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

Дистрибутивы СУБД Oracle:

* Oracle Database 12c
* Oracle Database 19c
* Oracle Database 21c

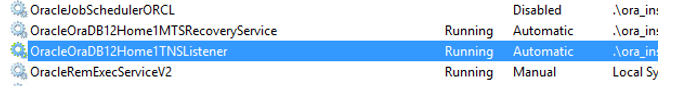
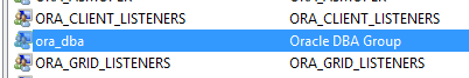
Global database name - уникальное имя установленной базы данных. Определяет базу данных в рамках одной сети. Состоит из имени и домена: sales.us.example.com, где sales.us - имя базы данных, а - example.com домен. Можно настроить в момент установки или в Oracle Database Configuration Assistant

SID - системный идентификатор базы данных. Уникальное имя ИНСТАНСА БД. Используется хешированным вместе с ORACLE\_HOME для создания ключа подключения SGA (system global area)

**Установка СУБД Oracle 12c на 64 битную Windows (установка только на англ версию винды, чтобы не было проблем):**

1. Открыть инсталлер
2. Пропустить вкладки Configure Security Updates и Software Updates
3. Во вкладке (3) Installation Options выбрать Create and Configure Database
4. Во вкладке (4) System Class выбрать Server Class
5. Во вкладке (5) Grid Installation Options выбрать Single Instance Database Installation
6. Во вкладке (6) Install Type выбрать Typical Install, т.е. базовая конфигурация
7. Во вкладке Oracle Home User Selection выбрать Create New Windows User и заполнить поля User Name, Password и подтвердить пароль
8. Во вкладке Typical Installation заполнить поля Oracle base (указывается путь в папку созданного в п.7 юзера), Software location (путь к dbhome\_1 ), поле Database file location (), в чекбоксе Database Edition выбрать Enterprise Edition, заполнить поле Global database name , поле пароля администратора, отметить чекбокс Create as Container database 
9. Ждем установку
10. Когда появится сообщение об изменении паролей пользователей SYS и SYSTEM не пропускаем, а изменяем им пароли. Однако можно оставить дефолтными: SYSTEM – MANAGER, SYS – CHANGE\_ON\_INSTALL, SCOTT – TIGER

**Проверка установки:**

* Сервисы 
* список установленных программ
* редактор реестра (win + r -> regedit -> HKEY\_LOCAL\_MACHINE -> SOFTWARE -> ORACLE -> KEY\_OraDB12Home1 или нечто подобное)
* установщик Universal Installer (who) -- ну тот которым бд ставили по идее
* пользователи и группы 
* соединение с бд (в SQL plus и SQL Developer)

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

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

**Основные системные 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.
* Роль **DBA** не включает **SYSDBA** и **SYSOPER**. Привилегии могут быть указаны при подключении (connect) пользователя к БД.

—---------------------- я не знаю, нужно ли это, но может быть полезно

### Системные права SYSDBA и SYSOPER SYSASM

Существуют два мощных набора административных прав — *SYSDBA* и *SYSOPER*. В связи с огромными возможностями, предоставляемыми этими полномочиями, на управление ими наложены определенные ограничения. Для назначения этих ролей нельзя использовать опцию *WITH ADMIN OPTION.* Только пользователь, подключенный в качестве *SYSDBA*, может выдавать (или отзывать) эти права другим пользователям. Кроме того, эти системные права нельзя предоставлять какой-то роли.

Системные права *SYSDBA* включают в себя полномочия *RESTRICTED SESSION* и содержат все системные права, помеченные опцией *WITH ADMIN OPTION*, в том числе, системные права *SYSOPER*. Полномочия *SYSDBA* позволяет решать следующие задачи.

* Выполнять операции *STARTUP* и *STDOWN*.
* Использовать команду *ALTER DATABASE*, чтобы открывать, монтировать, выполнять резервное копирование или изменять набор символов.
* Использовать команду *CREATE DATABASE*.
* Выполнять операции *ARCHIVELOG* и *RECOVERY*.
* Создавать *SPFILE*.

Полномочия *SYSOPER* также включают в себя права *RESTRICTED SESSION* и позволяют выполнять следующее.

