1. Основные понятия теории баз данных: база данных, система управления базами данных, основные требования к информации в БД, модели данных, логическая схема БД, основная терминология реляционных баз данных.

База данных – это совокупность взаимосвязанных данных.

Система управления базами данных (СУБД) - программная реализация технологии хранения, извлечения, обновления и обработки данных в базе данных.

Основные требования к информации БД:

1. Полезность
2. Полнота информации
3. Точность
4. Достоверность
5. Непротиворечивость
6. Актуальность

Модели данных:

1. Иерархическая модель: Данные организованы в виде дерева, где каждый элемент имеет одного родителя и ноль или более детей.
2. Сетевая модель: Данные организованы в виде графа, где элементы могут иметь несколько родителей и детей.
3. Реляционная модель: Данные представлены в виде таблиц, где отношения между ними устанавливаются с использованием ключей.

Логическая схема базы данных — это абстракция более высокого уровня, которая описывает организацию данных в базе данных с точки зрения таблиц, полей, связей и ограничений. Он не зависит от физической схемы и фокусируется на том, как данные логически организованы и связаны.

Терминология реляционных баз данных:

- **домен**: тип данных, то есть множество допустимых значений;
- **отношение (таблица)** – это множество связанных между собой данных
- **атрибут**: имя столбца таблицы (имя домена);
- **заголовок таблицы**: множество всех атрибутов;
- **кортеж**: элемент отношения или строка таблицы

2. Язык SQL.

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

Операторы SQL:

1. **DDL** - Data Definition Language - язык определения данных. Предназначены для создания, удаления и изменения объектов БД или сервера СУБД: **CREATE, DROP, ALTER**.
2. **DML** - Data Manipulation Language - язык манипулирования данными. Предназначены для работы с данными: **INSERT, DELETE, SELECT, UPDATE**.

3. **TCL** - Transaction Control Language - язык управления транзакциями. Предназначены для управления транзакциями: **BEGIN TRAN, SAVE TRAN, COMMIT TRAN, ROLLBACK TRAN**.
4. **DCL** - Data Control Language - язык управления данными. Предназначены для управления процессом авторизации: **GRANT, REVOKE, DENY.**

3. Системные базы данных в SQL Server: master, msdb, model, tempdb.

1. **master**: эта главная база данных сервера, в случае ее отсутствия или повреждения сервер не сможет работать. Она хранит все используемые логины пользователей сервера, их роли, различные конфигурационные настройки, имена и информацию о базах данных, которые хранятся на сервере, а также ряд другой информации.
2. **msdb:** хранит информацию о работе, выполняемой таким компонентом как планировщик SQL. Также она хранит информацию о бекапах баз данных.
3. **model** эта база данных представляет шаблон, на основе которого создаются другие базы данных.
4. **tempdb** эта база данных используется как хранилище для временных объектов. Она заново пересоздается при каждом запуске сервера.

4. Структура файла базы данных в SQL Server. Файловые группы.

БД представляет собой набор файлов операционной системы трех типов

1. **Первичный файл данных** (.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.

2. **Вторичные файлы данных** (.ndf). Необязательные определяемые пользователем файлы данных. Данные могут быть распределены на несколько дисков, в этом случае каждый файл записывается на отдельный диск.

3. **Файл журнала транзакций** (.log): Журнал содержит информацию для восстановления базы данных. Для каждой базы данных должен существовать хотя бы один файл журнала.

Основной единицей хранилища данных является **страница.** Размер страницы постоянен и составляет **8 Кб**. Каждая страница имеет заголовок размером **96 байт**, содержащий системную информацию, такую, как ID таблицы, которой принадлежит страница, а также указатели на следующую и предыдущую страницы для связанных страниц. Строки данных размещаются на странице сразу же после заголовка.

Физическая единица дискового пространства, используемая для выделения памяти для таблиц и индексов, называется **экстентом**. Размер экстента составляет **восемь 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.

Файловые группы – это поименованный набор файлов БД. Все файлы БД, кроме файлов журнала транзакций, распределены по файловым группам (Первичные, Вторичные). Файловая группа, называемая **первичной**, является **обязательной**. Для ее обозначения используются ключевые слова ON PRIMARY.

Во **вторичных** файловых группах могут располагаться только вторичные файлы.

В **первичной** файловой группе помимо обязательного первичного файла тоже могут быть расположены вторичные файлы.

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.

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

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

CREATE TABLE AUDITORIUM
(    AUDITORIUM   char(20)  primary key,
     AUDITORIUM_TYPE  char(10) foreign key references
                              AUDITORIUM_TYPE(AUDITORIUM_TYPE),
     AUDITORIUM_CAPACITY  int default 1
                     check  (AUDITORIUM_CAPACITY between 1 and 300),
     AUDITORIUM_NAME  varchar(50)
) on FG1;
```

5. Нормализация таблиц базы данных. Нормальные формы таблиц.

Нормализация данных - это процесс, в результате выполнения которого таблицы базы данных проверяются на наличие зависимостей между столбцами и, если необходимо, то исходная таблица разделяется на несколько таблиц.

Чтобы таблица соответствовала **1-й нормальной форме (1NF)**, необходимо, чтобы все значения ее полей были неделимыми и не вычисляемыми, а все записи – уникальными (не должно быть полностью совпадающих строк).

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 она находились во 2-й нормальной форме и все не ключевые поля в таблицах не зависели взаимно друг от друга.

6. Таблицы и их типы данных Microsoft SQL Server.

Таблицы (Tables): Основные объекты для хранения данных.

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

7. Подзапросы. Конструкции in, exists, all, any, some.

Подзапрос – это SELECT-запрос, который выполняется в рамках другого запроса.

В выражении SELECT мы можем вводить подзапросы четырьмя способами:

1. В условии в выражении WHERE
2. В условии в выражении HAVING
3. В качестве таблицы для выборки в выражении FROM
4. В качестве спецификации столбца в выражении SELECT

Коррелируемый подзапрос зависит от внешнего запроса и выполняется для каждой строки результирующего набора.
Независимый подзапрос выполняется один раз и его результат используется во внешнем запросе. Такой подзапрос не зависит от внешнего запроса.

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. (Противоположность - **NOT IN**)
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.
Конструкции **ANY** и **SOME** являются синонимами и используются для сравнения значения с любым значением, возвращаемым подзапросом. Запрос с ANY или SOME возвращает TRUE, если условие верно хотя бы для одного значения.

8. Группировка данных с использованием cube, rollup

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).

Оператор **ROLLUP** создает иерархические подытоги на основании порядка столбцов, указанных в выражении GROUP BY.

ROLLUP сначала создает стандартный набор группировок, как в обычном GROUP BY, по всем указанным столбцам. Это самый детализированный уровень.

Постепенное сокращение количества столбцов: Затем ROLLUP последовательно удаляет один столбец с правой стороны списка группировки, создавая подытоги. При

этом на каждой итерации группировка выполняется по оставшимся столбцам, а удаленный столбец получает значение NULL.

Общий итог: В конце процесса ROLLUP создает общий итог, где все столбцы получают значение NULL.

Если у нас есть столбцы A, B, C, то ROLLUP(A, B, C) создаст следующую иерархию группировок:

(A, B, C) – наиболее детализированный уровень
(A, B) – подытог по A и B
(A) – подытог по A
() – общий итог (все столбцы NULL)

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

Оператор **CUBE** создает подытоги для всех возможных комбинаций столбцов, указанных в выражении GROUP BY.

9. Операторы union (all), intersect, except.

UNION - Оператор UNION выполняет теоретико-множественную операцию объединения результирующих наборов SELECT-запросов, в котором строки не могут повторяться (объединение без копий).

UNION ALL - Если требуется механическое объединение строк, можно применить оператор (объединение с копиями)

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

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

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

10. Операторы соединения таблиц.

INNER JOIN - Возвращает строки, которые имеют совпадающие значения в обеих таблицах

LEFT OUTER JOIN - Все строки из левой таблицы и совпадающие строки из правой таблицы. Если совпадений нет, результаты из левой таблицы будут NULL.

RIGHT OUTER JOIN - Все строки из правой таблицы и совпадающие строки из левой таблицы. Если совпадений нет, результаты из левой таблицы будут NULL.

FULL OUTER JOIN - Возвращает все строки из обеих таблиц, соединяя их по указанному условию. Если в строке одной из таблиц нет соответствующей строки в другой таблице, то в результат вместо значений из несуществующей строки будут вставлены NULL значения.

CROSS JOIN - Возвращает декартово произведение всех строк из обеих таблиц. Каждая строка из первой таблицы соединяется со всеми строками из второй таблицы.

11. Язык T-SQL. Пакеты. Объявление переменных. Операторы присвоения. Оператор цикла while.

Transact-SQL — процедурное расширение языка SQL. 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 набор одной или нескольких SQL инструкций, которая обрабатывается сервером СУБД вместе.
Команда **GO** используется для разделения пакета на несколько пакетов.

Для объявления переменных, используемых в программах, предназначен оператор **DECLARE**. Для каждой переменной указывается имя и тип. Имя переменной должно начинаться с **символа @.**

```
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;
```

С помощью оператора **SET можно одной переменной** присвоить значение и

выполнять вычисления. Оператор **SELECT позволяет нескольким переменным** присвоить значения.

Областью видимости переменной являются все инструкции между ее объявлением и концом пакета или хранимой процедуры, где она объявлена. С помощью операторных скобок **BEGIN END** можно **объединять операторы в группы**. Переменные, объявленные внутри блока кода между BEGIN и END, видны только внутри блока. После END переменная больше не будет доступна.

Оператор WHILE содержит две составляющие: **логическое выражение** и **тело цикла**. Логическое выражение определяет условие выполнения тела цикла. Тело цикла содержит один или более операторов, которые выполняются в том случае и до тех пор, пока логическое выражение принимает значение «истина».

```
DECLARE @Counter INT = 1;

WHILE @Counter <= 10
BEGIN
	IF @Counter % 2 = 0
	BEGIN
	PRINT 'Even Number: ' + CAST(@Counter AS NVARCHAR(10));
	END

	SET @Counter = @Counter + 1;
END;
```

12. Язык T-SQL. Операторы print, if-else, case. Операторы begin-end, waitfor и return.

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

```
PRINT 'Hello, World!';
```

Оператор IF-ELSE используется для выполнения условных операций.

```
DECLARE @value INT;
SET @value = 10;

IF @value > 0
BEGIN
	PRINT 'Value is positive.';
END
ELSE
BEGIN
	PRINT 'Value is zero or negative.';
END
```

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.

Простая форма CASE:	Форма CASE с условием:

`DECLARE @Variable INT = 3;`
`DECLARE @Result NVARCHAR(50);`
 	
`SET @Result =`
`  CASE @Variable`
`  WHEN 1 THEN 'Один'`
`  WHEN 2 THEN 'Два'`
`  WHEN 3 THEN 'Три'`
`  ELSE 'Другое'`
`  END;`
`PRINT @Result;`
	`DECLARE @Variable INT = 3;`
	`DECLARE @Result NVARCHAR(50);`
	
	`SET @Result =`
	`  CASE`
	`  WHEN @Variable = 1 THEN 'Один'`
	`  WHEN @Variable = 2 THEN 'Два'`
	`  WHEN @Variable = 3 THEN 'Три'`
	`  ELSE 'Другое'`
	`  END;`
	`PRINT @Result;`

С помощью операторных скобок **BEGIN END** можно объединять операторы в группы (блоки).

```
BEGIN
      PRINT 'Start of the block';
      DECLARE @value INT;
      SET @value = 100;
      PRINT 'Value is set to ' + CAST(@value AS NVARCHAR(10));
END
```

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.

```
-- Delay the process by 20 seconds:
WAITFOR DELAY '00:00:20';
GO

-- Delay the process until 6:15 PM
WAITFOR TIME '18:15:00';
GO
```

Оператор **RETURN** служит для немедленного завершения работы пакета(триггера) и возврата значения в процедуре/функции.

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

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

В Transact-SQL для обработки ошибок используется конструкция **TRY-CATCH**, предоставляющая механизм обработки исключений.

TRY: В блоке TRY помещается код, который может вызвать ошибку.
CATCH: В блоке CATCH указывается код, который будет выполнен в случае возникновения ошибки в блоке TRY.

```
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
```

Функция **ERROR_MESSAGE()** возвращает текстовое описание ошибки.
ERROR_NUMBER(): Возвращает номер ошибки.
ERROR_SEVERITY(): Возвращает уровень серьезности ошибки.
ERROR_STATE(): Возвращает состояние ошибки.
ERROR_PROCEDURE(): Возвращает имя процедуры, в которой произошла ошибка.
ERROR_LINE(): Возвращает номер строки, где произошла ошибка.

Разработчик сценария на языке T-SQL может сам сгенерировать ошибку с помощью специальной **инструкции RAISERROR**. В простейшем случае, при вызове инструкции можно передать три параметра: **текстовое сообщение** об ошибке, уровень **серьезности ошибки** (0-25, Уровень 0-10 указывает на информационные сообщения, 11-16 — ошибки пользователя, 17-25 — ошибки программного обеспечения и системные ошибки) и **состояние ошибки** (0-255).

```
RAISERROR (message_string, severity, state);
```

14. Локальные и глобальные временные таблицы в SQL Server.

Основное отличие **временных** таблиц от постоянных в том, что они хранятся в системной БД **TEMPDB** и удаляются после окончания сеанса работы с базой. Как правило, временные таблицы создаются для временного хранения результатов SELECT-запросов.

Существует два вида временных таблиц: **локальные и глобальные.**

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

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

Глобальные временные таблицы имеют имена, начинающиеся с символа ##, доступны всем пользователям, подключенным к серверу. 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 её пользователем, то она будет удалена сразу после его отключения.

15. Курсоры в SQL Server. Объявление курсора. Общая схема работы с курсором: declare, open, fetch, close, deallocate. Типы курсоров.

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

Локальный курсор может применяться в рамках одного пакета и ресурсы, выделенные ему при объявлении, освобождаются сразу после завершения работы пакета.

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

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

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

Статический курсор не обновляется. Данные, полученные при открытии курсора, остаются неизменными в течение всего времени работы курсора. Требует меньше ресурсов, так как данные копируются во временную структуру при открытии курсора, и затем курсор работает с этой копией.

Динамический курсор обновляется. Если данные в базе данных изменяются, они будут отражены в результатах динамического курсора. Требует больше ресурсов, так как данные не копируются, и курсор работает непосредственно с базой данных. Из-за этого он может отражать изменения в реальном времени.

Курсор объявляется в операторе **DECLARE**. Курсор открывается с помощью оператора **OPEN**. 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.

После выполнения FETCH проверяется значение функции **@@fetch_status**, которая возвращает значение 0, если оператор FETCH выполнен успешно; -1, если достигнут конец результирующего набора и строка не считывается; -2, если выбранная строка отсутствует в БД. В зависимости от полученного результата цикл продолжается и считывается следующая строка, или цикл заканчивается.

Курсор закрывается оператором **CLOSE**. Если курсор глобальный, то он должен быть освобожден с использованием оператора **DEALLOCATE**.

По умолчанию для курсора установлен атрибут **SCROLL**, позволяющий применять оператор **FETCH** с дополнительными опциями позиционирования. Если добавить атрибут **NO SCROLL**, то дополнительные опции позиционирования нельзя использовать.

FETCH NEXT: Извлекает следующую строку данных из курсора.
FETCH PRIOR: Извлекает предыдущую строку данных из курсора.
FETCH FIRST: Извлекает первую строку данных из курсора.
FETCH LAST: Извлекает последнюю строку данных из курсора.
FETCH ABSOLUTE n: Извлекает строку данных на определенной позиции (по абсолютному номеру).
FETCH RELATIVE n: Извлекает строку данных относительно текущей позиции (по относительному номеру).

16. Хранимые процедуры и функции T-SQL. Создание хранимых процедур. Передача параметров. Входные и выходные параметры.

Хранимая процедура – это объект БД, представляющий собой поименованный код T-SQL. Хранимая процедура может быть создана с помощью CREATE, изменена с помощью ALTER и удалена с помощью оператора DROP. После имени процедуры должно **идти ключевое слово AS**.

```
CREATE PROCEDURE PrZakazy  -- создание процедуры
as
begin
   declare @k int = (select count (*) from Заказы);
   select * from Заказы;
   return @k;
end;
```

После названия процедуры идет **список входных параметров**, которые определяются также как и переменные - название начинается с символа @, а после названия идет тип переменной. Параметры можно отмечать как **необязательные**, присваивая им некоторое значение по умолчанию. При этом необязательные параметры лучше помещать в конце списка параметров процедуры.

```
CREATE PROCEDURE AddProduct
    @name NVARCHAR(20),
    @manufacturer NVARCHAR(20),
    @count INT,
    @price MONEY
AS
INSERT INTO Products(ProductName, Manufacturer, ProductCount, Price)
VALUES(@name, @manufacturer, @count, @price)
```

При вызове процедуры ей через запятую передаются значения. При этом значения передаются параметрам процедуры **по позиции**.

```
DECLARE @prodName NVARCHAR(20), @company NVARCHAR(20);
DECLARE @prodCount INT, @price MONEY
SET @prodName = 'Galaxy C7'
SET @company = 'Samsung'
SET @price = 22000
SET @prodCount = 5

EXEC AddProduct @prodName, @company, @prodCount, @price
```

Или **по имени**

```
DECLARE @prodName NVARCHAR(20), @company NVARCHAR(20)
SET @prodName = 'Honor 9'
SET @company = 'Huawei'

EXEC AddProduct @name = @prodName,
                @manufacturer=@company,
                @count = 3,
                @price = 18000
```

Выходные параметры позволяют возвратить из процедуры некоторый результат. Выходные параметры определяются с помощью **ключевого слова OUTPUT**. При вызове процедуры для выходных параметров передаются переменные с ключевым словом OUTPUT.

```
USE productsdb;
GO
CREATE PROCEDURE GetPriceStats
    @minPrice MONEY OUTPUT,
    @maxPrice MONEY OUTPUT
AS
SELECT @minPrice = MIN(Price),  @maxPrice = MAX(Price)
FROM Products
```

Кроме передачи результата выполнения через выходные параметры хранимая процедура также может возвращать какое-либо значение с помощью оператора **RETURN**. RETURN возвращает только целочисленные значения.

