
SELECT * FROM DBA_Objects_AE  
WHERE Object_Type = 'PACKAGE'  
ORDER BY Owner, Object_Name;
```

Пакеты APEX:

- apex – oracle application express – среда разработки веб-приложений
- apex_custom_auth – проверка подлинности управляемым сеансом
- apex_application – использует глобальные переменных
- apex_item – создание элементов форм на основе sql-запроса
- apex_util – различные утилиты состояния сеанса, файлов, авторизации ...

Пакеты DBMS:

- dbms_advanced_rewrite – перехват и замена sql-запросов
- dbms_advisor – часть набора экспертной системы для решения проблем производительности, связанных с компонентами сервера БД
- dbms_sqltune – сбор статистики, используется при анализе производительности sql-операторов
- dbms_appl_info – присвоение имени процессу для удобства мониторинга и отладки

DBMS_AQ – пакет для Advanced Queuing – система обмена очередями сообщений

DBMS_TRANSFORM – пакет для Oracle Advanced Queuing – преобразования данных

DBMS_DEFER – вызов удаленных процедур

DBMS_OFFLINE_OG, DBMS_REPCAT – поддержка репликации

DBMS_REFRESH – создание группы материализованных представлений для обновления как единого целого

DBMS_CRYPTO – пакет шифрования данных. Поддерживаются алгоритмы шифрования - DES, AES, RC4, 3DES, 3DES-2KEY

DBMS_CUBE - поддерживает развертывание кубических материализованных представлений из существующих реляционных материализованных представлений

DBMS_DIMENSION - отображение информации об измерениях – иерархических связях между данными

Data Warehouse - предметно-ориентированная база данных, предназначенная для отчетов и/или OLAP

DBMS_DEBUG – пакет для отладки PL/SQL-кода на сервере

- » Селективность таблицы — значение, представляющее долю строк таблицы, удовлетворяющих определенному условию выбора

- ▶ Селективность таблицы — значение, представляющее долю строк таблицы, удовлетворяющих определенному условию выбора
- ▶ Селективность таблицы связана с условием выбора строк
- ▶ Селективность индекса — значение, представляющее отношение количества уникальных значений индексируемых столбцов к общему числу строк таблицы
- ▶ Селективность индекса показывает долю строк от общего числа строк в таблице, которое приходится на одно значение индекса

1. SQLite. Общая характеристика СУБД. Архитектура. Внутренние таблицы.

SQLite - это легковесная реляционная СУБД (система управления базами данных), которая предлагает встроенное хранение и обработку данных в одном файле базы данных без необходимости внешних серверов или настроек. Она является открытым исходным кодом и широко используется во встраиваемых системах, мобильных приложениях и других проектах, где требуется компактное и простое в использовании хранилище данных.

Архитектура SQLite:

Ядро (Core): Ядро SQLite реализует основную функциональность СУБД, включая парсинг SQL-запросов, оптимизацию запросов, управление транзакциями, обработку индексов и другие основные задачи.

Парсер (Parser): Парсер анализирует входной SQL-код и создает соответствующее внутреннее представление, называемое деревом разбора (parse tree). Он проверяет синтаксис запроса и строит структуру данных для дальнейшей обработки.

Оптимизатор (Optimizer): Оптимизатор анализирует дерево разбора запроса и оптимизирует его для достижения наилучшей производительности. Он выбирает оптимальный план выполнения запроса, используя информацию о доступных индексах, статистику таблиц и другие факторы.

Виртуальная машина (Virtual Machine): Виртуальная машина SQLite выполняет оптимизированный план выполнения запроса, выполняя необходимые операции чтения, записи и обработки данных. Она взаимодействует с ядром SQLite и управляет доступом к файлу базы данных.

Система хранения (Storage System): Система хранения отвечает за организацию данных в файле базы данных. SQLite использует специальный формат файла, который позволяет эффективно сохранять, извлекать и модифицировать данные. Она поддерживает транзакции, блокировку и механизмы обеспечения целостности данных.

Внутренние таблицы SQLite:

SQLite использует несколько внутренних таблиц для управления базой данных. Некоторые из них включают:

`sqlite_master`: Эта таблица содержит метаданные о других таблицах в базе данных, таких как их имена, типы и создающие SQL-запросы.

`sqlite_sequence`: Если в базе данных определены таблицы с автоинкрементными полями (например, поле `INTEGER PRIMARY KEY AUTOINCREMENT`), SQLite использует эту таблицу для хранения информации о следующих доступных значениях для каждой таблицы.

`sqlite_stat1`: Данная таблица содержит статистическую информацию о данных, которая используется оптимизатором запросов для выбора оптимального плана выполнения запроса. Она содержит информацию о распределении значений в столбцах таблиц.

Это лишь несколько примеров внутренних таблиц SQLite. Система хранения SQLite также использует различные структуры данных и индексы для ускорения доступа к данным и обеспечения эффективной работы с базой данных.

2. SQLite. Таблицы. Классы хранения и аффинированность типов.

В SQLite таблицы представляют собой основные структуры для хранения данных. Они определяются с помощью оператора `CREATE TABLE`. При создании таблицы в SQLite можно определить класс хранения и аффинированность типов данных.

Классы хранения (Storage Classes):

`NULL`: Представляет отсутствие значения или нулевое значение.

`INTEGER`: Целочисленные значения.

`REAL`: Значения с плавающей точкой (вещественные числа).

`TEXT`: Текстовые значения в формате UTF-8 или UTF-16.

`BLOB`: Бинарные данные, такие как изображения или файлы.

Аффинированность типов (Type Affinity):

В SQLite типы данных имеют аффинированность, которая определяет, как SQLite преобразует значения между различными классами хранения. Это влияет на операции сравнения, сортировки и преобразования типов данных.

Некоторые примеры аффинированности типов данных в SQLite:

`INTEGER`: Может быть аффинирован к `INTEGER`, `REAL` или `TEXT`.

`REAL`: Может быть аффинирован к `REAL` или `TEXT`.

`TEXT`: Может быть аффинирован к `TEXT`.

`BLOB`: Может быть аффинирован к `BLOB`.

3. SQLite. Особенности сравнения, сортировки и группировки дат, строк и числовых данных.

В SQLite сравнение, сортировка и группировка данных выполняются с учетом следующих особенностей:

Сравнение дат: В SQLite даты могут храниться в различных форматах, включая текстовые значения или числовые значения (эпоха UNIX). При сравнении дат SQLite использует правила сравнения, определенные для каждого конкретного формата даты. Например, если даты хранятся в текстовом формате `YYYY-MM-DD`, то они будут сравниваться лексикографически, что может дать ожидаемый результат для правильно отформатированных дат.

Сравнение строк: Строки в SQLite сравниваются по умолчанию с учетом лексикографического порядка, используя кодировку UTF-8 или UTF-16 в зависимости от конкретной таблицы и столбца. Однако можно использовать различные функции и операторы сравнения для определения нестандартных правил сравнения, например, нерегистрозависимого сравнения.

Сортировка данных: При выполнении сортировки SQLite использует алгоритм сортировки, который учитывает типы данных и аффинированность столбцов. Для числовых значений и значений с плавающей точкой используется числовая сортировка, а для строк - лексикографическая сортировка.

Группировка данных: При группировке данных SQLite объединяет строки с одинаковыми значениями в указанных столбцах. При этом используется сравнение значений столбцов с учетом правил сравнения, описанных выше.

4. SQLite. Операторы объединения и соединения.

SQLite. Операторы объединения и соединения:

Оператор объединения (UNION): Оператор UNION позволяет объединить результаты двух или более SELECT-запросов в один результат. Результаты объединения включают все уникальные строки из каждого запроса. Формат оператора UNION выглядит следующим образом:

```
sql
Copy code
SELECT column1, column2, ...
FROM table1
UNION
SELECT column1, column2, ...
FROM table2;
```

Этот запрос выберет уникальные строки из table1 и table2 и объединит их в один результат.

Операторы соединения (JOIN): Операторы соединения используются для объединения строк из двух или более таблиц на основе условия соединения. В SQLite поддерживаются различные типы соединений, включая INNER JOIN, LEFT JOIN, RIGHT JOIN и FULL JOIN. Формат оператора соединения выглядит следующим образом:

```
sql
Copy code
SELECT column1, column2, ...
FROM table1
JOIN table2 ON table1.column = table2.column;
В этом примере используется INNER JOIN для объединения строк из table1 и table2 на основе условия равенства значений в столбцах.
```

Кроме INNER JOIN, можно использовать другие операторы соединения для выполнения более сложных операций объединения, таких как LEFT JOIN (левое соединение), RIGHT JOIN (правое соединение) и FULL JOIN (полное соединение).

5. SQLite. Использование обобщённых табличных выражений.

SQLite. Использование обобщённых табличных выражений:

Обобщённые табличные выражения (Common Table Expressions, CTE) в SQLite позволяют создавать временные таблицы, которые можно использовать в рамках одного SQL-запроса. Они упрощают и улучшают читаемость сложных запросов.

Пример использования CTE в SQLite:

```
sql
Copy code
WITH cte_name (column1, column2, ...) AS (
    SELECT column1, column2, ...
    FROM table_name
    WHERE condition
)
SELECT column1, column2, ...
FROM cte_name
WHERE condition;
```

В этом примере мы создаем CTE с именем "cte_name", определяем столбцы, а затем определяем SELECT-запрос, который будет использоваться в CTE. Затем мы можем использовать CTE в основном запросе для выбора столбцов и применения дополнительных условий.

6. SQLite. Ограничения целостности.

SQLite. Ограничения целостности:

Ограничения целостности в SQLite определяют правила, которым должны соответствовать данные в таблицах, чтобы обеспечить их целостность и согласованность. В SQLite поддерживаются следующие ограничения целостности:

Первичный ключ (PRIMARY KEY): Первичный ключ определяет уникальность идентификатора записи в таблице. Он гарантирует, что в столбце или комбинации столбцов не будет дубликатов или значений NULL.

Внешний ключ (FOREIGN KEY): Внешний ключ определяет связь между двумя таблицами. Он обеспечивает ссылочную целостность, гарантируя, что значения внешнего ключа ссылаются на существующие значения в другой таблице.

Уникальное ограничение (UNIQUE): Уникальное ограничение гарантирует, что значения в указанных столбцах являются уникальными, то есть не содержат дубликатов.

Ограничение NOT NULL: Ограничение NOT NULL требует, чтобы значения в указанных столбцах не были NULL.

Ограничения целостности могут быть определены при создании таблицы с помощью оператора CREATE TABLE или изменены или удалены существующие ограничения с помощью операторов ALTER TABLE.

7. SQLite. Указания PRAGMA.

PRAGMA - это команда в SQLite, которая используется для управления различными аспектами базы данных и настройками СУБД. Некоторые распространенные указания PRAGMA в SQLite включают:

PRAGMA database_list: Возвращает список баз данных, доступных в текущем соединении.

PRAGMA table_info(table_name): Возвращает информацию о столбцах таблицы, включая их имена, типы данных и ограничения.

PRAGMA database_name.table_name: Устанавливает или возвращает значения различных параметров для конкретной таблицы, таких как auto_vacuum (автоматическая очистка), journal_mode (режим журнала) и т. д.

PRAGMA foreign_key_list(table_name): Возвращает список внешних ключей, определенных для указанной таблицы.

PRAGMA integrity_check: Проверяет целостность базы данных и возвращает результат проверки.

PRAGMA journal_mode: Устанавливает или возвращает режим журнала базы данных, такой как DELETE (удаление), TRUNCATE (очистка) или WAL (режим работы с журналом Write-Ahead Logging).

Это лишь несколько примеров указаний PRAGMA, доступных в SQLite. Они предоставляют различные функции для настройки и управления базой данных SQLite.

8. SQLite. Виртуальные таблицы.

SQLite. Виртуальные таблицы:

Виртуальные таблицы в SQLite - это специальный тип таблиц, который может быть определен и заполняться не физическими данными из файловой системы, а через программное обеспечение, встроенное в SQLite. Они предоставляют интерфейс для доступа к данным, которые могут быть получены из различных источников, таких как расширения SQLite, внешние библиотеки или пользовательский код.

Примеры виртуальных таблиц в SQLite:

FTS (Full-Text Search) таблицы: Позволяют выполнить полнотекстовый поиск в текстовых данных с помощью встроенного модуля FTS в SQLite.

RTREE таблицы: Реализуют пространственные индексы для быстрого поиска географических объектов.

CSV таблицы: Позволяют чтение и запись данных в формате CSV.

Виртуальные таблицы могут быть созданы с помощью SQL-запросов или с использованием API расширений SQLite. Они предоставляют гибкость и возможность интеграции с различными источниками данных внутри SQLite.

9. SQLite. Индексы.

SQLite. Индексы:

Индексы в SQLite используются для ускорения операций поиска и сортировки данных в таблицах. Они представляют собой структуры данных, построенные на основе значений столбцов таблицы, которые обеспечивают быстрый доступ к данным, минимизируя необходимость сканирования всей таблицы.

В SQLite поддерживаются следующие типы индексов:

Индекс B-дерева (B-tree Index): Это наиболее распространенный тип индекса в SQLite. Он строится на основе структуры данных B-дерева и подходит для разных типов запросов, включая точные соответствия, диапазонные запросы и сортировку.

Индекс хеша (Hash Index): Используется для быстрого поиска точного соответствия значению столбца. Однако хеш-индексы доступны только для таблиц, созданных с опцией WITHOUT ROWID.

Индекс текста (Full-Text Index): Используется для полнотекстового поиска в текстовых данных. Он позволяет эффективно выполнять операции поиска по ключевым словам или фразам.

Индексы могут быть созданы на одном или нескольких столбцах таблицы с использованием оператора CREATE INDEX. Они могут значительно ускорить выполнение запросов, особенно при работе с большими объемами данных. Однако индексы требуют дополнительного пространства для хранения и могут замедлить операции вставки, обновления и удаления данных, так как при их выполнении индексы также должны быть обновлены. Поэтому необходимо тщательно выбирать, на какие столбцы создавать индексы, чтобы достичь оптимального баланса производительности.

10. SQLite. Представления.

SQLite. Представления:

Представления (Views) в SQLite являются виртуальными таблицами, которые определяются на основе запроса и могут быть использованы как обычные таблицы в других запросах. Представления предоставляют удобный способ абстрагирования сложных запросов или объединения нескольких таблиц в одну виртуальную таблицу.

Пример создания представления в SQLite:

```
sql
Copy code
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Здесь view_name - имя представления, column1, column2, ... - столбцы, которые будут отображаться в представлении, table_name - таблица, используемая в запросе, condition - условие фильтрации данных.

После создания представления, его можно использовать в других SQL-запросах, как если бы это была обычная таблица. При обращении к представлению будет выполнен запрос, определенный при создании представления, и возвращен результат.

11. SQLite. Триггеры.

SQLite. Триггеры:

Триггеры (Triggers) в SQLite - это специальные объекты, которые реагируют на определенные события (INSERT, UPDATE, DELETE) в таблицах базы данных и автоматически выполняют набор инструкций или запросов при возникновении этих событий. Триггеры позволяют реализовать дополнительную логику и контроль над данными.

Пример создания триггера в SQLite:

```
sql
Copy code
CREATE TRIGGER trigger_name
AFTER INSERT ON table_name
FOR EACH ROW
BEGIN
    -- Действия триггера
END;
```

Здесь trigger_name - имя триггера, table_name - таблица, на которую триггер будет назначен, AFTER INSERT - событие, на которое триггер реагирует, FOR EACH ROW - указывает, что триггер будет выполняться для каждой вставленной строки. В блоке BEGIN-END определяются инструкции или запросы, которые должны быть выполнены при возникновении события.

Триггеры могут быть созданы для различных событий (INSERT, UPDATE, DELETE) и могут выполнять различные действия, включая изменение данных, вставку данных в другие таблицы, генерацию ошибок и другое.

12. SQLite. Транзакции и блокировки.

SQLite. Транзакции и блокировки:

Транзакции в SQLite обеспечивают атомарность, согласованность, изолированность и долговечность операций с данными. Транзакция - это логический блок операций, который либо выполняется полностью, либо откатывается целиком, если возникла ошибка или была выполнена команда отката (ROLLBACK).

Пример использования транзакции в SQLite:

```
sql
Copy code
BEGIN TRANSACTION;
-- Выполнение операций
COMMIT;
```

Здесь BEGIN TRANSACTION - начало транзакции, COMMIT - фиксация изменений в базе данных. Если в ходе выполнения операций происходит ошибка, транзакция может быть отменена с помощью команды ROLLBACK.

SQLite также поддерживает блокировки, чтобы обеспечить согласованность и изоляцию при параллельном доступе к данным. Блокировки могут быть установлены на уровне таблиц или отдельных строк данных и могут быть эксклюзивными или разделяемыми.

Эксклюзивная блокировка (EXCLUSIVE): Позволяет только одной транзакции иметь доступ на запись к таблице или строке данных. Остальные транзакции будут заблокированы до завершения эксклюзивной транзакции.

Разделяемая блокировка (SHARED): Позволяет нескольким транзакциям иметь доступ на чтение к таблице или строке данных одновременно, но не позволяет ни одной транзакции иметь доступ на запись.

SQLite автоматически управляет блокировками и пытается разрешить конфликты между транзакциями с помощью механизма блокировки, известного как "оптимистическая блокировка". Если транзакции конфликтуют, одна из них будет откатываться для предотвращения нарушения целостности данных.

Базы данных