* Выполнять операции *STARTUP* и *STDOWN*.
* Использовать команду *ALTER DATABASE*, чтобы выполнять открытие, монтирование или системное копирование.
* Выполнять операции *ARCHIVELOG* и *RECOVERY*.
* Создавать *SPFILE*.

## **3. Понятия базы данных и экземпляра базы данных.**

*База данных Oracle* – это набор файлов с ОС, а *экземпляр* (*инстанс*) – процессы и память, причем одна база данных может быть доступна в нескольких экземплярах, а экземпляр единовременно обеспечивает доступ только к одной базе данных.

(из лекции)

Экземпляр Oracle:

* запущенный сервер (программа) СУБД Oracle
* общая (глобальная) область памяти (SGA – system global area) и др. системные области памяти
* фоновые процессы, предназначенные для управления файлами базы данных

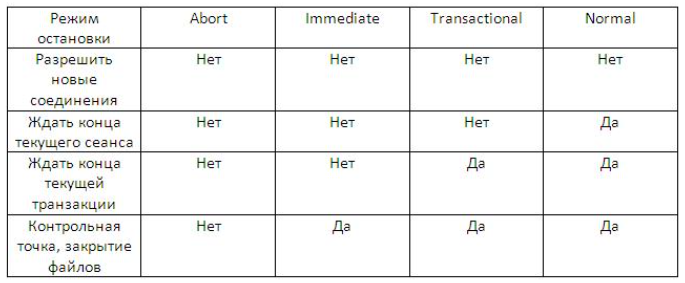
## **4. Запуск и остановка экземпляра базы данных Oracle.**

**StartUp:**

* **SHUTDOWN**
* **NOMOUNT** – Instance started
* **MOUNT** – Control file opened for this instance
* **OPEN** – All files opened as described by the control file for this Instance



*Останов базы данных Oracle*



**Abort** – перед остановкой производится наименьшее число действий. После этого при запуске система должна выполнить восстановление.

**Immediate** – обычно используемая опция. При этом незафиксированные транзакции откатываются.

**Transactional** – представляется возможность завершить транзакции.

**Normal** – экземпляр не останавливается, пока не отсоединяются сеансы.

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

**Словарь данных** – набор таблиц, созданных и обслуживаемых сервером Oracle; содержит информацию о базе данных.

**Словарь Oracle** – набор таблиц и связанных с ними представлений, который представляет возможность отследить внутреннюю структуру базы данных и деятельность СУБД Oracle.

Словарь данных:

* создаётся при генерации базы данных
* обновляется и обслуживается сервером Oracle в фоновом режиме после выполнения операторов DDL
* позволяет запрашивать данные в виде представлений
* содержит следующую информацию:
* имена пользователей сервера Oracle
* уровни привилегий базы данных
* табличные ограничения
* учётные данные

Основные представления:

* ALL - содержит информацию обо всех объектах, которые пользователь может просматривать
* USER - содержит информацию об объектах, для которых пользователь является владельцем
* DBA - содержит информацию всех пользователей. Недоступен без “звания” администратора (dba)
* V$ - динамические штуки (загруженность, процессы и тд) меняется от сессии к сессии

## **6. 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.
* Словарь разбивается на две части: общую часть и локальную

**CDB - container DB** – контейнер базы данных

**PDB - pluggable DB** – подключаемые базы данных

* Можно создавать несколько CDB – для разных версий программного обеспечения СУДБ.
* Одну и ту же PDB можно переносить между CDB
* В CDB создается главный контейнер Root. Root содержит метаданные CDB.
* В одной CDB можно создать до 252 PDB.

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

**Файл параметров** предназначен для хранения параметров экземпляра.

Для того, чтобы запустить экземпляр базы данных, Oracle должен прочесть файл параметров и определить, какие параметры инициализации установлены для этого экземпляра. Oracle считывает файл параметров при запуске базы данных. Можно создать несколько файлов параметров, каждый будет соответствовать различным конфигурациям экземпляра.

* spfile - бинарный файл, который используется сервером Oracle при старте (не читабельный)
* pfile - текстовый файл с параметрами, будет использоваться при старте, если не будет найден spfile (читабельный)

Разница между spfile и pfile в том, что в spfile все записывается само собой, а вот pfile придется заполнять ручками.