```
1  USE productsdb;
2
3  DECLARE @result MONEY
4
5  EXEC @result = GetAvgPrice
6  PRINT @result
```

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

17. Хранимые процедуры и функции T-SQL. Виды функций. Создание функций. Передача параметров.

Функция – это объект БД, представляющий собой поименованный код T-SQL. Для создания, удаления и изменения функций необходимо применять C**REATE, DROP и ALTER** соответственно. Возвращаемый тип функции указывается после **RETURNS**.

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

В функции не допускается применение DDL-операторов, DML-операторов, изменяющих БД (INSERT, DELETE, UPDATE), конструкций TRY/CATCH, а также использование транзакций.

Тело хранимой функции должно включать **оператор RETURN**, который указывает возвращаемое значение функции. Тело функции не должно включать никаких других операторов за исключением оператора RETURN. Хранимая функция может возвращать только одно значение и поддерживает только входные параметры.

Если функция возвращает единственное значение (число, строка, дата, время и пр.), то она называется ***скалярной***. Функция, возвращающая таблицу, называется ***табличной***. В зависимости от структуры кода, различают ***встроенные табличные*** функции и ***многооператорные табличные*** функции. Встроенные определяются одним оператором SELECT и обычно используются для простых запросов. Многооператорные могут содержать несколько операторов, переменные, условную логику и используются для более сложной логики обработки данных.

Пример табличной функции:

```
CREATE FUNCTION dbo.GetEmployeesByDepartment (@DepartmentID INT)
RETURNS TABLE
AS
RETURN
(
      SELECT EmployeeID, FirstName, LastName
      FROM Employees
      WHERE DepartmentID = @DepartmentID
);
```

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

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

18. Индексы. Назначение и применение индексов. Виды индексов. Применение различных видов индексов. Оптимизация запросов.

Индекс – это объект базы данных, позволяющий ускорить поиск в определенной таблице, так как при этом данные организуются в виде сбалансированного бинарного дерева поиска. Как и любой другой объект базы данных, индекс может быть создан с помощью оператора CREATE, модифицирован с помощью ALTER и удален с помощью оператора DROP. Для одной таблицы возможно построение нескольких индексов.

Индексы сохраняются в дополнительных структурах базы данных, называющихся страницами индексов. Для каждой индексируемой строки имеется элемент индекса, который сохраняется на странице индексов. Каждый элемент индекса состоит из ключа индекса и указателя. Ключом будет значение из поля для этой строки, а указатель будет указывать на местоположение самой строки в таблице.

Виды индексов:

1. **Кластеризованный индекс**. Обычно *кластеризованные* индексы создаются автоматически при создании таблицы если в ней присутствует первичный ключ (ограничение PRIMARY KEY). Кроме этого, каждый кластеризованный индекс однозначен по умолчанию, т.е. в столбце, для которого определен кластеризованный индекс, каждое значение данных может встречаться только один раз. Определяет физический порядок строк в таблице. В таблице может быть только один кластеризованный индекс. Если кластеризованный индекс создается для столбца, содержащего повторяющиеся значения, СУБД принудительно добавляет четырехбайтовый идентификатор к строкам, содержащим дубликаты значений.

```
CREATE CLUSTERED INDEX idx_EmployeeID ON Employees (EmployeeID);
```

2. **Некластеризованный индекс.** Не влияют на физический порядок. создают отдельную структуру, которая указывает на строки в таблице. В таблице может быть несколько некластеризованных индексов.

```
CREATE NONCLUSTERED INDEX idx_LastName ON Employees (LastName);
```

3. **Уникальный индекс** гарантирует, что значения в индексируемом столбце или комбинации столбцов будут уникальными.

```
CREATE UNIQUE INDEX idx_UniqueEmail ON Employees (Email);
```

4. **Фильтрованный** индекс это некластеризованные индексы, созданные на подмножестве строк таблицы. Они полезны, когда нужно индексировать только определенные данные.

```
CREATE  index #EX_WHERE on #EX(TKEY) where (TKEY>=15000 and TKEY < 20000);
```

5. **Индекс покрытия.** Когда индекс включает все столбцы, необходимые для выполнения запроса, он называется покрывающим индексом. Например, мы создали некластеризованный индекс на 3 столбцах, и мы запрашиваем только эти столбцы, тогда этот некластеризованный индекс называется покрывающим индексом.

Индексы могут быть **простыми** или **составными**, в зависимости от количества столбцов, которые они охватывают. Простые индексы включают один столбец, в то время как составные индексы включают два и более столбца.

19. План запроса. Этапы обработки select запроса. Понятие стоимости запроса. Понятия селективности и плотности. Индексы. Реорганизация, перестроение, включение и отключение индексов.

План запроса представляет собой алгоритм выполнения SQL-запроса.

При отправке sql-запроса серверный процесс:

1. Синтаксический разбор: проверка на соответствие правилам языка, Если запрос содержит синтаксические ошибки, он отвергается, и пользователь получает сообщение об ошибке.
2. Разрешение имён: Проверка наличия и корректности используемых объектов (таблиц, столбцов и т.д.). Если указанные объекты не существуют или к ним нет доступа, возвращается ошибка.
3. Оптимизация: Специальная компонента сервера – **оптимизатор**. Основная задача – построение плана запрос. Для каждого шага вычисляется стоимость – величина, пропорциональная продолжительности выполнения шаг. Суммарная стоимость шагов плана составляет стоимость всего запроса. Задача – минимизация общей стоимости запроса.
4. Компиляция: Преобразование выбранного плана выполнения в конкретные инструкции, которые система управления базами данных (СУБД) сможет выполнить. Этот этап включает создание внутреннего представления плана выполнения.
5. Выполнение: СУБД выполняет последовательность операций, необходимых

для получения результата запроса. Это может включать чтение данных из таблиц, применение фильтров, выполнение соединений и агрегаций и т.д.

План запроса состоит из шагов. Для каждого шага вычисляется стоимость. Суммарная стоимость шагов плана составляет стоимость всего запроса. **Стоимость запроса** - это оценка ресурсов (времени, памяти, процессора и т.д.), необходимых для выполнения запроса.

Селективность индекса - это мера того, насколько индекс сужает количество строк, которые необходимо проверить при выполнении запроса. Чем более селективный индекс, тем меньше строк будет просмотрено, что обычно приводит к более быстрому выполнению запроса.

Плотность запроса – количество возвращаемых строк запроса.

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

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

Операция *перестройки* (REBUILD) затрагивает все узлы дерева, поэтому после ее выполнения степень фрагментации равна нулю.

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

Отключение индекса:
ALTER INDEX index_name ON table_name DISABLE;

Включение индекса:
ALTER INDEX index_name ON table_name REBUILD;

20. Триггеры. Типы триггеров. Создание after-триггера.

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

Существует несколько типов триггеров:

1. After-триггеры: Эти триггеры выполняются после выполнения операции,

такой как вставка, обновление или удаление. Только для таблиц. Триггеры уровня оператора.

2. Instead of-триггеры: Эти триггеры могут быть определены для выполнения вместо стандартного действия, такого как вставка, обновление или удаление. Для таблицы и представлений. Триггеры уровня оператора. Выполняется после создания таблиц inserted и deleted. Выполняется перед выполнением проверки ограничений целостности или каких-либо других действий.

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

Специальные виртуальные таблицы:

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

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

DDL-триггеры уровня сервера

```
CREATE TRIGGER trigger_ConnectionLimit
    ON ALL SERVER WITH EXECUTE AS 'loginTest'
```

DDL-триггеры уровня базы данных

```
CREATE TRIGGER trigger_PreventDrop
    ON DATABASE FOR DROP_TRIGGER
```

DDL триггеры работают только с событием AFTER.

Если для таблицы или представления созданы INSTEAD OF и AFTER-триггеры, реагирующие на одно и то же событие, то выполняется только INSTEAD OF- триггер.

Можно создавать триггеры на набор событий:

```
create trigger TRIG_Tov  on Товары after INSERT, DELETE, UPDATE
as declare @a1 varchar(20), @a2 real, @a3 int, @in varchar(300);
declare @ins int = (select count(*) from inserted),
        @del int = (select count(*) from deleted);
if  @ins > 0 and  @del = 0
begin
    print 'Событие: INSERT';
    set @a1 = (select [Наименование] from INSERTED);
    set @a2 = (select [Цена] from INSERTED);
    set @a3 = (select [Количество] from INSERTED);
    set @in = @a1+' '+cast(@a2 as varchar(20))+' '+cast(@a3 as varchar(20));
    insert into TR_Tov(ST, TRN, C)  values('INS', 'TRIG_Tov', @in);
end;
else
if @ins = 0 and  @del > 0
begin
```

21. Триггеры. Создание и назначение instead of-триггеров. Таблицы inserted, deleted.

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).

inserted: Это виртуальная таблица, содержащая строки, которые будут вставлены или обновлены в результате выполнения операции. В случае триггера на вставку, в ней будут находиться только вставляемые строки. В случае триггера на обновление, в ней будут находиться новые значения обновляемых строк.

deleted: Это виртуальная таблица, содержащая строки, которые будут удалены или обновлены в результате выполнения операции. В случае триггера на удаление, в ней будут находиться удаляемые строки. В случае триггера на обновление, в ней будут находиться старые значения обновляемых строк.

```
create trigger Tov_INSTEAD_OF
        on Товары instead of DELETE
            as raiserror (N'Удаление запрещено', 10, 1);
return;
```

22. Транзакции. Свойства ACID. Транзакции. Уровни изолированности транзакций. Функция trancount.

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

1. **Неявная транзакция** — задает любую отдельную инструкцию INSERT, UPDATE или DELETE как единицу транзакции.
2. **Явная транзакция** — группа инструкций, начало и конец которой обозначаются инструкциями:

- BEGIN TRANSACTION
- COMMIT
- ROLLBACK

Любая реляционная база данных должна удовлетворять тесту ACID: должны быть гарантированы **атомарность** (Выполняются или все изменения данных в транзакции или ни одна), **согласованность** (Выполняемые транзакцией трансформации данных переводят базу данных из одного согласованного состояния в другое), **изолированность** (Незаконченная должна быть невидима для остального мира) и **долговечность** (Транзакцию после фиксации нельзя отменить, кроме как другой транзакцией).

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

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

1. **READ UNCOMMITTED.** Не изолирует операции чтения других транзакций. Транзакция не задает и не признает блокировок.

Проблемы:

- Грязное чтение (когда одна транзакция читает данные, которые были изменены другой транзакцией, но еще не зафиксированы (не завершена))

- Неповторяемое чтение (когда одна транзакция выполняет несколько чтений одного и того же набора данных в разные моменты времени, и при этом получает разные результаты из-за изменений, сделанных другой транзакцией между этими чтениями)

- Фантомное чтение (когда одна транзакция выполняет несколько чтений с использованием одного и того же условия, и получает **разные наборы данных** из-за вставки, удаления или обновления строк другой транзакцией между этими чтениям.)

2. **READ COMMITTED.** Транзакция выполняет проверку только на наличие монопольной блокировки для данной строки. Является уровнем изоляции по умолчанию

Проблемы:

- Неповторяемое чтение
- Фантомное чтение

3. 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

4. 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.

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**.

```
BEGIN TRANSACTION;

INSERT INTO AUDITORIUM VALUES ('127-1', 'ЛК', 60, '127-1');
SAVE TRANSACTION A;

INSERT INTO AUDITORIUM VALUES ('128-1', 'ЛК-К', 60, '128-1');
SAVE TRANSACTION B;

INSERT INTO AUDITORIUM VALUES ('129-1', 'ЛК', 60, '129-1');
ROLLBACK TRANSACTION B;

INSERT INTO AUDITORIUM VALUES ('130-1', 'ЛК-К', 60, '130-1');
ROLLBACK TRANSACTION A;

COMMIT TRANSACTION;

SELECT * FROM AUDITORIUM
WHERE AUDITORIUM IN('127-1', '128-1', '129-1', '130-1');
```

Результаты | Сообщения

AUDITORIUM	AUDITORIUM_TYPE	AUDITORIUM_CAPACITY	AUDITORIUM_NAME
127-1	ЛК	60	127-1

Оператор **SAVE TRANSACTION** используется для установки точки сохранения в пределах транзакции. Без SAVE TRANSACTION не существует способа управлять вложенными транзакциями. Он помогает установить точку сохранения в транзакции, к которой позднее может быть выполнен откат (если потребуется), используя ту же точку сохранения.

Транзакция завершается либо с помощью фиксации, либо отката. Фиксация может быть сделана с помощью **COMMIT TRANSACTION** (или COMMIT TRAN, или просто COMMIT). Но если откат должен быть сделан к точке сохранения, то она должна быть указана наряду с командой **ROLLBACK**. Например, ROLLBACK TRANSACTION <точка сохранения>.

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

Операторы TCL:

•BEGIN TRAN

•SAVE TRAN

•COMMIT TRAN

•ROLLBACK TRAN

1. Привилегии на уровне сервера. Привилегии, которые предоставляют доступ к операциям на уровне сервера, например, создание новых баз данных или управление логинами.

- Примеры привилегий:
  - `CREATE LOGIN`
  - `ALTER ANY LOGIN`
  - `SHUTDOWN`

- `VIEW SERVER STATE`

2. Привилегии на уровне базы данных. Привилегии, которые позволяют пользователям выполнять действия внутри конкретной базы данных, например, чтение данных, модификация таблиц и выполнение хранимых процедур.

- Примеры привилегий:
  - `SELECT`
  - `INSERT`
  - `UPDATE`
  - `DELETE`
  - `EXECUTE`
  - `ALTER`
  - `CONTROL`

Роли – это наборы привилегий, которые можно назначить пользователям или другим ролям. MSSQL имеет предопределенные роли и возможность создавать пользовательские роли.

1. Предопределенные роли:
   - На уровне сервера:
     - `sysadmin`: Полные права на сервер.
     - `serveradmin`: Управление настройками сервера.
     - `securityadmin`: Управление логинами и правами.
   - На уровне базы данных:
     - `db_owner`: Полные права на базу данных.
     - `db_datareader`: Чтение всех данных в базе данных.
     - `db_datawriter`: Изменение всех данных в базе данных.

2. Пользовательские роли - Администратор может создать роли с набором привилегий, соответствующих конкретным требованиям.

```
CREATE ROLE my_custom_role;
GRANT SELECT, INSERT ON my_table TO my_custom_role;
```

Привилегии могут назначаться напрямую пользователям или через роли. Примеры назначения привилегий:

```
GRANT SELECT ON my_table TO my_user;
```

```
EXEC sp_addrolemember 'db_datareader', 'my_user';
```

24. Дистрибутивы СУБД Oracle. Установка СУБД Oracle 12c на Windows. Global Database Name и SID.

Дистрибутивы:
Express Edition: бесплатная редакция
Standard Edition One: начальная редакция для дома и малого бизнеса
Standard Edition: для малого и среднего бизнеса
Enterprise Edition: решения промышленного уровня

Выберите тип установки:
Создать и настроить базу данных: Выберите, если вы хотите создать новую базу данных и настроить её.
Установить программное обеспечение базы данных: Выберите, если вы хотите только установить программное обеспечение без создания базы данных.
Обновить базу данных: Выберите, если вы хотите обновить существующую базу данных.
Классификация установки:
Desktop Class: Для персональных компьютеров или небольших серверов.
Server Class: Для более мощных серверов и корпоративных сред.
Режим установки:
Single Instance: Одна бд на одном сервере.
Real Application Clusters (RAC): Несколько серверов работают совместно и предоставляют доступ к одной базе данных.
RAC One Node: Развертывания базы данных на одном узле с возможностью перемещения этой базы данных на другой узел.
Настройки установки:
Typical: Для типичной установки с минимальными настройками.
Advanced: Для продвинутой установки с детальной настройкой параметров.
Создание пользователя Windows: Укажите, хотите ли вы создать нового пользователя Windows для управления Oracle. Введите имя пользователя и пароль, если требуется.
Путь установки:
Oracle base: Это корневая директория, где будут размещены все файлы Oracle.
Software location: Директория, в которой будет установлено программное обеспечение Oracle. Это подкаталог внутри Oracle base, обычно называемый product или что-то подобное.
Database file location: Директория, где будут храниться файлы базы данных (файлы данных, журналы восстановления, контрольные файлы и т.д.). Обычно это отдельный путь, но он может находиться внутри Oracle base.
Global database name: Глобальное имя базы данных включает имя базы данных и домен, например, orcl.example.com. Оно уникально идентифицирует базу данных в сети.
Administrative password: Пароль для административных учетных записей базы данных, таких как SYS и SYSTEM.
Pluggable database name: Имя подключаемой базы данных (PDB), которую вы создаете в контейнерной базе данных (CDB).

Global Database Name (глобальное имя базы данных) представляет собой полное имя базы данных, включающее доменное имя. Обычно оно состоит из двух частей:
Имя базы данных: Локальное имя базы данных.

Доменное имя: Доменная часть, идентифицирующая базу данных в сети.
Пример: orcl.example.com, где orcl — имя базы данных, а example.com — доменное имя.

SID (System Identifier, системный идентификатор) — уникальный идентификатор для конкретного экземпляра базы данных Oracle. SID используется для определения экземпляра базы данных в многопользовательской среде.
Пример: orcl

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

Системные пользователи:
SYS – предопределенный привилегированный пользователь ранга администратора базы данных, который является владельцем ключевых ресурсов БД Oracle
SYSTEM – предопределенный привилегированный пользователь, которому принадлежат ключевые ресурсы БД Oracle

Специальные привилегии:
SYSDBA и SYSOPER - специальные привилегии администратора, которые позволяют выполнять базовые задачи администрирования: запуск или остановка экземпляра БД; создание, удаление, открытие или монтирования базы и др.

DBA – предопределенная роль, которая автоматически создается для каждой базы данных Oracle и содержит все системные привилегии, кроме SYSDBA и SYSOPER

26. Понятия базы данных и экземпляра базы данных.

База данных — это физическое хранилище информации, а **экземпляр** — это программное обеспечение, которое работает на сервере и предоставляет доступ к информации, содержащейся в базе данных Oracle. Экземпляр исполняется на конкретном сервере либо компьютере, в то самое время как база данных хранится на дисках, подключённых к этому серверу.

При этом база данных Oracle является **физической сущностью**, состоящей из файлов, которые хранятся на дисках. В то же самое время, экземпляр – это **сущность логическая**, состоящая из структур в оперативной памяти и процессов, которые работают на сервере. Экземпляр может являться частью только одной базы данных. При этом с одной базой данных бывает ассоциировано несколько экземпляров.

27. Запуск и останов экземпляра базы данных Oracle.

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

Запуск

STARTUP NOMOUNT:

Запуск экземпляра подразумевает выполнение следующих задач:

1) Поиск файла параметров (хранит параметры экземпляра)
2) Выделение SGA;
3) Запуск фоновых процессов

STARTUP MOUNT:

Ассоциация базы данных с предварительно запущенным экземпляром:

Определение местоположения управляющих файлов (файлы, содержащие имена (местоположение) основных физических файлов), которые указаны в файле параметров:

STARTUP OPEN: Открытие базы данных подразумевает выполнение следующих задач:

• Открытие оперативных файлов данных;
• Открытие оперативных журнальных файлов.

Остановка

SHUTDOWN NORMAL: Запрещено создавать новые сессии. Ожидается завершение работы всех пользователей. 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. Никаких восстановительных работ при следующем старте не проводится.

SHUTDOWN IMMEDIATE: Запрещено создавать новые сессии. Запрещено запускать новые транзакции. Все незафиксированные транзакции откатываются. Никаких восстановительных работ при следующем старте не проводится.

SHUTDOWN ABORT: Все действия прекращаются. Все транзакции не фиксируются и не откатываются. Пользователей отсоединяют от БД. При следующем старте будет выполнено возможное восстановление.

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

Словарь Oracle - набор таблиц и связанных с ними представлений, который предоставляет возможность отследить внутреннюю структуру базы данных и деятельность СУБД Oracle

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.

Основные группы представлений:

USER	Объекты, принадлежащие пользователю
ALL	Объекты, к которым пользователь имеет доступ
DBA	Все объекты базы данных (для администратора БД)
V$	Текущее состояние и производительность экземпляра базы данных

29. Мультиарендная архитектура Oracle Multitenant.

Oracle Multitenant - технология, позволяющая запустить несколько независимых баз данных в рамках одного экземпляра. Каждая база данных имеет свой набор табличных пространств и набор схем, но при этом у них общая SGA и один набор серверных процессов. Базы данных изолированы, друг о друге ничего не знают, не конфликтуют между собой.

Основные компоненты архитектуры:

Container Database (CDB) — контейнерная база данных, содержащая несколько подключаемых баз данных (PDB). CDB включает общую SGA (System Global Area) и набор серверных процессов, которые разделяются между всеми контейнерами.

Pluggable Database (PDB) — отдельная база данных, которая может быть создана в CDB. PDB имеют свои табличные пространства и наборы схем, изолированные друг от друга. Однако у них общая SGA и один набор серверных процессов с другими PDB в той же CDB.

PDB$SEED — шаблон для создания новых, пустых PDB. Это read-only контейнер, который не может быть отключен.

В архитектуре Oracle Multitenant одна CDB может вмещать до **252 PDB**. PDB можно перемещать между разными CDB. Общая часть словаря базы данных хранится в CDB, а специфичная информация словаря каждой PDB хранится в самой PDB. Мультиарендная БД имеет один набор журнальных файлов (redo logs) и один набор управляющих файлов, общих для всех PDB в контейнерной БД.

Общий пользователь:

- создаётся CDB$ROOT и виден в каждой PDB
- может подключаться к тем PDB, где у него есть привилегия “Create Session”
- Может владеть объектами в root и в PDB
- может администрировать всю CDB-базу данных, если у него есть привилегия DBA

▪ обычно имеет имя, начинающееся на C##

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

Файлы параметров используются для конфигурирования действий Oracle предже всего при старте. Для того, чтобы запустить экземпляр базы данных, Oracle должен прочесть файл параметров и определить, какие параметры инициализации установлены для этого экземпляра. В файле параметров содержатся многочисленные параметры и их установленные значения. Oracle считывает файл параметров при запуске базы данных.

SPFILE - постоянно находящийся на сервере бинарный файл, который может быть изменен только с помощью команды "**ALTER SYSTEM SET**".
PFILE - статичный, пользовательский текстовый файл, который редактируется стандартными текстовыми редакторами

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

Изменение:

```
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 - файлы, содержащие имена (местоположение) основных физических файлов базы данных и некоторых параметров. База данных Oracle может иметь один или несколько управляющих файлов. Если имеется несколько управляющих файлов, все они должны быть абсолютно идентичными, можно разместить их на разных дисках.