**Управляющие файлы.**  База данных Oracle может иметь один или несколько управляющих файлов. Лучше иметь несколько и хранить на разных дисках. Если имеется несколько управляющих файлов, все они должны быть абсолютно идентичными.(select name from v$controlfile;) ControlFile представляет собой чеклист, который содержит значения для основных настроек инстанса.

**Файлы паролей.** Необязательный файл, используется для защиты информации о подключениях привилегированных пользователей. Если отсутствует, то вы можете выполнять администрирование своей базы данных, только локально.

**Файлы трассировки.** Когда возникает ошибка базы данных, может генерироваться файл трассировки (trace file). Они содержит подробную информацию о возникновении ошибки.(SQL> show parameter dump\_dest)

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

**Файлы данных**.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.

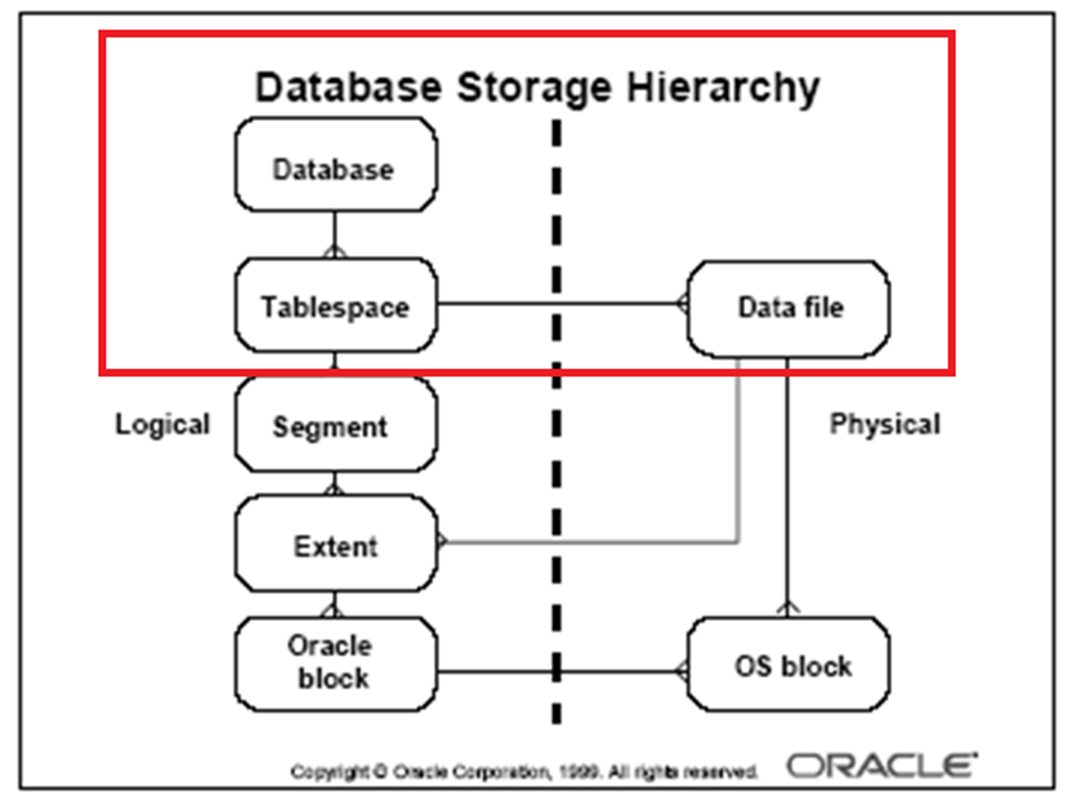
**Журналы повтора**. Оперативные файлы журналов повтора - предназначены для записи всех изменений, выполненных над данными базы данных Oracle. Используется для хранения на диске информации для повторного выполнения операций. Каждая база данных должна иметь не менее двух оперативных файлов журналов повтора. Текущий файл постепенно заполняется, после его заполнения (или переключения некоторыми командами), база данных приступает к записи в следующий файл. Эта операция называется переключением журналов. Поскольку файлы повтора необходимы для выполнения восстановления базы данных и являются критичными, их объединяют в группы. Запись происходит одновременно в файлы одной группы.

**Архивы**. Как только оперативный файл журнала повтора (Redolog) оказывается заполнен, программное обеспечение сервера Oracle начинает запись в следующий файл. Эта операция повторяется, как следствие информация в оперативных файлах журнала (Redolog) многократно перезаписывается.

Если необходимо сохранить историю изменений, нужно, чтобы после переключения журналов сохранялась их копия. Для этого достаточно перевести работу базы данных в режим работы ARCHIVELOG. (select name from v$archived\_log;)

## **9. Абстрактная модель Oracle. Логическая структура внешней памяти.**

Основными единицами физического хранения являются блок данных, экстент, файл (либо раздел жесткого диска).



**Блок** - минимальная единица объема памяти, применяемая при записи и чтении данных.

Размер кратен 2К, должен быть кратен величине блока ОС (2К, 4К, 8К, допустимы 16К и 32К). Размер устанавливается в файле параметров экземпляра при создании БД. В табличном пространстве все блоки одного размера. Вообще по дефолту 8192 размер.

**show parameter block и можно в файле параметров еще посмотреть**

**Экстент** - **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 2ГБ. Вроде как непрерывный он для того, чтоб типа хранить в одном месте (файле , хз).

Если экстенту задана опция UNIFORM, то все экстенты имеют одинаковую длину.

можно еще select \* from user\_extents;

**Сегмент** - область на диске, выделяемая под объекты.

Располагается в табличном пространстве. В одном табличном пространстве может быть много сегментов.

Если сегмент не секционирован, то располагается в одном табличном пространстве.

Управление размерностью сегментов может осуществляться автоматически (segment space management auto) и вручную (segment space management manual).

Сегменты типизируются в зависимости от типа данных, хранящихся в них - сегменты таблиц, индексов, кластеров и тд (всего 10 типов).

SELECT SEGMENT\_TYPE FROM DBA\_SEGMENTS;

Сразу после создания табличного пространства **в нем нет сегментов**.

Сразу после создания таблицы **сегмент не создается**.

Сегмент хранит **только данные**, поэтому он создается только при добавлении данных. При удалении строк сегмент не удаляется.

При удалении таблицы изменяется имя сегмента, и информация об удалении записывается в словарь БД (user\_recyclebin).

Сегмент состоит из одного и более экстентов.

**select \* from user\_recyclebin;**

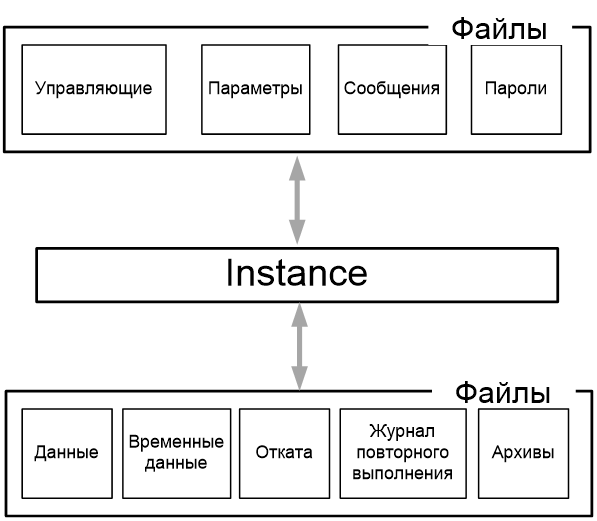
Экстент состоит из идущих подряд блоков.

**СЕГМЕНТ > ЭКСТЕНТ > БЛОК**

## **10. Абстрактная модель Oracle. Физическая структура внешней памяти.**

Основные компоненты архитектуры Oracle:

* Файлы, образующие базу данных и поддерживающие экземпляр - файлы параметров, сообщений, данных, временных данных и журналов повторного выполнения
* Структуры памяти - системная глобальная область (System Global Area — SGA) и входящие в SGA пулы
* Физические процессы или потоки - серверные процессы, фоновые процессы и подчиненные процессы



**Табличное пространство** - логическая структура хранения данных, контейнер сегментов. Это пограничный объект БД: с одной стороны логическая структура сервера, с другой - файл или файлы ОС.