Местоположение управляющего файла находится в файле параметров. По умолчанию создается две копии.

Если надо **изменить управляющий файл**, то следует создать сценарий, откорректировать его и выполнить:
Остановить Oracle (shutdown transactional или immediate);
Скопировать один из управляющих файлов;
Изменить параметр CONTROL_FILES в файле параметров;
Стартовать Oracle (startup open).

Уменьшение количества управляющих файлов:
Определите новое количество управляющих файлов с помощью параметра CONTROL_FILES в файле параметров базы данных (init.ora или spfile.ora). Этот параметр должен содержать полные пути к управляющим файлам.

```
ALTER SYSTEM SET CONTROL_FILES = '/path/to/new/controlfile1.ctl', '/path/to/new/controlfile2.ctl' SCOPE=SPFILE;
```

Файлы паролей: Содержат хеши паролей пользователей базы данных. Используются для аутентификации пользователей при подключении к базе данных. Название файла PWDORCLE_SID.ora.

```
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;
```

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

Все данные в базе данных Oracle сохраняются в **файлах данных**. Все таблицы, индексы, триггеры, последовательности, программы на PL/SQL, представления - все это находится в файлах данных. И хотя эти и другие объекты базы данных логически содержатся в табличных пространствах, в действительности они сохраняются в файлах на жестком диске компьютера.

В каждой базе данных Oracle имеется по крайней мере один файл данных (но обычно их бывает больше). Если вы создаете в Oracle таблицу и заполняете ее строками, Oracle помещает эту таблицу и строки в файл данных. Каждый файл данных может быть связан только с одной базой данных.

```
select * from dba_data_files;
```

Журналы повторного выполнения - дисковые ресурсы, в которых фиксируются изменения вносимых пользователями в базу данных.

Назначение оперативного файла журнала повтора заключается в сохранении информации об изменениях в базе данных таким образом, чтобы позже их можно было повторить.

Каждая база данных должна иметь не менее двух оперативных файлов 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**.

Группы журналов повтора позволяют Oracle обеспечить отказоустойчивость и производительность, так как запись изменений происходит **параллельно в несколько файлов**.

Принцип работы заключается в том, что при каждом изменении данных в базе данных Oracle сначала записывается изменение в журнал повтора, а затем только применяется к самим данным. Это обеспечивает согласованность данных и позволяет восстановить базу данных до любого момента времени, используя журналы повтора в случае сбоя или отказа системы.

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

Архивы: Содержат скопированные журналы повтора, которые могут использоваться для восстановления базы данных до определенного момента в прошлом, поддерживая точку восстановления.

alter database archivelog/noarchivelog

Когда вы включили архивирование журналов повтора (redo log archiving) в Oracle, база данных начала автоматически архивировать (сохранять) копии журналов повтора после их переключения. Каждый раз, когда вы переключаетесь на новую группу журнальных файлов (redo log group), текущий активный журнал повтора (redo log) закрывается и архивируется, а затем база данных переключается на следующий журнал повтора для записи.

32. 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.

Сегмент (segment) — область на диске, выделяемая под объекты. Сегменты типизируются в зависимости от типа данных, хранящихся в них – сегменты таблиц, сегменты индексов, сегменты кластеров и т.д.(всего 10 типов).

Экстенты — это непрерывный фрагмент дисковой памяти. 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). Размер 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

33. Абстрактная модель Oracle. Физическая структура внешней памяти.

Физическая структура состоит из :

Файлы данных (31 вопрос) : содержат фактические данные базы данных, такие как таблицы и индексы. Эти файлы являются основными контейнерами для данных и находятся в табличных пространствах.

Файлы параметров (30 вопрос) : содержат параметры конфигурации базы данных. Существуют два типа файлов параметров:

- PFILE
- SPFILE

Управляющие файлы (30 вопрос) : Хранят информацию о структуре базы данных, таких как местоположение файлов данных и журналов, а также состояние архивирования.

Файлы сообщений (30 вопрос) : содержат диагностическую информацию и сообщения об ошибках:

- Файлы трассировки (Trace Files): содержат подробную информацию для диагностики и отладки базы данных.
- Журнал предупреждений (Alert Log): регистрирует важные события, ошибки и предупреждения в базе данных.

Журналы повтора (31 вопрос) : записывают все изменения, внесенные в базу данных. Они обеспечивают целостность данных и используются для восстановления базы данных в случае сбоя.

- Online Redo Log Files: текущие журналы.
- Archived Redo Log Files: архивированные журналы для восстановления.

SGA (34 вопрос) - это группа общих структур памяти, используемых экземпляром базы данных Oracle. Она хранит данные и информацию о состоянии, которые необходимы для работы базы данных.

34. Абстрактная модель Oracle. Структура SGA.

SGA означает "System Global Area", что переводится как "Системная Глобальная Область", которую Oracle использует для хранения данных и управляющей информации одного конкретного экземпляра Oracle. SGA размещается в памяти при запуске экземпляра Oracle и освобождает память при останове. Каждый запущенный экземпляр Oracle имеет свою собственную SGA.

Пулы SGA:

1. **Shared Pool** (Разделяемый пул): Библиотечный кэш (хранит информацию о последних выполненных операторах SQL и PL/SQL.), кэш словаря данных (подмножество данных из словаря данных).Изменяется размер ALTER SYSTEM. SHARED_POOL_SIZE

2. **Large Pool** (Большой пул): Область памяти SGA, применяемая для хранения больших фрагментов памяти. Изменяется размер ALTER SYSTEM

3. **Фиксированная область**: хранит переменные, указывающие на другие области памяти, значения параметров. Размером управлять нельзя

4. **Java Pool** (Java-пул): Java-пул предназначен для работы Java-машины. Изменяется размер ALTER SYSTEM

5. **Буферный пул**: область SGA, которая содержит образы блоков, считанные из файлов данных или созданные динамически, чтобы реализовать модель согласованного чтения. Изменяется размер ALTER SYSTEM

Когда пользователь или приложение запрашивает данные из базы данных, Oracle сначала проверяет, есть ли эти данные в буферном кэше. Если данные уже находятся в кэше, они могут быть немедленно предоставлены пользователю без необходимости обращения к диску. Если данные отсутствуют в кэше, Oracle должен выполнить операцию чтения с диска и загрузить данные в кэш перед тем, как предоставить их пользователю.

Каждый блок данных в кэше имеет свой счётчик использования, который увеличивается каждый раз, когда этот блок читается или изменяется. Это позволяет Oracle определить, какие блоки наиболее активно используются, и оптимизировать кэширование данных в памяти.

- KEEP: Постоянно хранит блоки данных в памяти. У вас могут быть маленькие таблицы, к которым выполняются частые обращения, и для предотвращения их удаления из буферного кэша им можно назначить постоянный буферный пул при создании таблицы. DB_KEEP_CACHE_SIZE

- DEFAULT: Содержит все данные и объекты, которые не назначены в постоянный и повторно используемый буферные пулы DB_CACHE_SIZE

- RECYCLE: Удаляет данные из кэша немедленно после использования. DB_RECYCLE_CACHE_SIZE

Алгоритм LRU (least recently used)– первыми вытесняются блоки с наименьшим значением счетчика. Когда требуется место для нового блока данных, система вытесняет из кэша блок, который дольше всего не использовался.

6. **Буфер журнала повторного выполнения**. Буфер журнала повторного выполнения предназначен для временного циклического хранения данных журнала повтора. ??? Log_Buffer

7. **Streams Pool** (Пул потоков): Пул, предназначенный для работы с потоками данных в технологии Oracle Streams, которая используется для репликации данных и анализа данных изменений в реальном времени. Этот пул хранит информацию о потоках данных и обрабатывает их. ALTER SYSTEM STREAMS_POOL_SIZE

8. **Null Pool** (Неопределенный пул): В этот пул включается память, выделенная для буферов блоков, буферов журнала повторного выполнения и "фиксированной области SGA", которая является частью SGA, но не специфицирована как отдельный пул.

DB_CACHE_SIZE, DB_KEEP_CACHE_SIZE, DB_RECYCLE_CACHE_SIZE, SHARED_POOL_SIZE

35. Абстрактная модель Oracle. Серверные процессы Oracle.

Серверные процессы – процессы, выполняющиеся на основании клиентского запроса и обслуживает пользовательский процесс. Каждый пользователь, подключенный к базе данных, имеет свой отдельный серверный процесс, существующий на протяжении существования сеанса.
Пользовательский процесс. Эти процессы отвечают за выполнение приложения, подключающего пользователя к экземпляру базы данных.

Количество пользовательских процессов на каждый серверный процесс зависит от типа конфигурации сервера.

Так, например, когда пользователь посылает запрос на выборку данных, серверный процесс ,созданный для обслуживания пользовательского приложения, проверяет синтаксис кода и выполняет код SQL. Затем он читает данные из файлов в блоки памяти. (Если другой пользователь захочет прочесть те же данные, то его пользовательский процесс прочтет их не с диска, а из памяти Oracle, где данные обычно находятся некоторое время.) И,наконец, серверный процесс вернет запрошенные данные пользовательскому процессу.

SELECT * FROM V$PROCESS;

36. Абстрактная модель Oracle. Фоновые процессы Oracle.

Фоновые процессы – запускаются вместе с базой данных и выполняют разнообразные задачи обслуживания

LREG - Listener Registration Process – периодическая регистрация сервисов в процессе Listener

Listener — это серверный процесс в Oracle Database, который принимает входящие сетевые подключения к базе данных и направляет их к соответствующим экземплярам базы данных.

Экземпляр может иметь несколько **точек подключения**. Точки подключения называются **сервисами** и имеют символические имена.

DBWn - Database Writer Process: (n=0,…,9, a,…,z; BWm, m=36,…,99) – фоновый процесс записывающий по LRU измененные блоки (грязные блоки) в файлы базы данных.

LGWR - Log Writer process - При изменениях в таблице базы данных (INSERT, UPDATE, DELETE), Oracle записывает зафиксированные и незафиксированные изменения в redo буфер в памяти. После LGWR записывает эти изменения в redolog файлы.

CKPT – checkpoint Process:

1. Запись содержимого буферов redolog в redolog файлы
2. Внесение записи контрольной точки в redolog файл
3. Запись буферного кэша в файл данных
4. Обновление заголовков файла данных и control файлов после завершения контрольной точки.

ARCn - Archiver Process – копирует файлы журнала повтора после переключения группы журналов

PMON - Process monitor – отвечает за очистку после ненормального закрытия подключений, инициирует откат незафиксированных транзакций, снятие блокировок, и освобождение ресурсов SGA, следит за работой других фоновых процессов

SMON – System monitor - Восстановление незавершенных транзакций, Очистка временных сегментов данных, Очистка временных табличных пространств

RECO - Recovery Process – разрешение проблем связанных с распределенными транзакциями

FBDA - Flashback Data Archiever – архивирование ретроспективных данных

select * from v$bgprocess;

37. Процесс-слушатель Oracle и его основные параметры.

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

lsnrctl - утилита, управляющая процессом Listener.

1. start: Запускает слушателя.
2. stop: Останавливает слушателя.
3. status: Отображает текущий статус слушателя (запущен, остановлен).
4. reload: Перезагружает конфигурацию слушателя без его остановки.
5. save_config: Сохраняет текущую конфигурацию слушателя.
6. trace: Включает или отключает трассировку для слушателя.

7. set: Устанавливает параметры конфигурации слушателя (например, set password).
8. unset: Сбрасывает параметры конфигурации слушателя.
9. services: Отображает список сервисов, зарегистрированных в слушателе.
10. version: Отображает версию Oracle Net Listener.
11. reload_admin: Перезагружает файл (listener.ora).
12. exit: Выходит из утилиты lsnrctl.

Файл listener.ora содержит информацию о конфигурации Listener Oracle. Поскольку служба слушателя действует только на сервере, клиентские компьютеры не содержат никакого файла listener.ora.

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = CLRExtProc)
      (ORACLE_HOME = C:\WINDOWS.X64_193000_db_home)
      (PROGRAM = extproc)
(ENVS = "EXTPROC_DLLS=ONLY:C:\WINDOWS.X64_193000_db_home\bin\oraclr19.dll")
    )
  )

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = oracledb)(PORT = 1521))
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
  )
```

38. Сетевые настройки Oracle. Установление соединения по сети.

Сетевое программное обеспечение СУБД Oracle именуется **Oracle Net Services**. Oracle Net Services обеспечивает прозрачное соединение с экземпляром сервера Oracle. Ключевую роль при конфигурации сетевого программного обеспечения играют два файла конфигурации: SQLNET.ORA и TNSNAMES.ORA. Эти файлы содержат всю необходимую информацию для соединения с сервером Oracle. Файлы являются текстовыми и создаются при инсталляции сервера и клиента.

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

1. Службы Oracle Net
2. Oracle Net Listener
3. Oracle Net Configuration Assistant
4. Oracle Net Manager
5. Oracle Connection Manager

Файл listener.ora содержит информацию о конфигурации Listener Oracle. Поскольку служба слушателя действует только на сервере, клиентские компьютеры не содержат никакого файла listener.ora.

Файл tnsnames.ora в Oracle служит для конфигурации сетевых соединений с базами данных. Он содержит имена (TNS - Transparent Network Substrate) и соответствующие сетевые адреса для доступа к базам данных.

sqlnet.ora – конфигурации сетевых параметров.

Basic

Вы можете подключиться к локальному или удаленному экземпляру базы данных Oracle, используя базовый тип подключения. Для базовых подключений на вашем компьютере не требуется устанавливать какое-либо другое программное обеспечение Oracle. Явно указываются все параметры соединения.

CONNECT имя/пароль@хост:порт/имя_службы(SID)

TNS

TNS (Transparent Network Substrate) - это слой сетевого протокола в Oracle, который обеспечивает прозрачное и удобное соединение клиента с базой данных. Он позволяет клиентским приложениям обращаться к базам данных с использованием понятных имен сервисов, а не прямо указывать IP-адреса и порты серверов баз данных.

Для подключения к TNS используется запись псевдонима из файла tnsnames.ora.

CONNECT имя/пароль@псевдоним

LDAP

Oracle Internet Directory - это служба каталогов, которая позволяет централизованно хранить имена сетевых служб и управлять ими.

Дескриптор соединения – объединенная спецификация двух обязательных компонентов подключения к базе данных:

a. Имени службы базы данных

b. Местоположения адреса базы данных

```
(DESCRIPTION
        (ADDRESS = (PROTOCOL = TCP)
                (HOST = имя_хоста)
                (PORT = 1521))
        (CONNECT_DATA =
                (SERVICE_NAME = имя_службы_базы данных)))
```

39. Табличные пространства СУБД Oracle и их основные параметры.

Табличное пространство – логическая структура хранения данных. Оно объединяет один или несколько файлов данных для организации данных внутри базы данных.
Один файл данных может принадлежать только одному табличному пространству.
Одно табличное пространство может включать в себя один или несколько файлов данных.

PERMANENT – для постоянных данных: таблицы, индексы, представления, функции, процедуры и т.д. Может быть несколько, одно прописано по умолчанию. (users)

TEMPORARY – для временных файлов. Oracle использует временные табличные пространства в качестве рабочих областей для выполнения таких задач, как операции сортировки при выполнении запросов пользователей, и операции сортировки при создании индексов. Oracle не позволяет пользователям создавать объекты во временном табличном пространстве.

Oracle использует специальные структуры, именуемые **записями отмены** (undo records) для обеспечения автоматической согласованности чтения на уровне операторов. Если транзакция модифицирует данные, то Oracle сохранит их образ “до того” в записях отмены. Записи отмены Oracle хранятся в табличном пространстве отмены, специфицированном во время создания базы данных. Табличное пространство отмены (UNDO Tablespace) всегда содержит старые образы данных таблицы для пользователей на случай, если другие транзакции обновят их после запуска запроса.

UNDO – для отката: при выполнении, например, DML-оператора новые данные сохраняются в таблице в постоянном тейблспейсе, а старые данные (так называемые undo records – записи отмены) отправляются в UNDO- тейблспейс. Используются для роллбека транзакций или восстановления старых\поврежденных данных. Может быть создано несколько, но используется только одно, прописанное в файле параметров.

CREATE (TEMPORARY) TABLESPACE	
DATAFILE/TEMPFILE путь	Путь к файлу

AUTOEXTEND OFF или AUTOEXTEND ON NEXT (число K или M)	**AUTOEXTEND OFF** - параметр указывает, что средство автоувеличения размера файла использоваться не будет **AUTOEXTEND ON** - автоувеличение размера файла будет использовано. Дополнительно можно указать: - NEXT число K или M - когда файл данных самоувеличивается, он изменяется на указанный объем. (Килобайтах или в Мегабайтах)
SIZE число K или M	Начальный размер файла данных табличного пространства
MAXSIZE UNLIMITED или MAXSIZE число K или M	**MAXSIZE UNLIMITED** - размер файла будет ограничен лишь физическим диском и особенностями операционной системы. **MAXSIZE число K или M** - файл данных не может быть больше указанного объема.
NOLOGGING или LOGGING	-**LOGGING** - указывает, что в журнал выполненных операций будет заноситься информация о таблицах, индексах и разделах. Параметр по умолчанию. Журналирование может быть отменено для этих операций опцией NOLOGGING. -**NOLOGGING** - журналирование не будет выполняться для операций, поддерживающих эту опцию.
MINIMUM EXTENT число K или M	Указывает минимальный размер экстентов табличного пространства.
DEFAULT STORAGE storage_clause	Указывает параметры по умолчанию хранения данных в табличном пространстве
ONLINE или OFFLINE	**ONLINE** - табличное пространство становится оперативным сразу после своего создания. **OFFLINE** - табличное пространство недоступно непосредственно после своего создания (до тех пора, пока не будет переведено в оперативное состояние).

EXTENT MANAGEMENT LOCAL ALLOCATE или UNIFORM SIZE K или M; EXTENT MANAGEMENT DICTIONARY	табличное пространство будет использовать битовые карты для выделения экстентов информация о выделении и освобождении экстентов хранится в системных таблицах словаря данных
SEGMENT SPACE MANAGEMENT AUTO Manual	Сегменты в табличном пространстве будут использовать битовые карты для отслеживания использования блоков Использует списки свободных блоков для управления пространством

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

Привилегия - это право выполнять конкретный тип предложений SQL, или право доступа к объекту другого пользователя.

ORACLE имеет два вида привилегий: системные и объектные.

Назначаются оператором GRANT.

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

Отзываются оператором REVOKE

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

Объектная привилегия разрешает выполнение определенной операции над конкретным объектом (например, над таблицей). **Системная привилегия** разрешает выполнение операций над целым классом объектов.

SYSDBA и SYSOPER - специальные привилегии администратора, которые позволяют выполнять базовые задачи администрирования: запуск или остановка экземпляра БД; создание, удаление, открытие или монтирования базы и др

WITH ADMIN OPTION - позволяет получившему системные полномочия или роль предоставлять их в дальнейшем другими пользователям или ролям. Такое решение в

частности включает и возможность изменение или удаления роли. Аналогично **WITH GRANT OPTION** для объектных привилегий.

Объекты системных грантов:
DATABASE
USER
PROFILE
TABLESPACE
ROLE
TABLE
INDEX
TRIGGER
PROCEDURE
SEQUENCE
VIEW

Объекты объектных грантов:
TABLE
VIEW
SEQUENCE
PROCEDURE

Роль – это именованный набор привилегий.