С одним табличным пространством связаны один или **несколько** файлов, с каждым файлом связано только **одно** табличное пространство.

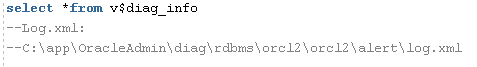
Данные, временные данные, данные отката организованы в виде табличных пространств.

**Физические структуры базы данных Oracle (Oracle Database) — это действительные файлы данных Oracle на уровне операционной системы. База данных Oracle состоит из следующих трех основных типов файлов.**

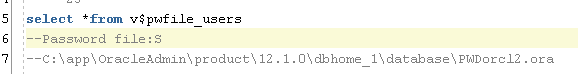
* [**Файлы данных**](https://oracle-patches.com/oracle/begin/1272-%D1%84%D0%B0%D0%B9%D0%BB%D1%8B-%D0%B1%D0%B0%D0%B7%D1%8B-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85-oracle)**.**

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. **SQL>** **select** file**#**, name, status **from** v$datafile;

* [**Управляющие файлы**](https://oracle-patches.com/oracle/prof/641-%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%8F%D1%8E%D1%89%D0%B8%D0%B5-%D1%84%D0%B0%D0%B9%D0%BB%D1%8B-oracle)**.** База данных Oracle может иметь один или несколько управляющих файлов. Если имеется несколько управляющих файлов, все они должны быть абсолютно идентичными. При каждом запуске базы данных Oracle читает информацию управляющего файла, а при каждом изменении размещения или добавления новых файлов данных и журналов базы данных обновляет управляющий файл.SQL> select name from v$controlfile;
* [**Файлы журналов повторного выполнения**](https://oracle-patches.com/oracle/prof/642-%D1%84%D0%B0%D0%B9%D0%BB%D1%8B-%D0%B6%D1%83%D1%80%D0%BD%D0%B0%D0%BB%D0%BE%D0%B2-%D0%BF%D0%BE%D0%B2%D1%82%D0%BE%D1%80%D0%B5%D0%BD%D0%B8%D1%8F-%D1%82%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D0%B9-oracle)**.** . Оперативные файлы журналов повтора - предназначены для записи всех изменений, выполненных над данными базы данных Oracle. Используется для хранения на диске информации для повторного выполнения операций. 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. Поскольку файлы повтора необходимы для выполнения восстановления базы данных и являются критичными, их объединяют в группы. Запись происходит одновременно в файлы одной группы.
* **Файл параметров.** Файлы параметров используются для конфигурирования действий Oracle прежде всего при старте. Для того, чтобы запустить экземпляр базы данных, Oracle должен прочесть файл параметров и определить, какие параметры инициализации установлены для этого экземпляра.
  + spfile - бинарный файл, который используется сервером Oracle при старте.
  + pfile - текстовый файл с параметрами, будет использоваться при старте, если не будет найден spfile.
* **Файл сообщений** Когда возникает ошибка базы данных, может генерироваться файл трассировки (trace file). Они содержит подробную информацию о возникновении ошибки.(SQL> show parameter dump\_dest)



* **Файл паролей** Необязательный файл, используется для защиты информации о подключениях привилегированных пользователей.

****

## **11. Абстрактная модель Oracle. Структура SGA.**

## При запуске экземпляра:

* выделяется область разделяемой памяти, называемая SGA
* запускаются фоновые процессы

Процесс – это механизм в операционной система, который может выполнить последовательность шагов.

**SGA** – системная глобальная область;

**PGA** – глобальные программные области;

SGA:

* группа областей разделяемой памяти
* содержат данные и управляющую информацию для одного экземпляра Oracle
* совместно используется всеми серверными и фоновыми

Структура SGA:



Основные пулы области SGA:

**Буферный пул** (буферный кэш) – область SGA, которая содержит образы блоков, считанные из файлов данных или созданные динамически, чтобы реализовать модель согласованного чтения.