```
SELECT * FROM DBA_ROLES
```

```
CREATE ROLE RLEACORE;

GRANT CREATE SESSION,
CREATE TABLE,
CREATE VIEW,
CREATE PROCEDURE TO RLEACORE;
```

Сиситемные роли:
DBA – предопределенная роль, которая автоматически создаётся для каждой базы данных Oracle и содержит все системные привилегии, кроме SYSDBA и SYSOPER
CONNECT: предоставляет минимальные привилегии, необходимые для установления соединения с базой данных.
RESOURCE: предоставляет привилегии, которые полезны для разработки и тестирования. Она включает в себя права на создание таблиц, процедур, триггеров и других объектов.

41. Пользователь СУБД Oracle и его основные параметры.

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.

- 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 безопасности 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).

```
CREATE USER new_user
IDENTIFIED BY password
DEFAULT TABLESPACE users
TEMPORARY TABLESPACE temp
QUOTA 100M ON users
PROFILE default
ACCOUNT UNLOCK;
```

42. Профиль безопасности СУБД Oracle и его основные параметры.

Профиль устанавливает ограничение на использование ресурсов конкретным пользователем, а также управляет паролями. Может ограничивать кол-во открытых сеансов, продолжительность соединения, время бездействия, использование ЦП, кол-во попыток ввода пароля, время его жизни и т.д.

```
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 назначенном профиле отсутствуют некоторые параметры, то они берутся из профиля по умолчанию

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

43. Таблица в СУБД Oracle и ее основные параметры. Типы данных базы данных. Ограничения целостности в таблицах.

В СУБД Oracle таблицы являются основными объектами, в которых хранятся данные. При создании таблицы можно указать различные параметры, влияющие на её структуру и поведение.

Основные параметры:

- Название таблицы (table_name): Имя создаваемой таблицы.
- Колонки (columns): Имена и типы данных для каждой колонки в таблице.
- Табличное пространство (tablespace): Определяет, в каком табличном пространстве будет храниться таблица.
- Ограничения (constraints): Правила, применяемые к данным в таблице (например, PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK).
- Индексы (indexes): Способы организации данных для быстрого доступа.
- Хранилище (storage): Параметры управления физическим хранением данных.

```
CREATE TABLE employees (
    employee_id NUMBER(10) PRIMARY KEY,
    first_name VARCHAR2(50),
    last_name VARCHAR2(50),
```

```
    email VARCHAR2(100) UNIQUE,
    hire_date DATE DEFAULT SYSDATE,
    salary NUMBER(8,2) CHECK (salary > 0),
    department_id NUMBER(10),
    CONSTRAINT fk_department
        FOREIGN KEY (department_id)
        REFERENCES departments(department_id)
)
TABLESPACE users
STORAGE (
    INITIAL 1M
    NEXT 1M
    MINEXTENTS 1
    MAXEXTENTS 50
    PCTINCREASE 0
);
```

TABLESPACE users: Указание табличного пространства, в котором будет храниться таблица.

Параметры хранения:

STORAGE: Параметры управления физическим хранением данных.

INITIAL 1M: Начальный размер сегмента таблицы.

NEXT 1M: Размер следующего экстента.

MINEXTENTS 1: Минимальное количество экстентов.

MAXEXTENTS 50: Максимальное количество экстентов.

PCTINCREASE 0: Процентное увеличение размера экстентов.

Дополнительные параметры

PARTITION BY: Определяет, как данные будут разбиваться на партиции.

ORGANIZATION INDEX: Указывает, что таблица организована как индекс.

CACHE: Определяет, будет ли кэшироваться содержимое таблицы в буферном кэше.

NOPARALLEL: Указывает, что операции с таблицей не будут выполняться параллельно.

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

Параметр PCTUSED – процент занятой части памяти блока

Automatic segment space management - только PCTFREE

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

Символьные типы данных:

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

Числовые типы данных:

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

Дата и 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 TIMESTAMP	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

Ограничения целостности:

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

44. Временные таблицы СУБД Oracle.

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

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.
Место для временных таблиц выделяется во временных сегментах только после первого применения команды INSERT с этими таблицами (по умолчанию во временном табличном пространстве TEMP). Каждый пользователь видит только свои данные (свой сегмент данных).

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

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

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

45. Индексы базы данных СУБД Oracle. Виды и особенности применения индексов.

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

Индекс создается с помощью оператора CREATE INDEX.

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

Селективность индекса — значение, представляющее отношение количества уникальных значений индексируемых столбцов к общему числу строк таблицы

Типы индексов:

Функциональный индекс – предварительно вычисляют значения функции по заданному столбцу и сохраняют результат в индексе. Он позволяет ускорить выполнение запросов, которые используют выражения или функции для обработки данных в столбцах таблицы.

```
CREATE INDEX idx_employee_name_length ON employees (LENGTH(last_name));
```

Битовый индекс в Oracle создает битовые карты для каждого возможного значения столбца. В каждой битовой карте каждому биту соответствует строка, а значение 1 (0) указывает на наличие (отсутствие) индексируемого значения. Он предназначен для индексации столбцов с низкой селективностью и хорошо подходит для хранилищ данных, где таблицы обычно имеют большие объемы данных. Однако битовые индексы не рекомендуется использовать для таблиц с частым обновлением, так как они могут привести к частым блокировкам и увеличению накладных расходов при обновлении индексов. Эти индексы создаются с использованием битового потока, что делает их очень компактными по сравнению с индексами на основе B-деревьев.

```
CREATE BITMAP INDEX idx_gender ON employees (gender);
```

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
Это **обычные индексы**, создаваемые по умолчанию, когда вы применяете оператора CREATE INDEX.

ROWID представляет собой уникальный идентификатор строки в таблице Oracle.
Он содержит информацию о файле данных, блоке данных и местоположении строки внутри блока. ROWID может быть использован для быстрого доступа к конкретной строке в таблице, так как он является фактическим адресом строки внутри базы данных.
Первые шесть символов – это номер объекта данных, следующие три – относительный номер файла, следующие шесть – номер блока, последние три – номер строки.

46. Последовательность СУБД Oracle и ее параметры.

Последовательность – это объект базы данных, который генерирует целые числа в соответствии с правилами, установленными во время его создания.

Параметры:
start with —первое генерируемое ею значение.
increment by n — определяет приращение последовательности при каждой ссылке на виртуальный столбец NEXVAL. Если значение не указано явно, по умолчанию устанавливается 1.
minvalue — определяет минимальное значение, создаваемое последовательностью. Если оно не указано, Oracle применяет значение по умолчанию NOMINVALUE
nominvalue — указывает, что минимальное значение равно 1, если последовательность возрастает, или -10(26), если последовательность убывает

maxvalue — определяет максимальное значение, создаваемое последовательностью. Если оно не указано, Oracle применяет значение по умолчанию NOMAXVALUE
nomaxvalue — указывает, что максимальное значение равно 10(27), если последовательность возрастает, или -1, если последовательность убывает. По умолчанию принимается NOMAXVALUE
cycle — позволяет последовательности повторно использовать созданные значения при достижении MAXVALUE или MINVALUE. Т.е. последовательность будет продолжать генерировать значения после достижения своего максимума или минимума. Возрастающая последовательность после достижения своего максимума генерирует свой минимум. Убывающая последовательность после достижения своего минимума генерирует свой максимум.
nocycle — указывает, что последовательность не может продолжать генерировать значения после достижения своего максимума или минимума
cache n — указывает, сколько значений последовательности ORACLE распределяет заранее и поддерживает в памяти для быстрого доступа. Минимальное значение этого параметра равно 2. Для циклических последовательностей это значение должно быть меньше, чем количество значений в цикле. Если кэширование нежелательно или не установлено явным образом, Oracle применяет значение по умолчанию – 20 значений.
order — гарантирует, что номера последовательности генерируются в порядке запросов.
noorder — не гарантирует, что номера последовательности генерируются в порядке запросов
Привилегия необходимая для создания последовательности : CREATE SEQUENCE

47. Кластер и его параметры.

Кластерами называют две или более таблиц, которые физически хранятся вместе, чтобы использовать преимущества совпадающих между таблицами столбцов. Если две таблицы имеют идентичный столбец и вам, к примеру, часто приходится соединять таблицы по нему, то становится выгодно хранить значения общих столбцов в одном и том же блоке данных. Целью такой организации является сокращение объема ввода-вывода с увеличением скорости доступа при соединении связанных таблиц. Однако кластеры понижают производительность операторов INSERT, поскольку для хранения данных нескольких таблиц требуется больше блоков.

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

Хэш-кластер создается с помощью команды **CREATE CLUSTER**. Далее в скобках указывается столбец или набор столбцов (ключевые столбцы), которые будут **использоваться для кластеризации данных**. Например, следующее предложение создает кластер с именем TRIAL_CLUSTER, который используется для хранения таблицы TRIAL, кластеризуемой по столбцу TRIALNO. Столбцы в таблицах не обязательно должны называться TRIALNO, но они должны быть типа NUMBER(5, 0):

```
CREATE CLUSTER trial_cluster (trialno NUMBER(5,0))
    PCTUSED 80
    PCTFREE 5
    SIZE 2K
    HASH IS trialno
    HASHKEYS 150;

CREATE TABLE trial (
    trialno NUMBER(5,0) PRIMARY KEY,
    )
    CLUSTER trial_cluster (trialno);
```

PCTUSED: Устанавливает процент заполнения блока данных в кластере до того, как он будет считаться полным для вставки строк. В данном случае 80% означает, что Oracle будет пытаться заполнить каждый блок данных на 80% прежде, чем переходить к следующему блоку.

PCTFREE: Определяет минимальный процент свободного места, которое должно оставаться в блоке данных кластера. Это нужно для обновлений данных в строках.

SIZE: Устанавливает размер блока данных кластера в байтах. В данном случае размер блока - 2 килобайта.

HASH IS: Определяет столбец trialno как столбец хеширования. Это означает, что Oracle будет использовать хеш-функцию для распределения строк по блокам данных кластера на основе значений в столбце trialno.

HASHKEYS: Задает количество блоков данных, которые будут выделены для хранения хеш-значений.

48. Представление в СУБД Oracle и его параметры.

Представление — результат хранимого запроса. Представление не существует физически как обычная таблица, являющаяся частью табличного пространства. Фактически представление создает виртуальную таблицу или подтаблицу только с теми строками и/или столбцами, которые нужно показать пользователю. При условии, что пользователь имеет соответствующие права доступа к лежащим в основе представления таблицам, можно запрашивать представления или даже модифицировать, удалять либо добавлять данные с использованием операторов UPDATE, DELETE и INSERT.

Чтобы создать представление в своей схеме, необходимо иметь системную **привилегию CREATE VIEW**, а чтобы создать представление в любой схеме, а не только в собственной, понадобится системная **привилегия CREATE ANY VIEW**.

Синтаксис создания представления:

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW
[schema.]viewname [(alias [,alias]…]
AS subquery
[WITH CHECK OPTION [CONSTRAINT constraintname]]
[WITH READ ONLY [CONSTRAINT constraintname]] ;
```

OR REPLACE – если представление уже существует оно будет удалено перед созданием нового

FORCE или NOFORCE – Использование FORCE приведёт к созданию представления даже если базовые таблицы не существуют. NOFORCE значение по умолчанию и если таблицы не существуют команды выполняется с ошибкой

WITH CHECK OPTION – эта директива влияет на DML команды. Если подзапрос включает условие WHERE, тогда эта директива предотвратит возможность вставки строк, которые не видно в представлении, или совершать обновления данных которое приведёт к пропаже данных из представления. По умолчанию эта директива отключена что может приводить к неожиданным результатам выполнения запросов

WITH READ ONLY – отключения возможности использование DML команд к представлению

CONSTRAINT constraintname – позволяет назначить имя ограничениям WITH CHECK OPTION и WITH READ ONLY и сообщения об ошибке станут более понятными

49. Материализованное представление и его параметры.

Всякий раз, когда нужен доступ к представлению, Oracle должен выполнить запрос,по которому определено представление, и вернуть результат. Этот процесс наполнения представления называется разрешением представления (view resolution) и он повторяется при каждом обращении пользователя к представлению. Если вы имеете дело с представлениями с множеством конструкций JOIN и GROUP BY, то этот процесс разрешения представления может потребовать очень длительного времени. Если нужно часто обращаться к представлению, будет весьма неэффективно каждый раз повторять разрешение представления.

Материализованные представления Oracle предлагают выход из этого затруднения. Упомянутые представления можно воспринимать как специализированные представления, в отличие от обычных представлений, имеющие физическое воплощение. Они занимают место и требуют хранения подобно обычным таблицам. Материализованные представления можно даже секционировать и при необходимости создавать на них индексы. . Привилегия – CREATE MATERIALIZED VIEW

```
CREATE MATERIALIZED VIEW sales_summary
BUILD IMMEDIATE
```

```
REFRESH COMPLETE
ON DEMAND
ENABLE QUERY REWRITE
AS
```

- Предложение **BUILD IMMEDIATE** заполняет материализованное представление во время его создания (значение по умолчанию). Альтернативное предложение **BUILD DEFERRED** создает только структуру;
- Предложение **REFRESH** указывает, как оракл обновляет данные материализованного представления. **COMPLETE** — полное обновление данных из базовых таблиц. **FAST** – обновлять данные более эффективно, пересчитывая только измененные данные, а не всю таблицу. **FORCE** – попытка быстрого обновления; если быстрое обновление невозможно, то выполняется полное обновление
- В предложении REFRESH также указывается, когда оракл обновляет данные материализованного представления. В этом примере, материализованное представление будет обновляться по требованию (**ON DEMAND**) – только, когда вы явно обновляете его. (**ON COMMIT**) – каждый раз когда выполняется фиксация транзакции для мастер таблицы (таблиц) представления.
- **QUERY REWRITE** Oracle имеет возможность использовать это представление для автоматического переписывания или замены запросов пользователей на более эффективные запросы, использующие данные из этого материализованного представления.
- Предложение AS описывает столбцы и строки материализованного представления, используя определяющий запрос.
- **START WITH** – показывает, когда выполнится в первый раз (если не был построен сразу)
- **NEXT**– определяет интервал между последующими обновлениями

50. Частные и публичные синонимы СУБД Oracle.

Синонимы— это псевдонимы объектов базы данных, которые служат в основном для облегчения пользователям доступа к объектам, принадлежащим другим пользователям, а также в целях безопасности. Синонимы скрывают идентичность лежащих в их основе объектов и могут быть как приватными (private), так и общедоступными (public). Общедоступные синонимы доступны всем пользователям базы данных,а приватные синонимы являются составной частью схемы отдельного пользователя, и другим пользователям базы следует выдавать права доступа для использования приватных синонимов. Может указывать на: **таблицы, процедуры, функции, последовательности, представления, пакеты, объекты** в локальной или удаленной базе данных. Привилегия – CREATE (PUBLIC) SYNONYM.

Возможность видеть таблицу через общедоступный (или приватный) синоним, еще не означает возможность выполнения над ней операций SELECT, INSERT, UPDATE или DELETE. Для выполнения таких операций пользователю нужны специальные привилегии для доступа к исходному объекту, выданные владельцем приложения непосредственно или через роли.

Общедоступные (public) синонимы относятся к специальной схеме базы данных Oracle, именуемой PUBLIC. Общедоступные синонимы видны всем пользователям базы данных. Общедоступные синонимы обычно создаются владельцем приложения для таблиц и прочих объектов, таких как процедуры и пакеты,чтобы пользователи приложения могли видеть эти объекты.

Приватные синонимы, в отличие от общедоступных, видны только в схеме, владеющей таблицей или объектом. Приватные синонимы можно создать, когда нужно обращаться к одной и той же таблице в разных контекстах под разными именами. Они создаются точно так же, как и общедоступные, но без ключевого слова PUBLIC в операторе CREATE.

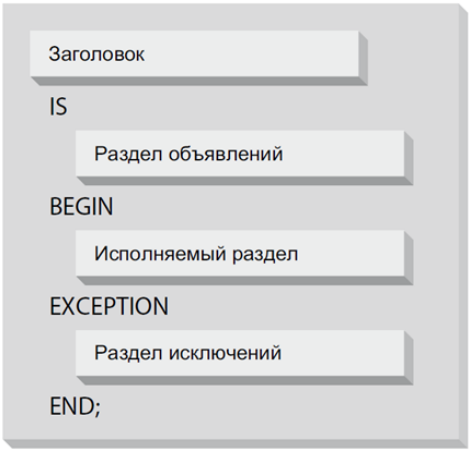
И приватный, и общедоступный синонимы уничтожаются командой DROP SYNONYM, но есть одно отличие. При уничтожении общедоступного синонима после ключевого слова DROP должно находиться ключевое слово PUBLIC.

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM [schema .] synonym_name
  FOR [schema .] object_name;
```

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

В PL/SQL код не выполняется в однострочном формате, а всегда выполняется путем группировки кода в один элемент, называемый **блоками**. В блоке PL/SQL может содержаться до 4-х разделов, но лишь один считается обязательным.

1. **Заголовок**. Применяется лишь в именованных блоках, служит для определения способа вызова программы либо именованного блока.
2. **Раздел объявлений**. Включает описания переменных, вложенных блоков и курсоров.
3. **Раздел исключений**. Служит для обработки исключений (предупреждений и ошибок).
4. **Исполняемый раздел**. Речь идёт о командах, которые выполняются ядром PL/SQL в процессе работы приложения. Обратите внимание, что это ОБЯЗАТЕЛЬНЫЙ раздел.



В **анонимном блоке** нет раздела заголовка, блок начинается ключевым словом DECLARE (или BEGIN). Анонимный блок не может быть вызван из другого блока, поскольку он не имеет идентификатора, по которому к нему можно было бы обратиться. Таким образом, анонимный блок представляет собой контейнер для хранения команд PL/SQL — обычно с вызовами процедур и функций. Их необходимо создавать и использовать в одном сеансе, поскольку они не будут храниться на сервере как объекты базы данных.

Именованные блоки имеют определенное и уникальное имя. Они хранятся как объекты базы данных на сервере. Поскольку они доступны как объекты базы данных, к ним можно обращаться или использовать, пока они присутствуют на сервере. Структура блока такая же, как у анонимного блока, за исключением того, что он никогда не начинается с ключевого слова «DECLARE». Вместо этого он будет начинаться с ключевого слова CREATE, которое указывает компилятору создать его как объект базы данных. Именованные блоки бывают двух типов: функции или процедуры.

52. Типы данных, основные операции, константы языка PL/SQL. Псевдостолбцы.

Символьные типы данных:

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

Числовые типы данных:

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

Дата и 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 TIMESTAMP	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

Логические типы данных:
PL/SQL поддерживает тип данных **BOOLEAN**. Переменные этого типа могут принимать одно из трех значений (TRUE, FALSE или NULL).

Тип данных REF CURSOR:

Курсорная переменная в PL/SQL представляет собой переменную, которая содержит ссылку на курсор. Это позволяет создавать динамические курсоры и управлять ими во время выполнения программы. Курсорная переменная объявляется с использованием специального типа данных **REF CURSOR** в PL/SQL.

Числовые операторы

Оператор	Операция	Приоритет
**	Возведение в степень	1
+	Тождество	2
-	Отрицание	2
*	Умножение	3
/	Деление	3
+	Сложение	4
–	Вычитание	4
=	Равно	5
<	Меньше чем	5

>	Больше чем	5
<=	Меньше либо равно	5
>=	Больше либо равно	5
<>, !=, ~=, ^=	Не равно	5
IS NULL	Проверка неопределенности	5
BETWEEN	Принадлежность диапазону	5
NOT	Логическое отрицание	6
AND	Конъюнкция	7
OR	Дизъюнкция	8

Идентификаторы – это имена переменных. Идентификаторы используются для присвоения имен любым объектам PL/SQL.

- Не более 30 символов
- Начинается с буквы
- Не содержит пробелов
- Может включать $ _ #
- Компилятор приводит идентификаторы к верхнему регистру
- “идентификатор” регистрозависим

Инициализация переменной происходит с помощью **оператора присваивания** (:=). Если вы не задаете начальное значение сами, то по умолчанию новая переменная принимает неопределенное значение (NULL), которое сохраняется до тех пор, пока вы не присвоите другое значение позже.

Неявные преобразования происходят автоматически, когда Oracle Database конвертирует один тип данных в другой без явного указания на это в коде. Эти преобразования могут происходить в различных ситуациях, таких как выполнение выражений, присваивание значений переменным, передача параметров процедурам и функциям, а также при выполнении операторов SQL.

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

Явные преобразования:

1. TO_NUMBER: Преобразует строку в число.

2. TO_CHAR: Преобразует число или дату в строку. Также можно указать формат даты или числа.

3. TO_DATE: Преобразует строку в дату. Формат строки должен соответствовать указанному формату.

4. TO_TIMESTAMP: Преобразует строку в метку времени (timestamp).

5. TO_CLOB / TO_NCLOB: Преобразует строку в объект CLOB или NCLOB.

6. CAST: Универсальная функция для преобразования одного типа данных в другой.

```
SELECT CAST('1234.56' AS NUMBER) FROM DUAL;

SELECT CAST(SYSDATE AS VARCHAR2(30)) FROM DUAL;

SELECT CAST(1234 AS CHAR(10)) FROM DUAL;
```

Oracle/PLSQL **оператор конкатенации** || позволяет объединить две или более строк вместе.

Синтаксис **объявления константы** имеет следующий вид:

```
имя_переменной тип_данных CONSTANT := выражение;
```

В отличие от переменных **константам** обязательно присваивается значение, которое нельзя изменять на протяжении времени жизни константы. Константы очень полезны для поддержания безопасности и дисциплины при разработке больших и сложных приложений. Например, если вы хотите гарантировать, что процедура PL/SQL не будет

модифицировать передаваемые ей данные, можете объявить их константами. Если процедура все же попытается их модифицировать, PL/SQL возбудит исключение.

Псевдостолбцы- это дополнительные функции, которые можно вызвать только из SQL-операторов. Синтаксически они аналогичны столбцам таблиц, однако реально совсем не похожи на них. Их скорее можно назвать частью процесса выполнения SQL-операторов.

Псевдостолбцы **CURRVAL** (текущее значение) и **NEXTVAL** (следующее значение) применяются в последовательностях. Последовательность (sequence) — это объект Oracle, который используется для генерирования уникальных чисел. Последовательность создается с помощью DDL-команды CREATE SEQUENCE. После того как последовательность создана, к ней можно обратиться.

ROWID — это псевдостолбец, который является уникальным идентификатором строки в таблице и фактически описывает точное физическое расположение данной конкретной строки. При каждом перемещении, экспорте, импорте строки, а также при выполнении любых других операций, которые приводят к изменению ее местонахождения, изменяется ROWID строки, поскольку она занимает другое физическое положение. Для хранения данных ROWID требуется 80 бит (10 байт). Идентификаторы ROWID состоят из четырех компонентов: номера объекта (32 бита), относительного номера файла (10 бит), номера блока (22 бита) и номера строки (16 бит). Эти идентификаторы отображаются как 18-символьные последовательности, указывающие местонахождение данных в БД, причем каждый символ представлен в формате base-64, состоящем из символов A-Z, a-z, 0-9, + и /. Первые шесть символов – это номер объекта данных, следующие три – относительный номер файла, следующие шесть – номер блока, последние три – номер строки.

Псевдостолбец **ROWNUM** возвращает номер текущей строки запроса. Он полезен для ограничения общего числа строк и используется в первую очередь в условиях WHERE запросов и в предложениях SET операторов UPDATE. Значение, возвращаемое ROWNUM, имеет тип NUMBER. Отсчёт с 1.

ROWSCN (системный номер изменения строки) представляет собой число, которое увеличивается при каждом изменении строки в таблице. Это используется для контроля версий данных и предоставляет механизм определения времени, когда была внесена последняя модификация в строку. Когда строка обновляется, ей присваивается новый ROWSCN. ROWSCN можно использовать, например, для определения, когда данные были изменены, и выполнения действий на основе времени изменения данных.

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

Подсистема поддержки национальных языков (National Language Support, NLS) предоставляет наборы символов и прочие данные, например форматы записи чисел и дат, для различных языков.

Запросы для просмотра NLS параметров:

```
--Параметры базы данных (задаются при создании)
SELECT * FROM nls_database_parameters

--Параметры экземпляра (можно изменить с помощью ALTER SYSTEM)
SELECT * FROM nls_instance_parameters

--Клиентские (на сессию) параметры (можно изменить с помощью ALTER SESSION)
SELECT  * FROM nls_session_parameters
```

Какая из кодировок (клиент/сервер) имеет приоритет? Параметры сеанса (клиентские) имеют приоритет над параметрами экземпляра и базы данных. Параметры экземпляра имеют приоритет выше, чем настройки базы данных.

Переменная окружения NLS_LANG состоит из трёх частей **NLS_LANG = language_territory.charset**

Язык (LANGUAGE) – имена месяцев, имена дней, направление текста, сокращения для времени и дат. По умолчанию AMERICAN

Территория (TERRITORY) – настройки календаря, формат даты, формат денежной единицы. Если не указан, то будет взято значение, соответствующее языку (для RUSSIAN - CIS)

Набор символов (CHARACTER SET) – отображение символов, отображение и конвертация заглавных букв, порядок замещения символов при преобразовании. Каждому языку поставлен в соответствие набор символов по умолчанию

Основной набор символов используется для:

- хранения символьных типов char, varchar2, clob и long
- описания имен объектов, переменных
- Ввода и хранения PL/SQL модулей

Дополнительный набор символов используется для:

- хранения символьных типов nchar, nvarchar2, nclob

Кроме символов алфавита в набор включаются знаки препинания, числа, символы денежных единиц и пр.

Байтовая семантика рассматривает строки как последовательность байтов

Символьная семантика рассматривает строки как последовательность символов

Задается параметром **NLS_LENGTH_SEMANTICS**. По умолчанию - BYTE

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

%TYPE и **%ROWTYPE** используются для объявления переменных в PL\SQL с тем типом, который соответствует определенному объекту в БД. Если тип или размерность поля в БД изменится, программа автоматически подхватит изменения без надобности вносить изменения в код.

%TYPE используется для объявления переменной того же типа, что и поле в таблице.

```
v_name  table_name.column_name%TYPE
```

%ROWTYPE используется для объявления записи того же типа, что и строка в необходимой таблице, представлении или курсоре.

```
v_rec table_name%ROWTYPE
dept_rec1 = v_rec.column_name -- дальнейшее использование
```

55. Локальные процедуры и функции языка PL/SQL.

Локальный (или вложенный) модуль — это процедура или функция PL/SQL, определяемая в разделе объявлений блока PL/SQL (анонимного или именованного). Локальным такой модуль называется из-за того, что он определяется только внутри родительского блока PL/SQL и не может быть вызван из другого блока, определенного вне родительского.

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

Локальные процедуры и функции могут быть **перегружены** при соблюдении следующих условий:

- Параметры должны отличаться семейством (number, character, datetime, boolean)
- Тип программного модуля должен отличаться – можно перегружать процедуру и функцию с одинаковым именем и списком параметров
- Число параметров должно быть разным

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

Запись (Record) — это группа связанных элементов данных, которые хранятся в полях, причем каждая имеет своё имя и тип данных. При этом мы можем применять Record как переменную, способную содержать строку таблицы либо некоторые столбцы из строки таблицы.

Объявить запись поможет атрибут **%ROWTYPE**, представляющий строку в таблице БД (представлении, курсоре) без непосредственного перечисления всех столбцов. При этом ваш код будет продолжать функционировать даже после добавления в таблицу столбцов. Пример:

```
CREATE TABLE books (
   book_id         INTEGER,
   author          VARCHAR2(200),
   date_published  DATE
);
DECLARE
   my_book books%ROWTYPE;
BEGIN
   SELECT *
     INTO my_book
     FROM books
     WHERE title = 'Oracle PL/SQL Programming, 6th Edition';
   IF my_book.author LIKE '%Feuerstein%'
   THEN
     DBMS_OUTPUT.put_line ('Код ISBN: ' || my_book.isbn);
   END IF;
END;
```

Синтаксис определения **типа Record и объявления переменной** в Oracle PL/SQL:

```
TYPE type_rec_name IS Record  (field_1 datatype,
                               field_2 datatype,
                              … field_n datatype);
var_rec type_rec_name;
```

Параметры или аргументы:

type_rec_name – имя определенного типа Record

var_rec – имя переменной типа Record

field_1, field_2,… field_n – поля типа Record

datatype – тип данных для полей типа Record.

Следующий пример показывает, как можно определить **тип Record как вложенную в другой тип Record**.

```
DECLARE
  TYPE name_rec IS RECORD (
    first  employees.first_name%TYPE,
    last   employees.last_name%TYPE
  );

  TYPE contact IS RECORD (
    name  name_rec,              -- вложенная запись name_rec
    phone employees.phone_number%TYPE
  );
  friend contact;--переменная типа contact
BEGIN
  friend.name.first := 'John';--доступ к вложенной записи
  friend.name.last := 'Smith';
  friend.phone := '1-650-555-1234';

  DBMS_OUTPUT.PUT_LINE (
    friend.name.first  || ' ' || friend.name.last   || ', ' ||
    friend.phone
  );
END;
```

В этом примере определили тип name_rec как вложенную запись типа contact. Доступ к полям вложенного типа записи получили с помощью точечной нотации.

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

PL/SQL предоставляет **три различные конструкции для итеративной обработки**. Каждая из них позволяет циклически выполнять набор операторов PL/SQL. Выход из цикла осуществляется в зависимости от некоторого условия.

```
имя_цикла LOOP
операторы;
EXIT имя_цикла [WHEN условие_выхода];
операторы;
END LOOP;
```

```
WHILE условие_выхода LOOP
операторы;
END LOOP;
```

```
FOR счетчик IN [REVERSE] нижняя_граница .. верхняя_граница LOOP
операторы;
END LOOP;
```

Оператор IF имеет следующий синтаксис:

```
IF условие_1 THEN
действие_1;
[ELSIF условие_2 THEN действие_2;]
…
[ELSE альтернативное_действие;]
END IF;
```

Оператор Case имеет следующий синтаксис:

```
CASE (expression)
 WHEN <valuel> THEN action_blockl;
 WHEN <value2> THEN action_block2;
 WHEN <value3> THEN action_block3;
 ELSE action_block_default;
END CASE;
```

58. Курсоры. Виды курсоров. Схемы обработки курсора. Атрибуты курсора. Курсоры с параметрами. Динамические курсоры.

Курсор – это средство извлечения данных из базы данных Oracle.

При каждом выполнении команды DML (INSERT, UPDATE, MERGE или delete) или команды SELECT INTO, возвращающей строку из базы данных в структуру данных программы, PL/ SQL автоматически создает для нее курсор. Курсор этого типа называется **неявным**, поскольку Oracle автоматически выполняет многие связанные с ним операции, такие как выделение курсора, его открытие, выборку строк и т. д.

Явные курсоры - вы можете явно объявить курсов в разделе объявлений (локального блока или пакета). В этом случае курсор можно будет открывать и извлекать данные в одной или нескольких программах, причем возможности контроля будут шире, чем при использовании неявных курсоров.

Сначала необходимо объявить курсор в PL/SQL блоке в разделе объявлений с помощью ключевого слова **CURSOR**, указав запрос, который будет выполняться при открытии курсора. Курсор можно объявить как с параметрами, так и без них.

```
CURSOR cursor_name IS SELECT column1, column2 FROM table_name WHERE condition;
```

После объявления курсора его нужно открыть с помощью оператора **OPEN**, создавая новый результирующий набор на базе указанного запроса.

```
OPEN cursor_name;
```

После открытия курсора можно использовать циклы или оператор **FETCH** для извлечения данных из курсора по одной строке за раз. FETCH — выполняет последовательное извлечение строк из результирующего набора от начала до конца.

Данные могут быть присвоены переменным или использованы для выполнения каких-либо действий.

```
LOOP
          FETCH cursor_name INTO variable1, variable2;
          EXIT WHEN cursor_name%NOTFOUND;
END LOOP;
```

По завершении работы с курсором его следует закрыть с помощью оператора **CLOSE**. Это освободит ресурсы, занятые курсором, и завершит его работу.

При **использовании FOR-цикла** PL/SQL автоматически открывает курсор перед началом цикла и закрывает его после завершения работы с ним. Это упрощает код и обеспечивает автоматическое управление ресурсами. При проходе явного курсора при помощи for-цикла создается переменная при начале описания цикла, которая будет содержать строку, берущуюся из курсора на каждой итерации цикла for.

```
FOR record IN cursor_name LOOP
      DBMS_OUTPUT.PUT_LINE('Column1: ' || record.column1 || ', Column2: ' ||
record.column2);
END LOOP;
```

Атрибуты курсора — это свойства, которые позволяют получить информацию о текущем состоянии курсора в PL/SQL.

%ISOPEN — возвращает значение TRUE, если курсор открыт

%FOUND — определяет, найдена ли строка, удовлетворяющая условию

%NOTFOUND — возвращает TRUE, если строка не найдена

%ROWCOUNT — возвращает номер текущей строки

Явные:

Атрибут	*Описание*
%ISOPEN	TRUE, если курсор открыт. FALSE, если курсор закрыт.
%FOUND	Ошибка INVALID_CURSOR, если курсор еще не открыт или уже закрыт. NULL перед первой выборкой. TRUE, если запись была успешно выбрана. FALSE, если запись не выбрана.
%NOTFOUND	Ошибка INVALID_CURSOR, если курсор еще не открыт или уже закрыт. NULL перед первой выборкой. FALSE, если запись была успешно выбрана. TRUE, если запись не выбрана.
%ROWCOUNT	Ошибка INVALID_CURSOR, если курсор еще не открыт или уже закрыт. Количество строк, выбранных в курсоре.

Неявные:

Атрибут	*Описание*
%ISOPEN	Всегда FALSE, так как курсор открывается неявно и закрывается сразу после выполнения
%FOUND	NULL перед выполнением. TRUE, если одна или более строк были вставлены, изменены, удалены или одна строка выбрана. Иначе FALSE
%NOTFOUND	NULL перед выполнением. FALSE, если одна или более строк были вставлены, изменены, удалены или одна строка выбрана. Иначе TRUE
%ROWCOUNT	Количество строк, выбранных в курсоре

Параметры курсора используются для передачи значений в запрос, который используется в курсоре. Они позволяют использовать один и тот же запрос с разными наборами параметров без необходимости изменения самого запроса. Параметры курсора указываются в объявлении курсора.

```
DECLARE
        CURSOR cursor_name (param1 VARCHAR2, param2 NUMBER) IS
        SELECT column1, column2
        FROM table_name
        WHERE condition_column = param1
        AND another_condition_column = param2;
```

```
OPEN имя_курсора [ ( аргумент, ….) ];
```

Динамические курсоры в Oracle используются для выполнения SQL-запросов, определяемых во время выполнения программы. Они позволяют выполнять различные SQL-запросы, которые могут быть известны только во время выполнения, а не во время компиляции. Это достигается использованием динамического SQL через EXECUTE IMMEDIATE.

Oracle/PLSQL **оператор EXECUTE IMMEDIATE** подготавливает (анализирует) и немедленно выполняет динамический SQL-запрос или анонимный PL/SQL блок.

С помощью EXECUTE IMMEDIATE можно динамически выполнять практически любую SQL строку в Oracle, включая DML и DDL операции, PL/SQL блоки и SELECT запросы.

```
EXECUTE IMMEDIATE dynamic_string
[ INTO define_variable]
[USING [IN | OUT | IN OUT] bind_argument ]
returning_clause;
```

dynamic_string

Строковый литерал, переменная или выражение, представляющее один оператор SQL или блок PL/SQL. Он должен иметь тип CHAR или VARCHAR2, а не NCHAR или NVARCHAR2.

INTO

Используется только для однострочных запросов, в этом разделе указываются переменные или записи, в которые извлекаются значения столбцов. Для каждого значения, полученного запросом, в предложении INTO должна быть соответствующая тип-совместимая переменная или поле.

bind_argument

Выражение, значение которого передается в динамический оператор SQL, или переменная, в которой сохраняется значение, возвращаемое динамическим оператором SQL.

USING

По умолчанию - IN. Определяет список входных и/или выходных аргументов привязки.

returning_clause

Возвращает значения из вставленных строк, устраняя необходимость SELECT строки после. Вы можете извлечь значения столбца в переменные или в коллекции. Вы не можете использовать предложение RETURNING для

удаленной или параллельной вставки. Если инструкция не влияет ни на какие строки, значения переменных, указанных в предложении RETURNING, не определены.

```
-- Динамический SQL-запрос
dynamic_string := 'SELECT first_name FROM employees WHERE employee_id = :1';

-- Выполнение запроса и присвоение результата переменной
EXECUTE IMMEDIATE dynamic_string INTO employee_name USING employee_id;
```

59. Курсоры. Курсорные переменные. Курсорные подзапросы. Использование конструкции CURRENT OF в курсорах.

Курсорная переменная в PL/SQL представляет собой переменную, которая содержит ссылку на курсор.

Во всех приведенных выше примерах явных курсоров рассматривались статические курсоры (static cursors), связанные с одним SQL-оператором, который был известен при компиляции блока. Курсорная же переменная (cursor variable) может быть связана с различными операторами во время выполнения программы. Курсорные переменные аналогичны переменным PL/SQL, в которых могут содержаться различные значения. Статические же курсоры аналогичны константам PL/SQL, так как они могут быть связаны только с одним запросом на этапе выполнения программы.

- Курсорные переменные могут передаваться в аргументах процедур или функций.
- Полноценное использование функциональности статических курсоров PL/SQL. Вы можете применять OPEN, CLOSE и FETCH к своим курсорным переменным в программах PL/SQL, а также обращаться к стандартным атрибутам курсоров — %ISOPEN, %FOUND, %NOTFOUND и %ROWCOUNT.
- Присваивание содержимого одного курсора (и его результирующего набора) другой курсорной переменной. Курсорная переменная, как и любая другая переменная, может использоваться в операциях присваивания.

Чтобы воспользоваться курсорной переменной, ее необходимо предварительно объявить. Во время выполнения программы должна быть выделена память для хранения этой переменной, так как курсорные переменные имеют тип. Затем ее можно открывать, считывать и закрывать так же, как статический курсор.

Курсорная переменная объявляется в два этапа.

1. Определение типа курсора.
2. Объявление фактической переменной на базе этого типа.

Синтаксис объявления типа курсора:

```
TYPE имя_типа_курсора IS REF CURSOR [ RETURN возвращаемый_тип];
```

Здесь имя курсора — имя типа курсора, а возвращаемый тип — спецификация возвращаемых данных курсора (не обязательно). В Oracle9i появился предопределенный тип REF CURSOR с именем SYS_REFCURSOR.

```
DECLARE my_cursor SYS_REFCURSOR;
```

Значение (объект курсора) присваивается курсорной переменной при открытии курсора. Таким образом, синтаксис традиционной команды OPEN позволяет использовать после секции FOR команду SELECT:

```
OPEN имя_курсора FOR команда_select;
```

Курсорное выражение — это выражение со специальным оператором CURSOR, используемое в SQL-запросе и определяющее вложенный курсор.

Синтаксис курсорного выражения очень прост:

```
CURSOR (вложенный_запрос)
```

Когда вы выполняете запрос, содержащий вложенный курсор, Oracle автоматически открывает этот вложенный курсор для каждой строки внешнего (родительского) курсора. Вложенный курсор выполняется и возвращает свой результат для каждой строки внешнего запроса.

Он закрывается, когда:

- вы явно закрываете курсор;
- родительский курсор повторно выполняется, закрывается или отменяется;
- при выборке из родительского курсора инициируется исключение. В этом случае вложенный курсор закрывается вместе с родительским.

Существует два разных, но очень полезных способа использования курсорных выражений:

- для выборки вложенного запроса как столбца внешнего запроса;
- для преобразования запроса в результирующий набор, который может передаться в аргументе функции.

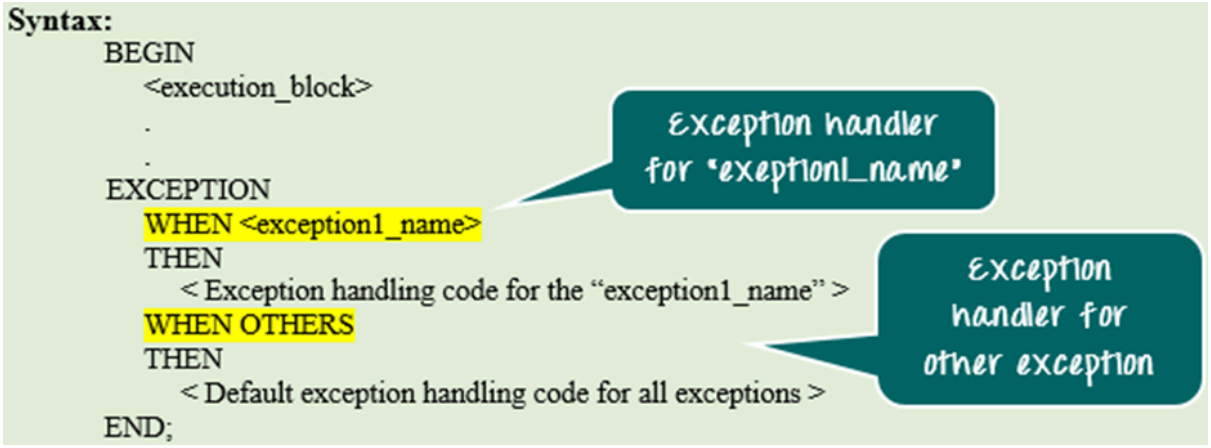
PL/SQL позволяет использовать в командах UPDATE и DELETE специальную конструкцию **WHERE CURRENT OF**, облегчающую процесс изменения последней выбранной из курсора строки данных.

```
UPDATE имя_таблицы SET
 предложение_set WHERE CURRENT
```

```
OF имя_курсора;
```

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

Исключение возникает, когда механизм PL / SQL встречает инструкцию, которую он не может выполнить из-за ошибки, возникающей во время выполнения. Исключения будут препятствовать дальнейшему выполнению программы, поэтому, чтобы избежать такого условия, они должны быть записаны и обработаны отдельно. Этот процесс называется обработкой исключений, при которой программист обрабатывает исключение, которое может возникнуть во время выполнения.



- Каждое условие WHEN сопровождается именем исключения, которое, как ожидается, будет вызвано во время выполнения.
- Когда какое-либо исключение возникает во время выполнения, механизм PL / SQL будет искать в этой части исключение в части обработки было исключений. Он будет начинаться с первого предложения WHEN и последовательно будет выполнять поиск.
- Если он обнаружил обработку исключения для возникшего исключения, то он выполнит эту конкретную часть кода обработки.
- Если ни одно из условий «WHEN» не присутствует в исключении, которое сгенерировано, то механизм PL / SQL выполнит часть «WHEN OTHERS» (если присутствует). Это общее для всех исключений.
- После выполнения исключения управление деталями выйдет из текущего блока.

Oracle предопределил некоторые распространенные исключения. Эти исключения имеют уникальное имя исключения и номер ошибки. Эти **исключения уже определены** в пакете STANDARD в Oracle. В коде мы можем напрямую использовать эти предопределенные имена исключений для их обработки. Примеры:

Исключение	Код	Причина
ZERO_DIVIDE	ORA-01476	Разделив число на «0»
TOO_MANY_ROWS	ORA-01422	Когда оператор SELECT с предложением INTO возвращает более одной строки
NO_DATA_FOUND	ORA-01403	Когда оператор SELECT, содержащий предложение INTO, не извлекает строки.
INVALID_CURSOR	ORA-01001	Недопустимые операции с курсором, такие как закрытие неоткрытого курсора
CASE_NOT_FOUND	ORA-06592	Ни одно из условий «WHEN» в операторе CASE не выполнено, а предложение «ELSE» не указано
CURSOR_ALREADY_OPEN	ORA-06511	Попытка открыть курсор, который уже открыт

Поскольку в PL/SQL имена пользовательским исключениям автоматически не назначаются, вы должны делать это самостоятельно, определяя исключения в разделе объявлений блока PL/SQL. При этом задается имя исключения, за которым следует ключевое слово EXCEPTION:

```
имя_исключения EXCEPTION;
```

Чтобы программист имел возможность самостоятельно инициировать именованные исключения, в Oracle поддерживается команда **RAISE**. С ее помощью можно инициировать как собственные, так и системные исключения.

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

Блок, в котором может быть инициировано исключение, определяется правилами области действия исключений. В программе инициированное исключение распространяется в соответствии с определенными правилами.

Сначала PL/SQL ищет обработчик исключения в текущем блоке (анонимном блоке, процедуре или функции). Если такового нет, исключение передается в родительский блок. Затем PL/SQL пытается обработать исключение, инициировав его еще раз в родительском блоке. И так происходит в каждом внешнем по отношению к другому

блоке до тех пор, пока все они не будут исчерпаны. После этого PL/SQL возвращает необработанное исключение в среду приложения, выполнившего «самый внешний» блок PL/SQL. И только теперь исключение может прервать выполнение основной программы.

Структура процесса обработки локальных, определяемых программистом исключений в PL/SQL такова, что можно легко потерять информацию об исключении (то есть о том, какая именно произошла ошибка). Если исключение не обрабатывается в данном блоке, оно передается в родительский, где нет никакой информации о нем. Известно только то, что произошла ошибка, а какая именно — неизвестно. Ведь все пользовательские исключения имеют один и тот же номер ошибки 1 и одно и то же сообщение «User Defined Exception» — если только вы не воспользуетесь директивой EXCEPTION_INIT, чтобы связать с объявленным исключением другой номер, и не присвоите ему другое сообщение об ошибке при вызове RAISE_APPLICATION_ERROR.

Таким образом, локально объявленные (и инициированные) исключения всегда следует обрабатывать по имени.

Директива компилятора **EXCEPTION_INIT** (команда, выполняемая во время компиляции) связывает идентификатор, объявленный с ключевым словом EXCEPTION, с внутренним кодом ошибки.

```
PROCEDURE my_procedure
IS
   invalid_month EXCEPTION;
   PRAGMA EXCEPTION_INIT (invalid_month, -1843);
BEGIN
   ...
EXCEPTION
   WHEN invalid_month THEN
```

Для создания собственных сообщений об ошибках, более содержательных, чем именованные исключительные ситуации, пользователи могут применять встроенную функцию **RAISE_APPLICATION_ERROR**. Сообщения об ошибках, определяемых пользователем, передаются из блока в вызывающую среду так же, как и сообщения об ошибках Oracle. Преимущество перед командой RAISE (которая тоже может инициировать специфические для приложения явно объявленные исключения) заключается в том, что она позволяет связать с исключением сообщение об ошибке.

Синтаксис функции **RAISE_APPLICATION_ERROR(номер_ошибки, сообщение_об_ошибке, [сохранение_ошибки]);**

где номер_ошибки — это параметр, лежащий в диапазоне от -20 000 до -20 999, сообщение_об_ошибке - текст, соответствующий данной ошибке, а сохранение_ошибки — логическое значение. Длина параметра сообщение_об_ошибке не должна превышать 512 символов. Логический параметр сохранение_ошибки

необязателен. Если он установлен в TRUE, новая ошибка пополнит список ранее ошибок (при наличии этого в случае же FALSE новая ошибка заместит текущий список ошибок.

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

Сводка числовых функций PL/SQL

ABS(n). Возвращает абсолютное значение числа.

CEIL(n). Возвращает наименьшее целое число, которое больше либо равно заданному значению. Округление вниз.

ROUND(n). Возвращает значение n, округленное до ближайшего целого.

ROUND(n, m). Возвращает значение n, округленное до m разрядов. Значение m может быть отрицательным: в этом случае функция ROUND отсчитывает позиции округления влево, а не вправо от десятичной запятой.

TRUNC(n). Усекает значение n до целого числа. Например, результат вызова TRUNC(10.51) равен 10.

TRUNC(n, m). Усекает значение n до m разрядов. Например, результат вызова TRUNC(10.789, 2) равен 10.78. Значение m может быть отрицательным: в этом случае функция TRUNC отсчитывает позиции усечения влево, а не вправо от десятичной запятой.

FLOOR(n). Возвращает наибольшее целое число, которое меньше или равно заданному значению. Округление вверх.

REMAINDER(n, m). Возвращает «псевдоостаток» от деления n на m. Значение вычисляется по следующей формуле:

n - (m*ROUND(n/m))

MOD(n, m). Возвращает остаток от деления n на m. Остаток вычисляется по формуле, эквивалентной n-(m*FLOOR(n/m)) при совпадении знаков n и m или n-(m*CEIL(n/m)) при различающихся знаках. Например, результат вызова MOD(10, 2.8) равен 1.6. Если аргумент m равен 0, возвращается значение n.

BITAND(n, m). Выполняет поразрядную операцию AND над битами двух положительных целых чисел. Например, вызов BITAND(12,4) дает результат 4, то есть в значении 12 (двоичное 1100) установлен 4-й бит.

COS(n). Возвращает косинус угла n, заданного в радианах.

COSH(n). Возвращает гиперболический косинус n.

EXP(n). Возвращает число e в степени n, где n — аргумент функции.

ACOS(n). Возвращает арккосинус угла n из диапазона [–1; 1].

ASIN(n). Возвращает арксинус угла n из диапазона [–1; 1].

ATAN(n). Возвращает арктангенс угла n из диапазона (–∞; +∞).

SIN(n). Возвращает синус угла n, заданного в радианах.

SINH(n). Возвращает гиперболический синус n.

TAN(n). Возвращает тангенс угла n, заданного в радианах.

TANH(n). Возвращает гиперболический тангенс n.

POWER(n, m). Возводит n в степень m.

LOG(b, n). Возвращает логарифм заданного числа по указанному основанию. Значение аргумента n должно быть больше 0 или равно ему, а основание b должно быть больше 1.

SIGN(n). Возвращает −1, 0 или +1, если значение n меньше нуля, равно нулю или больше нуля соответственно.

GREATEST(n1, n2,...n3). Возвращает наибольшее число во входном списке; например, результат вызова GREATEST (1,0, -1, 20) равен 20.

Сводка символьных функций PL/SQL

ASCII(символ). Возвращает числовой код (NUMBER) представления заданного символа в наборе символов базы данных.

CHR(код). Возвращает символ типа VARCHAR2 (длина 1), соответствующий заданному коду. Функция является обратной по отношению к функции ASCII.

ASCIISTR(строка1). Получает строку в любом наборе символов и преобразует ее в строку ASCII-символов. Все символы, отсутствующие в кодировке ASCII, представляются в форме \XXXX, где XXXX — код символа в Юникоде.

UNISTR(строка1). Возвращает строку1, преобразованную в Юникод; таким образом, функция является обратной по отношению к ASCIISTR.

CONCAT(строка1, строка2). Присоединяет строку2 в конец строки1.

CONVERT(строка1, набор_символов). Преобразует строку из набора символов базы данных в заданный набор символов.

INITCAP(строка1). Изменяет регистр символов строкового аргумента, переводя первую букву каждого слова строки в верхний регистр, а остальные буквы — в нижний.

INSTR(строка1, строка2). Возвращает позицию, с которой строка2 входит в строку1; если вхождение не обнаружено, функция возвращает 0.

LENGTH(строка1). Возвращает количество символов в строке.

LOWER(строка1). Преобразует все буквы заданной строки в нижний регистр.

RTRIM(строка1). Удаляет пробелы с правого края строки1.

SUBSTR(строка1, начальная_позиция, длина). Возвращает подстроку из строки1, которая начинается с начальной_позиции и имеет заданную длину.

UPPER(строка1). Преобразует все буквы заданной строки в верхний регистр.

TRIM(FROM строка1). Возвращает строку, полученную в результате удаления из строки1 всех начальных и конечных пробелов.

LTRIM(строка1). Удаляет пробелы с левого края строки1.

Сводка функций для работы с датами и временем PL/SQL

Имя	Описание
ADD_MONTHS	Возвращает значение DATE, полученное в результате увеличения заданного значения DATE на заданное количество месяцев. `ADD_MONTHS('09-02-2000', 5)`
CURRENT_DATE	Возвращает текущую дату и время в часовом поясе сеанса как значение типа DATE
EXTRACT	Возвращает значение NUMBER или VARCHAR2, содержащее конкретный элемент даты/времени — час, год или сокращение часового пояса. `EXTRACT (YEAR FROM DATE '2019-08-22')`

LAST_DAY	Возвращает последний день месяца для заданного входного значения DATE
MONTHS_ BETWEEN	Возвращает значение NUMBER, содержащее количество месяцев между двумя датами. `MONTHS_BETWEEN('16-04-1973', '16-03-1997')`
NEXT_DAY	Возвращает дату первого дня недели, следующего за указанной датой `NEXT_DAY('07-04-2003', 'Понедельник')`

ROUND	Возвращает значение типа DATE, округленное до заданных единиц
	`ROUND(TO_DATE ('31.07.2014'),'MONTH')`
	`--Результат:  01.08.2014`

SYSDATE	Возвращает текущую дату и время сервера Oracle как значение типа DATE
SYSTIMESTAMP	Возвращает текущую дату и время сервера Oracle как значение типа TIMESTAMP WITH TIME ZONE

63. Коллекции. Массивы переменной длины. Вложенные таблицы. Ассоциативные массивы.

Коллекция – структура данных, содержащая элементы одного типа.

Необходимо объявить тип коллекции – **командой TYPE.**

Все элементы коллекции должны относиться к одному типу, то есть коллекция является **однородной**.

Коллекция называется **ограниченной**, если заранее определены границы возможных значений индексов (номеров) ее элементов. Если же верхняя или нижняя граница номеров элементов не указана, то коллекция называется **неограниченной**.

Коллекция называется **плотной**, если все ее элементы, от первого до последнего, определены и каждому из них присвоено некоторое значение (таковым может быть и NULL). Коллекция считается **разреженной**, если отдельные ее элементы отсутствуют. Не обязательно определять все элементы коллекции и заполнять ее полностью.

К любому элементу коллекции можно обратиться по номеру, который представляет собой **целочисленное значение**. В качестве индексов ассоциативных массивов можно использовать не только номера, но и **символьные строки**.

Ассоциативные массивы

Это **одномерные неограниченные разреженные** коллекции, состоящие из **однородных** элементов, доступные только в PL/SQL. Номера строк **не обязаны** быть последовательными.

```
TYPE type_assoc_arr IS TABLE OF element_type
              INDEX BY key_type;
var_type type_assoc_arr;
```

type_assoc_arr – имя типа Associative Arrays
element_type - любой тип данных PL/SQL, за исключением REF CURSOR
key_type тип индекса, может быть числовым: PLS_INTEGER или BINARY_INTEGER, это может быть также VARCHAR2 или один из его подтипов VARCHAR, STRING или LONG.
var_type - имя переменной типа Associative Arrays

Вложенные таблицы

Так называются **одномерные** несвязанные коллекции, также состоящие из **однородных** элементов. Первоначально они заполняются полностью, но позднее из-за удаления некоторых элементов могут стать **разреженными**. Вложенные таблицы могут определяться и в PL/SQL, и в базах данных (например, в качестве столбцов таблиц). Вложенные таблицы представляют собой мультимножества, то есть элементы вложенной таблицы **не упорядочены**.

```
TYPE nt_type IS TABLE OF element_type [NOT NULL];
var_nt nt_type;
```

nt_type - имя типа, используемое позже для объявления коллекций.
element_type - любой тип данных PL/SQL, за исключением: REF CURSOR
var_nt – переменная типа Nested Tables.

Массив типа VARRAY

Подобно двум другим типам коллекций, массивы VARRAY (массивы переменной длины) также являются **одномерными** коллекциями, состоящими из **однородных** элементов. Однако их размер всегда **ограничен**, и они **не бывают разреженными**. Как и вложенные таблицы, массивы VARRAY используются и в PL/SQL, и в базах данных. Однако порядок их элементов при сохранении и выборке, в отличие от вложенных таблиц, сохраняется.

```
TYPE type_varray IS {VARRAY | VARYING ARRAY} (size_limit)  OF
element_type [NOT NULL];
v_arr type_varray;
```

nt_type - имя типа, используемое позже для объявления коллекций.
element_type - любой тип данных PL/SQL, за исключением: REF CURSOR
var_nt – переменная типа Nested Tables

Характеристика	Ассоциативный массив	Вложенная таблица	Массив VARRAY
Размерность	Одномерная коллекция	Одномерная коллекция	Одномерная коллекция
Может использоваться в SQL?	Да	Да	Да
Может использоваться как тип данных столбца таблицы?	Нет	Да; данные хранятся в отдельной вспомогательной таблице	Да; данные хранятся в той же таблице
Неинициализированное состояние	Пустая коллекция (не может быть равна NULL); элементы не определены	Атомарное значение NULL; ссылки на элементы не-действительны	Атомарное значение NULL; ссылки на элементы не-действительны
Инициализация	Автоматическая при объявлении	При вызове конструктора, выборке или присваивании	При вызове конструктора, выборке или присваивании

Ссылка в элементах PL/SQL	BINARY_INTEGER и любые из его подтипов (-2 147 483 647 .. 2,147,483,647)	VARCHAR2 (Oracle9i Release2 и выше)	Положительное целое число от 1 до 2 147 483 647
Разреженная?	Да	Изначально нет, но может стать после удалений	Нет
Ограниченная?	Нет	Может расширяться	Да
Допускает присваивание любому элементу в любой момент времени?	Да	Нет; может потребоваться предварительный вызов EXTEND	Нет; может потребоваться предварительный вызов EXTEND, который не может выходить за верхнюю границу
Способ расширения	Присваивание значения элементу с новым индексом	Встроенная процедура EXTEND (или TRIM для сжатия) без заранее определенного максимума	EXTEND (или TRIM), но не более объяв-ленного максимального размера
Поддерживает проверку равенства?	Нет	Да, Oracle10g и выше	Нет
Поддерживает использование операций над множествами?	Нет	Да, Oracle10g и выше	Нет

Сохраняет порядок следования элементов и индексы при сохранении в базе данных?	–	Нет	Да

Встроенные методы коллекций:

Метод (функция или процедура)	Описание
COUNT (функция)	Возвращает текущее значение элементов в коллекции
DELETE (процедура)	Удаляет из коллекции один или несколько элементов. Уменьшает значение, возвращаемое функцией COUNT, если заданные элементы еще не удалены. Со структурами VARRAY может использоваться только для удаления всего содержимого
EXISTS (функция)	Возвращает значение TRUE или FALSE, определяющее, существует ли в коллекции заданный элемент
EXTEND (процедура)	Увеличивает количество элементов во вложенной таблице или VARRAY, а также значение, возвращаемое функцией COUNT
FIRST, LAST (функции)	Возвращают индексы первого (FIRST) и последнего (LAST) элемента в коллекции
LIMIT (функция)	Возвращает максимальное количество элементов в массиве VARRAY
PRIOR, NEXT (функции)	Возвращают индексы элементов, предшествующих заданному (PRIOR) и следующему за ним (NEXT). Всегда используйте PRIOR и NEXT для перебора коллекций, особенно при работе с разреженными (или потенциально разреженными) коллекциями
TRIM (функция)	Удаляет элементы, начиная с конца коллекции (элемент с наибольшим индексом)

тут можно посмотреть примеры использования

https://oracle-patches.com/db/sql/%D0%B7%D0%BD%D0%B0%D0%BA%D0%BE%D0%BC%D1%81%D1%82%D0%B2%D0%BE-%D1%81-%D0%BA%D0%BE%D0%BB%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%D0%BC%D0%B8-pl-sql

64. 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. Процедура хранится в базе данных, поэтому и называется хранимой.

Для создания процедуры используется оператор **CREATE PROCEDURE**.

Упрощенный синтаксис оператора CREATE PROCEDURE выглядит следующим образом.

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(имя_параметра [IN | OUT | IN OUT] тип [, …])]
IS | AS
BEGIN
тело процедуры
END имя_процедуры
```

OR REPLACE – если процедура с таким же именем уже хранится в базе данных, то с данным параметром, она просто заменит существующую. Без этого параметра, если процедура с данным именем уже существует, появится сообщение об ошибке.

IN | OUT | IN OUT – специфицирует режим параметров. Для каждого параметра можно выбрать один из следующих режимов:

- IN – режим по умолчанию для параметра. К моменту выполнения параметр должен иметь значение и это значение не изменится в результате выполнения процедуры. Входной параметр.
- OUT – специфицируется для параметров, значения которых устанавливаются только в теле процедуры. Выходной параметр.
- IN OUT - специфицируется для параметров, которые могут иметь значение к моменту вызова процедуры, но эти значения могут быть изменены в теле процедуры.

AUTHID – определяет, как будет выполняться модуль и разрешаться имена в БД:

DEFINER – (по умолчанию) от имени владельца модуля

CURRENT_USER - от имени пользователя, выполняющего модуль

```
  -- 16/07.sql
create or replace procedure svvcore.xxsum(
         min_cy  in      svvcore.auditorium.auditorium_capacity%type default 20,  -- минимальная в
         max_cy  in out svvcore.auditorium.auditorium_capacity%type,  -- максимальная вместимость
         n_aud   out     number                                       -- колличество
                                    )
    authid current user 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 xxsum;
```

Процедура вызывается как исполняемая команда PL/SQL. Другими словами, ее вызов должен начинаться с оператора EXECUTE (можно пренебречь данным оператором) и заканчиваться точкой с запятой (;). Если процедура не имеет параметров, она может вызываться с пустыми круглыми скобками или без них.

Позиционная форма передачи параметров – 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);
```

Параметрам IN можно задать **значения по умолчанию**. Если параметр IN имеет значение по умолчанию, включать этот параметр в вызов программы не обязательно. Значение по умолчанию параметра вычисляется и используется программой только в том случае, если параметр не был включен в список при вызове. Конечно, для всех параметров IN OUT фактические параметры должны включаться в список. Значение по умолчанию определяется для параметра так же, как для объявляемой переменной. Предусмотрены два способа задания значений по умолчанию — с ключевым словом DEFAULT и с оператором присваивания (:=):

```
CREATE OR REPLACE PROCEDURE TEST_POZ(
   PR_A IN NUMBER,
   PR_B IN NUMBER,
   PR_C IN VARCHAR2 := 'HELLO',
   PR_D IN VARCHAR2 DEFAULT 'WORLD!!!')
IS
BEGIN
```

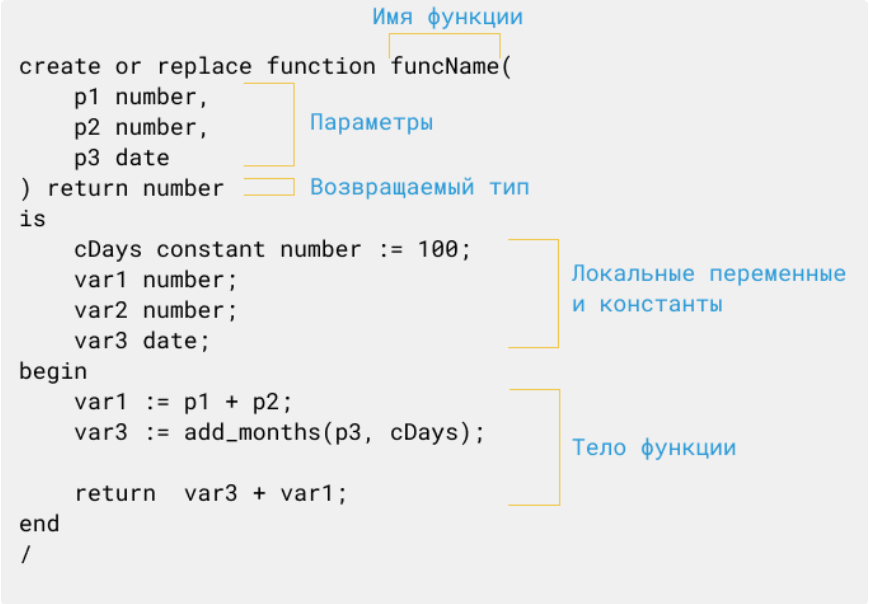
```
NULL;
END TEST_POZ;
```

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

Функция – именованный модуль, который выполняет ноль или более выражений через фразу Return.

Основные элементы которые нужно указать при создании функции это:

- Имя функции
- Параметры(могут отсутствовать)
- Тип возвращаемого значения
- Тело функции
- Команда Return



IN | OUT | IN OUT – специфицирует режим параметров. Для каждого параметра можно выбрать один из следующих режимов:

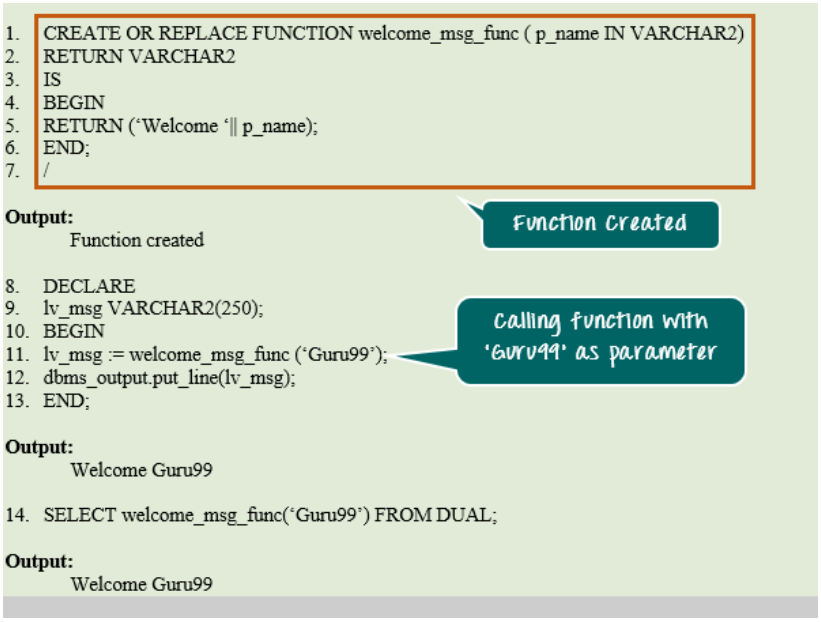
- IN – режим по умолчанию для параметра. К моменту выполнения параметр должен иметь значение и это значение не изменится в результате выполнения. Входной параметр.
- OUT – специфицируется для параметров, значения которых устанавливаются только в теле. Выходной параметр.
- IN OUT - специфицируется для параметров, которые могут иметь значение к моменту вызова, но эти значения могут быть изменены в теле.

Функция использует ключевое слово **RETURN** для возврата значения, а тип данных обязательно определяется во время создания. Функция PLSQL также может возвращать значение через параметры OUT, кроме использования RETURN.

Функция без операторов DML может быть вызвана напрямую в запросе SELECT, тогда как функция с DML operation можно вызывать только из других блоков PL/SQL.

Поскольку функция всегда возвращает значение, в операторе вызова он всегда сопровождается присваиванием.

Создание функции и ее вызов с использованием анонимного блока



Функция называется **детерминированной**, если при одном наборе параметров IN и IN OUT она всегда возвращает одно и то же значение.

Если при вызове функции с передачей одинакового набора параметров, она всегда будет возвращать одинаковый результат, то почему бы Oracle не сохранить результат, связанный с конкретным набором аргументов? Чтобы добиться подобного эффекта, включите предложение **DETERMINISTIC** в заголовок функции:

```
  -- l6/l3 .sql
create or replace function maximumcy_d
  return number deterministic  is
     rc  number(5);
  begin
     select max(auditorium_capacity) into rc from svvcore.auditorium;
     return rc;
  exception
     when others then return -1;
end maximumcy_d;
```

Решение об использовании сохраненной копии возвращаемого результата принимается оптимизатором запросов Oracle. Сохраненные копии могут браться из материализованного представления, функционального индекса или повторного вызова одной функции в команде SQL. Детерминированная функция может улучшить производительность команд SQL, вызывающих такие функции.

Pipelined функция или конвейерная функция – это функция, которая возвращает набор данных постепенно, строка за строкой, не дожидаясь завершения обработки всех данных. Обычные функции в Oracle обрабатывают все данные перед возвратом результата. Pipelined функции, напротив, могут начать возвращать данные до завершения обработки всего набора.

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 в примере вызов RETURN пустой, поскольку нет никакой коллекции, возвращаемой из 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;
```

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

Пакет PL/SQL представляет собой сгруппированный именованный набор элементов кода PL/SQL.

Пакет состоит из двух частей — **спецификации** и **тела**. Спецификация является обязательной частью и определяет, как разработчик может использовать пакет: какие программы можно вызывать, какие курсоры открывать и т. д. Тело пакета — необязательная, но почти всегда присутствующая часть; она содержит код перечисленных в спецификации программ (и возможно, курсоров), а также другие необходимые элементы кода.

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

Состоянием пакета (package state) называется совокупность текущих значений его глобальных и локальных переменных и констант, а также текущих состояний курсоров (курсор открыт или закрыт, а также значения атрибутов курсора, отражающие в том числе сколько строк было считано из открытого курсора к настоящему времени).

Состояния пакетов сохраняется в течение всей жизни сессии пользователя. Очень важно то, что состояния пакетов у каждой сессии пользователя свое. Нельзя «подсмотреть» или «подправить» значения пакетных переменных или считать строку курсора из состояния пакета другой сессии. Во многих приложениях на этом построены специфические системы управления доступом.

```
CREATE OR REPLACE PACKAGE emp_actions AS  -- спецификация
   TYPE EmpRecTyp IS RECORD (emp_id INT, salary REAL); -- тип запись
   CURSOR desc_salary RETURN EmpRecTyp;                -- курсор
   /*процедура приема на работу*/
   PROCEDURE hire_employee (
      ename  VARCHAR2,
      job    VARCHAR2,
      mgr    NUMBER,
      sal    NUMBER,
      comm   NUMBER,
      deptno NUMBER);
       /*процедура увольнения*/
   PROCEDURE fire_employee (emp_id NUMBER);
END emp_actions;

CREATE OR REPLACE PACKAGE BODY emp_actions AS  -- тело
   CURSOR desc_salary RETURN EmpRecTyp IS
      SELECT empno, sal FROM emp ORDER BY sal DESC;
   PROCEDURE hire_employee (
      ename  VARCHAR2,
      job    VARCHAR2,
      mgr    NUMBER,
      sal    NUMBER,
      comm   NUMBER,
      deptno NUMBER) IS
   BEGIN
      INSERT INTO emp VALUES (empno_seq.NEXTVAL, ename, job,
       mgr, SYSDATE, sal, comm, deptno);
   END hire_employee;

   PROCEDURE fire_employee (emp_id NUMBER) IS
   BEGIN
      DELETE FROM emp WHERE empno = emp_id;
   END fire_employee;
END emp_actions;
```

67. 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, которые срабатывают по запускающему их событию.

По привязанному объекту:
На таблице

На представлении - instead of trigger

По событиям запуска:
DML-команда (DELETE, INSERT или UPDATE).

DDL-команды (CREATE, ALTER или DROP).

Операции базы данных, такие как SERVERERROR, LOGON, LOGOFF, STARTUP или SHUTDOWN.

По области действия:
Уровень оператора - statement level triggers

Уровень записи - row level triggers

Составные триггеры - compound triggers

По времени срабатывания:
Перед выполнением операции – before

После выполнения операции - after

Привилегии:

CREATE TRIGGER - создавать, удалять, изменять в своей подсхеме

CREATE ANY TRIGGER - создать любой триггер в любой схеме, кроме SYS, не рекомендуется для словаря, не разрешает менять текст триггера

ALTER ANY TRIGGER - разрешать, запрещать, изменять, компилировать, любые, кроме SYS-триггеров, триггеры

DROP ANY TRIGGER - удалять любой триггер, кроме SYS-триггеров

ADMINISTER DATABASE TRIGGER - создавать, изменять, удалять системные триггеры, должен иметь привилегию CREATE TRIGGER или CREATE ANY TRIGGER

Порядок выполнения:

операторные BEFORE;

для каждой строки BEFORE;

выполняется оператор;

для каждой строки AFTER;

операторные AFTER.

В теле триггера можно использовать **предикаты** (если триггер состоит из нескольких DML-операций):

- IF DELETING,
- IF UPDATING,
- IF INSERTING,

чтобы определить, какое событие вызвало триггер, который слушает несколько событий.

NEW и OLD (доступны только в строковых триггерах):

DML-команда	OLD	NEW
INSERT	NULL	введённое значение
UPDATE	значение до обновления	значение после обновления
DELETE	удаляемое значение	NULL

Триггеры замещения (instead of) можно создавать **только для представлений**. В отличие от триггеров DML, которые выполняются в дополнение к операторам DML, триггеры замещения выполняются вместо операторов DML, вызывающих их срабатывание. Триггеры замещения должны быть **строковыми триггерами**.

Отключенный триггер не запускается при наступлении связанного с ним события. Удаленный триггер полностью исчезает из базы данных. Синтаксис **отключения триггера**:

```
ALTER TRIGGER имя_триггера DISABLE;

ALTER TRIGGER имя_триггера ENABLE; – включение
```

В Oracle появилась возможность создания триггеров в отключенном состоянии:

```
TRIGGER just_testing
AFTER INSERT ON abc
DISABLE
BEGIN
   NULL;
END;
```

68. Секционирование таблиц. Виды секционирования.

Секционирование (partitioning) — это способ логического разделения крупной таблицы на более мелкие части для облегчения обработки запросов, операций DML и управления базой данных. Секционирование приводит к повышению производительности потому, что базе данных в ответ на запрос приходится выполнять поиск только в определенных разделах таблицы.

Секционирование по диапазонам ключей (ключи - стобцы секционирования)

Диапазонное секционирование используется для **данных, которые разделяются на диапазоны на основе некоторого критерия**.

```
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
);
```

При секционировании по диапазонам принято последним устанавливать раздел “остальные” (catchall). Когда такой раздел создается, он содержит значения, меньшие максимального (**maxvalue**), которое просто является значением, большим значений в предпоследнем разделе.

Интервальное секционирование

Интервальное секционирование — это расширение традиционного метода секционирования по диапазону ключей. Чтобы реализовать интервальное секционирование, сначала потребуется специфицировать минимум один диапазонный раздел таблицы. Используете вы минимальный однодиапазонный раздел или многодиапазонные разделы, максимальное значение ключа диапазонного секционирования называется **точкой перехода** (transition point). После того, как данные пересекают точку перехода, база данных автоматически создает интервальные разделы.

Ниже перечислено, что следует знать об интервальном секционировании.

- Чтобы создать таблицу, секционированную по интервалам, укажите в операторе CREATE TABLE **конструкцию INTERVAL**.
- Прежде чем специфицировать интервальные разделы, определите, как минимум, один диапазонный раздел с помощью **конструкции PARTITION.**
- Можно использовать ключ секционирования, который включает более одного столбца.
- Ключ секционирования должен быть типа **NUMBER или DATE**.
- С помощью **конструкции STORE IN** в операторе CREATE TABLE можно дополнительно специфицировать табличные пространства для секционирования данных.

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

Хеш-секционирование

Все, что потребуется сделать — выбрать количество разделов, и алгоритм хеширования Oracle назначит хеш-значение ключу раздела каждой строки, после чего поместит ее в соответствующий раздел. Вам ничего не нужно знать о распределении данных в таблице,помимо того, что данные не должны попадать в один, легко определяемый диапазон. Все, что необходимо предоставить — это ключ раздела:

```
CREATE TABLE DEACore.salesH
(
  prod_id NUMBER,
  time_id DATE
)
PARTITION BY HASH (prod_id)
PARTITIONS 16
STORE IN (data00, data01, data02, data03);
```

В приведенном примере создаются 16 хеш-разделов в четырех табличных пространствах. Мы не будем знать, в каком разделе находятся данные, скажем, за 10 июня 2008 г. Oracle определит место хранения на основе алгоритма хеширования, и вы никак не управляете отображением строк на разделы.

Секционирование по списку значений ключа

Секционирование по списку значений ключа, или списковое секционирование, —предпочтительный способ по отношению к диапазонному или хеш-секционированию,когда данные распределены среди множества **дискретных значений**. Например, может потребоваться группирование данных о продажах компании по регионам, а не по кварталам.

```
CREATE TABLE DEACore.salesL
(
  prod_id NUMBER,
  time_id DATE,
  group_id NUMBER
)
PARTITION BY LIST (group_id)
(
  PARTITION sales_l1 VALUES(3, 9),
  PARTITION sales_l2 VALUES(5, 4),
  PARTITION sales_other VALUES (DEFAULT)
);
```

Ссылочное секционирование

Если две таблицы связаны друг с другом, можно воспользоваться преимуществом этого отношения, выполнив секционирование этих двух таблиц на основе **существующего отношения "родительский–дочерний".** Это отношение задается ограничениями первичного и внешнего ключа. Если две таблицы разделяют отношение "родительский–дочерний", нужно лишь **формально секционировать родительскую таблицу**. Тем самым **исключается дублирование ключевых столбцов**. Любые операции обслуживания разделов на родительской таблице автоматически распространятся на дочернюю таблицу.

Ниже приведен простой пример для прояснения концепции ссылочного секционирования. Таблицы SalesR и SalesR_details связаны друг с другом на основе столбца fact_id из обеих таблиц. Это отношение зафиксировано ссылочным ограничением sales_details_fk. Родительская таблица — SalesR — секционирована по столбцу time_id с использованием схемы диапазонного секционирования.

```
CREATE TABLE DEACore.salesR
(
  prod_id NUMBER,
  time_id DATE,
  fact_id NUMBER CONSTRAINT sales_pk PRIMARY KEY
)
PARTITION BY RANGE (time_id)
(
  PARTITION sales_1lq1 VALUES LESS THAN ('01-APR-2011'),
  PARTITION sales_1lq2 VALUES LESS THAN ('01-JUL-2011'),
  PARTITION sales_max VALUES LESS THAN (MAXVALUE)
);

CREATE TABLE DEACore.salesR_details
(
  details_id NUMBER,
  fact_id NUMBER NOT NULL
  CONSTRAINT sales_details_fk REFERENCES DEACore.salesR(fact_id)
)
PARTITION BY REFERENCE (sales_details_fk);
```

В данном случае дочерняя таблица SalesR_details наследует секционирование от родительской таблицы через внешнее ключевое ограничение sales_details_fk. Таблица SalesR_details разбивается в соответствии с секционированием по ссылке на родительскую таблицу.

Вместе со ссылочным секционированием можно использовать любую стратегию секционирования, за исключением секционирования по интервалам.

При создании таблицы со ссылочным секционированием конструкция **partition by reference** в операторе CREATE TABLE специфицирует имя ссылочного ограничения, служащего основой для ссылочного секционирования. Необходимо гарантировать, что соответствующее ссылочное ограничение является действительным и включенным.

Системное секционирование

Системное секционирование — уникальный метод секционирования таблиц, при котором расположением данных управляет приложение, а не база данных. База данных просто позволяет разделить таблицу на разделы, не имея никакого представления относительно того, что будет содержать каждый из разделов. Приложение управляет тем,что попадает в каждый раздел. При вставке данных в таблицу с системным секционированием **необходимо явно специфицировать раздел**. Таким образом, если попытаться вставить данные в таблицу с системным секционированием, не указывая раздела, в который эти данные следует поместить, то вставка закончится неудачей.

```
CREATE TABLE test (c1 integer, c2 integer)
PARTITION BY SYSTEM
(
PARTITION p1 TABLESPACE tbs_1,
PARTITION p2 TABLESPACE tbs_2,
PARTITION p3 TABLESPACE tbs_3,
PARTITION p4 TABLESPACE tbs_4
);
```

При вставке данных с применением оператора INSERT или MERGE необходимо указать раздел, в который должна быть помещена новая строка. Ниже приведен пример вставки в таблицу с системным секционированием.

```
INSERT INTO test PARTITION (p1) VALUES (4,5);
```

Составное секционирование

Иногда простого секционирования по диапазону, хешу или списку оказывается недостаточно. Для более тонкого контроля над размещением данных можно выполнить дальнейшее секционирование крупной таблицы на подразделы. Oracle предлагает несколько типов составного секционирования.

- диапазон - хеш
- диапазон - список
- интервал - список

- интервал - диапазон

Секционирование по виртуальному столбцу

Для создания виртуального столбца можно использовать один или более реальных столбцов таблицы. Допускается секционировать таблицу в соответствии со значениями виртуального столбца. Это означает возможность секционирования таблицы по ключу секционирования, который на самом деле в таблице не существует. Такой ключ секционирования определяется тем же выражением, которое база данных использует для виртуального столбца.

Вместе с секционированием на основе виртуальных столбцов таблицы можно применять все базовые стратегии секционирования, включая разнообразные комбинации составного секционирования.

В следующем примере таблица секционируется с использованием виртуального столбца в качестве ключа подраздела. Виртуальный столбец TotalRevenue определен как произведение значений столбцов AverageRevenue и TotalProduct.

```
CREATE TABLE DEACore.salesR
(
  prod_id NUMBER,
  AverageRevenue NUMBER(10, 2),
  TotalProduct NUMBER(10, 2),
  TotalRevenue AS (AverageRevenue*TotalProduct)
)
PARTITION BY RANGE (TotalRevenue)
(
  PARTITION TR1 VALUES LESS THAN (1000) TABLESPACE data00,
  PARTITION TR2 VALUES LESS THAN (5000) TABLESPACE data01,
  PARTITION TR3 VALUES LESS THAN (10000) TABLESPACE data02,
  PARTITION TR_max VALUES LESS THAN (maxvalue) TABLESPACE data03
);
```

69. Транзакции в СУБД Oracle. Виды транзакций. Понятие автономной транзакции.

Транзакцией (transaction) называется логическая единица работы, состоящая из одного или более операторов SQL. Любая реляционная база данных должна удовлетворять тесту ACID: должны быть гарантированы **атомарность** (Выполняются или все изменения данных в транзакции или ни одна), **согласованность** (Выполняемые транзакцией трансформации данных переводят базу данных из одного согласованного состояния в другое), **изолированность** (Незаконченная должна быть невидима для остального мира) и **долговечность** (Транзакцию после фиксации нельзя отменить, кроме как другой транзакцией).

Сессия **начинает транзакцию** в момент когда она выполняет любую DML команду. Транзакция продолжается сколько угодно следующих DML команд, пока сессия не

выполнит команду **ROLLBACK** или **COMMIT**. Только подтверждённые изменения станут гарантированными и будут доступны для других сессий. **Невозможно начать транзакцию внутри транзакции**. Стандарт SQL не разрешает пользователям начать транзакцию, а затем начать новую перед завершение первой. Это можно сделать используя PL/SQL (язык Oracle третьего поколения), но не стандартным SQL.

Командами управления транзакциями являются команды COMMIT, ROLLBACK и SAVEPOINT. Также могут возникнуть другие обстоятельства помимо явного вызовая команды COMMIT или ROLLBACK которые **немедленно прекращают транзакцию**

- Выполнение DDL или DCL команды
- Завершение пользовательского процесса (к примеру пользователь вышел из программы SQL *Plus или SQL Developer)
- Клиентская сессия «умерла»
- Проблемы в системе

Если пользователь выполняет DDL команду (CREATE, ALTER или DROP) иди DCL команду (GRANT или REVOKE) то активная транзакция будет подтверждена. Так происходит потому что команды DDL и DCL сами являются транзакциями. Так как в SQL невозможно создать вложенные транзакции, если у пользователя уже выполнялась какая либо транзакция, все команды пользователя будут подтверждены вместе с командой DDL или DCL.

Синтаксис для отмены транзакции

ROLLBACK [TO SAVEPOINT savepoint];

Точка сохранения позволяет программистам устанавливать флаг в транзакции которые затем можно использовать для контроля эффекта отмены транзакции. Вместо отмены всей транзакции и её завершения, становится возможным отменить изменения сделанные после конкретного флага но оставить изменения сделанные до этого флага. Действие транзакции в этот момент продолжается: транзакция не подтверждена, всё ещё можно отменить транзакцию целиком и изменения не видны для других сессий.

Синтаксис команды

SAVEPOINT savepoint

Последняя команда для управления транзакциями это **SELECT FOR UPDATE**: чтение данных не блокирует запись, запись не блокирует изменение.

Директива FOR UPDATE приведёт к блокировке таблиц которые возвращает запрос. Другие сессии не смогут изменить данные и таким образом последующие изменения будут успешны. Блокировка строк вызванная командой FOR UPDATE будет длиться пока сессия не выполнит команду COMMIT или ROLLBACK.

- **SERIALIZABLE (сериализуемый)**:
    - Полная изоляция транзакций, как если бы они выполнялись последовательно.
    - Вставки, удаления и обновления блокируют записи до фиксации или отката.
    - Исключаются грязные данные, невоспроизводимые чтения и фантомные данные.
- **REPEATABLE READ (повторяемое чтение)**:
    - Гарантирует, что повторное чтение данных вернет те же значения.
    - Исключает грязные и невоспроизводимые чтения.
- **READ UNCOMMITTED (чтение незафиксированных данных)**:
    - Позволяет читать промежуточные данные других транзакций.
    - Приводит ко всем проблемам параллельного использования.
- **READ COMMITTED (чтение зафиксированных данных)**:
    - Уровень изоляции по умолчанию в Oracle.
    - Запросы видят только зафиксированные данные на момент начала запроса.
    - Гарантирует неизменность данных строки во время обращения к ней.

Уровень изоляции	Грязное чтение	Невоспроизводимое чтение	Фантомное чтение
READ UNCOMMITTED	Да	Да	Да
READ COMMITTED	Нет	Да	Да
REPEATABLE READ	Нет	Нет	Да
SERIALIZABLE	Нет	Нет	Нет

```
SQL> ALTER SESSION SET ISOLATION LEVEL SERIALIZABLE;
```

Автономная транзакция — это отдельная транзакция, которая выполняется независимо от основной транзакции. Она может быть зафиксирована или откатана без влияния на основную транзакцию.

Определить блок PL/SQL как автономную транзакцию очень просто. Для этого достаточно включить в раздел объявлений (в любом месте) следующую директиву:

```
PRAGMA AUTONOMOUS_TRANSACTION;
```

Директива приказывает компилятору PL/SQL назначить блок PL/SQL как автономный (независимый). В контексте автономных транзакций автономным может быть объявлен любой из следующих блоков:

- Анонимный блок PL/SQL верхнего уровня (не вложенный!).
- Функция или процедура, определенная в пакете или как отдельная программа.
- Метод (функция или процедура) объектного типа.
- Триггер базы данных.

- Чтобы выход из программы с автономной транзакцией, выполнившей как минимум одну команду INSERT, UPDATE, MERGE или DELETE, произошел без ошибок, **необходимо выполнить явную фиксацию или откат транзакции**. Если в программе содержится незавершенная транзакция, то исполняющее ядро инициирует исключение и производит откат незафиксированной транзакции.
- Команды COMMIT и ROLLBACK **завершают активную автономную транзакцию**, но не программу, в которой она содержится.
- При выполнении отката до точки сохранения эта **точка должна быть определена в текущей транзакции**. В частности, нельзя выполнить откат из автономной транзакции до точки сохранения, определенной в главной транзакции.
- **Параметр TRANSACTIONS** в инициализационном файле Oracle определяет допустимое количество выполняемых одновременно автономных транзакций для одного сеанса.

После выполнения в автономной транзакции команд COMMIT или ROLLBACK изменения по умолчанию немедленно становятся видимыми в главной транзакции. А если вы предпочитаете скрыть их от последней? Для этой цели используется следующая модификация команды **SET TRANSACTION**:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

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

В Oracle существует возможность запланировать выполнение определенного набора действий в виде **заданий**. Задание может, представляет собой хранимую процедуру, анонимный блок PL /SQL, внешнюю процедуру на языке C или Java. Время выполнения может иметь значение любого времени суток и подчиняться заданному интервалу.

Для того чтобы задания начались выполняться необходимо, установить параметр инициализации **JOB_QUEUE_PROCESSES**. Изначально он имеет значение 0 и задаёт максимальное количество фоновых процессов для выполнения заданий.

Существует процесс, который является координатором заданий. начала он выбирает таблицу SYS.JOB$, в которой хранятся параметры заданий. Если среди заданий имеются те, которые будут выполняться в ближайший интервал времени указанный в скрытом параметре JOB_QUEUE_INTERVAL (по умолчанию его значение составляет 5 секунд), то для них порождаются фоновые процессы очереди заданий, которые в свою очередь создают сеансы для непосредственного выполнения запланированных действий.

Для управления заданиями в Oracle существует **специальный пакет DBMS_JOB**. С его помощью над ними можно осуществлять различные действия. Для начала попробуем создать новое задание. В нашем случае для этого, нужно применить следующую процедуру пакета:

```
DBMS_JOB.SUBMIT(
   JOB       OUT BINARY_INTEGER,
   WHAT      IN  VARCHAR2,
   NEXT_DATE IN  DATE DEFAULT SYSDATE,
   INTERVAL  IN  VARCHAR2 DEFAULT NULL,
   NO_PARSE  IN  BOOLEAN DEFAULT FALSE,
   INSTANCE  IN  BINARY_INTEGER DEFAULT any_instance,
   FORCE     IN  BOOLEAN DEFAULT FALSE);
```

Опишем параметры этой процедуры:

- JOB - это **идентификатор задания**. Имеет уникальное значение для каждого задания, генерируемое системной последовательностью.
- WHAT - **тело задания**. Представляет собой анонимный PL/SQL блок. Всё что здесь указано, будет выполнено в процессе работы задания. Если вы запускаете только одну процедуру, то можно не заключать её в блок достаточно поставить в конце названия процедуры точку с запятой. Значение WHAT в этом случае автоматически будет помещено в PL/SQL блок. Если процедура имеет строковые параметры, то они обязательно должны заключаться в две одинарные кавычки с каждой стороны. В PL/SQL блоке

можно также писать DML и DDL команды, но нельзя производить создание и запуск заданий.

- NEXT_DATE - **дата следующего выполнения задания**. Время непосредственно задаётся владельцем или автоматически вычисляется Oracle.
- INTERVAL - **формула интервала времени**. Представляет собой DATE функцию. Именно от её правильного значения будет зависеть дата следующего выполнения задания указанного в столбце NEXT_DATE.
- NO_PARSE - **флаг разбора PL/SQL**. Если его значение равно TRUE, то тело задания будет синтаксически проверено при первом выполнении. Если FALSE, то тело задания будет синтаксически проверено в момент создания задания.
- INSTANCE - какой экземпляр производит **выполнение задания**.
- FORCE - указывает, следует ли принудительно создавать задание даже при наличии конфликтов или проблем с созданием.

Теперь попробуем изменить некоторые его параметры. **Для изменения** доступны следующие параметры задания: WHAT, NEXT_DATE, INTERVAL и INSTANCE. Их можно менять все одновременно или по отдельности.

CHANGE - изменение параметров задания

```
DBMS_JOB.CHANGE (
   JOB       IN  BINARY_INTEGER,
   WHAT      IN  VARCHAR2,
   NEXT_DATE IN  DATE,
   INTERVAL  IN  VARCHAR2,
   INSTANCE  IN  BINARY_INTEGER DEFAULT NULL,
   FORCE     IN  BOOLEAN DEFAULT FALSE);
```

WHAT - изменение тела задания

```
DBMS_JOB.WHAT (
   JOB       IN  BINARY_INTEGER,
   WHAT      IN  VARCHAR2);
```

NEXT_DATE - изменение даты следующего выполнения задания
INTERVAL - изменение формулы интервала времени
INSTANCE - изменение экземпляра, который производит выполнение задания

REMOVE - удаление задания

```
DBMS_JOB.REMOVE (
   JOB       IN  BINARY_INTEGER );
```

BROKEN - выключение задания

```
DBMS_JOB.BROKEN (
   JOB       IN  BINARY_INTEGER,
   BROKEN    IN  BOOLEAN,
   NEXT_DATE IN  DATE DEFAULT SYSDATE);
```

Если флаг BROKEN имеет значение истинно, то такое задание считается выключенным и выполняться не будет. Параметр NEXT_DATE определяет здесь дату следующего выполнения задания и действует только при его включении.

RUN - принудительное выполнение задания

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. Oracle Scheduler позволяет использовать сценарии PL/SQL, сценарии оболочки операционной системы, программы Java, и родные двоичные исполняемые файлы для выполнения запланированных заданий.

Основные компоненты DBMS_SCHEDULER

1. Jobs (задания):
    - Представляют собой задачи, которые должны быть выполнены. Они могут запускать PL/SQL блоки, анонимные PL/SQL блоки, внешние скрипты или приложения.

```
BEGIN
   DBMS_SCHEDULER.create_job (
      job_name        => 'my_job',
      job_type        => 'PLSQL_BLOCK',
      job_action      => 'BEGIN my_procedure; END;',
      start_date      => SYSTIMESTAMP,
      repeat_interval => 'FREQ=DAILY; BYHOUR=14; BYMINUTE=0;
BYSECOND=0',
      enabled         => TRUE
   );
END;
```

1. **job_name**: (обязательный) Имя задания.
2. **job_type**: (обязательный) Тип задания.
    - Возможные значения: `'PLSQL_BLOCK'`, `'STORED_PROCEDURE'`, `'EXECUTABLE'`

3. **job_action**: (обязательный) Действие, которое должно быть выполнено.
4. **start_date**: (необязательный) Дата и время начала выполнения задания.
5. **repeat_interval**: (необязательный) Интервал повторения задания.
6. **enabled**: (необязательный) Флаг, указывающий, должно ли задание быть включено при создании.

2. Schedule (расписание):

Создание расписания (schedule) с помощью DBMS_SCHEDULER.create_schedule в Oracle позволяет определить временные интервалы и условия, когда задания должны выполняться. Расписание само по себе не выполняет никаких действий; оно только описывает временные шаблоны, которые затем могут быть назначены заданиям.

```
BEGIN
   DBMS_SCHEDULER.create_schedule (
      schedule_name   => 'daily_schedule',
      start_date      => SYSTIMESTAMP,
      repeat_interval => 'FREQ=DAILY; BYHOUR=14; BYMINUTE=0;
BYSECOND=0',
      end_date        => NULL,
      comments        => 'Daily at 2 PM'
   );
END;
```

1. **schedule_name**: Уникальное имя расписания, которое будет использоваться для ссылки на это расписание при назначении его заданиям.
2. **start_date**: Дата и время, когда расписание должно начать действовать. Если не указано, расписание начинает действовать немедленно.
3. **repeat_interval**: Интервал повторения, определяющий, как часто и когда расписание должно срабатывать.
4. **end_date**: Дата и время окончания действия расписания. После этой даты расписание больше не будет срабатывать.
5. **comments**: Описание или комментарии к расписанию.

PS комменты и end date можно оставлять и к jobs мне лень их в код дописывать
После того, как мы создали задание и расписание, мы можем назначить заданию это расписание)

```
   DBMS_SCHEDULER.set_attribute (
      name       => 'my_job', -- имя задания
      attribute  => 'schedule_name',
      value      => 'daily_2pm_schedule'-- имя расписания
   );
```

3. Программы

Программы (`programs`) в Oracle DBMS_SCHEDULER — это логические объекты, которые определяют, что должно быть выполнено при запуске задания. Программы позволяют вам многократно использовать одну и ту же бизнес-логику для различных заданий, что упрощает управление и администрирование.

```
BEGIN
   DBMS_SCHEDULER.create_program (
      program_name   => 'my_program',
      program_type   => 'PLSQL_BLOCK',
      program_action => 'BEGIN my_procedure; END;',
      number_of_arguments => 0,
      enabled        => TRUE,
      comments       => 'Program to execute my_procedure'
   );
END;
```

program_name: (обязательный) Имя программы.
program_type: (обязательный) Тип программы.

- Возможные значения: `'PLSQL_BLOCK'`, `'STORED_PROCEDURE'`, `'EXECUTABLE'`

program_action: (обязательный) Действие, которое должно быть выполнено.
number_of_arguments: (необязательный) Количество аргументов, передаваемых в программу.
enabled: (необязательный) Флаг, указывающий, должна ли программа быть включена при создании.
comments: (необязательный) Комментарии к программе.

А теперь прикол: мы создали программу, создали расписание, и можно создать жопу используя их:

```
BEGIN
   DBMS_SCHEDULER.create_job (
      job_name        => 'my_job_using_program',
      program_name    => 'my_program',
      schedule_name   => 'daily_2pm_schedule',
      enabled         => TRUE,
      comments        => 'Job to run my_program daily at 2 PM'
   );
END;
```