* совместно используется всеми пользователями
* данные ищутся в буферном пуле в первую очередь
* блок - единица обмена информацией между оперативной памятью и диском
* поддерживает два списка блоков: (грязные и чистые блоки)

список грязных блоков (отличаются от своей копии на диске и должны быть записаны в табличное пространство)

}список чистых блоков (не измененные блоки)

* алгоритм LRU (least recently used) - первым вытесняются блоки с наименьшим значением счётчика
* пулы буферного кэша: default, keep, recycle

create table XXX (k int) storage(buffer\_pool keep/recycle/default) tablespace users;

**Буфер журналов повторного** **выполнения** предназначен для временного циклического хранения данных журнала повтора.

* позволяет ускорить работу сервера за счет буферизации
* размер буфера задается в файле параметров

**Фиксированная область SGA** – хранит переменные, указывающие на другие области памяти, значения параметров;

* представляет собой загрузочный бинарный код;
* размер области зависит от платформы, версии и ОС;
* размером этой области управлять нельзя;

**Разделяемый пул**

* библиотечный кэш
* разделяемую область SQL
* кэш словаря данных
* управляющие структуры

**Большой пул** – область памяти SGA, применяемая для хранения больших фрагментов памяти

* не применяется вытеснение по алгоритму LRU
* память освобождается сразу как перестает использоваться
* аналог RECYCLE (разделяемый пул - KEEP)
* хранятся данные при резервном копировании, специальные области UGA и т.д.

**Java-пул** – предназначен для работы Java-машины

## **12. Абстрактная модель Oracle. Серверные процессы Oracle.**

*Процесс* (process) – механизм ОС Windows, осуществляющий запуск и выполнение приложений:

* процесс создаётся, когда запускается приложение
* в общем случае выполняется в собственной области памяти

*Поток* (thread) – индивидуальная ветвь внутри процесса, выполняющая конкретные программные инструкции.

*Серверные процессы* – процессы, выполняющиеся на основании клиентского запроса.

Серверные процессы:

* клиентский процесс прямо взаимодействует с соответствующим серверным процессом по сети
* Первичное установление соединения происходит через специальный серверный процесс Oracle Net Listener
* Серверный процесс:
* получает и выполняет SQL-операторы
* читает файлы данных
* осуществляет поиск в кеше
* Назначение – отвечать на получаемый SQL-запросы
* Клиентом может быть любая программа, пытающаяся подключиться к СУБД

Отправка SQL-запроса select \* from students;

* производится синтаксический разбор
* помещается в разделяемый пул (находит в пуле)
* создаёт план запроса и выполняет его
* при необходимости, производит поиск данных в буферном кэше или запрашивает в кэш с диска

Серверные процессы выполняют работу от имени сеанса клиента:

* подключение с помощью выделенного сервера
* подключение с помощью разделяемого сервера

Режимы сессий (dedicated, shared):

select server from v$session;

## **13. Абстрактная модель Oracle. Фоновые процессы Oracle.**

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

*Подчиненные процессы* – аналогичны фоновым, но выполняют дополнительные действия для фонового или серверного процессов.

*Фоновые процессы* – специальная группа процессов для обеспечения производительности и поддержки работы большого числа пользователей:

* **LREG** – Listener Registration Process – периодическая регистрация сервисов в процессе Listener;
* **DBW** – Database Writer Process: (n=0,...9,a,..,z; BWm, m=36,...,99) – фоновые процессы записывающие по LRU измененные блоки (грязные блоки) в файлы базы данных.
* **CKPT** – CheckPoint – выполняет процесс чек-поинт.
* выполняется при shutdown, alter system checkpoint,переключении REDO-журнала, периодическом сообщении от DWR, backup
* даёт команду DBW и LGWR на сброс буферов
* **LGWR** – Log Writer Process – только один. Управляет буфером журнала повтора; записывает блоки буфера журналов повтора в группы журналов; переключает текущую группу.
* **ARCn** – Archiver Process – копирует файлы журнала повтора после переключения группы журналов. Необязательный процесс.
* **PMON** – Process monitor, очистка после ненормального закрытия подключений, инициирует откат незафиксированных транзакций, снятие блокировок, освобождение ресурсов SGA, следит за работой других фоновых процессов, отвечает за их перезапуск
* **SMON** – System Monitor Process, восстановление экземпляра для узла, восстановление незавершенных транзакций, очистка временных сегментов данных
* **RECO** – Recovery Process - разрешение проблем связанных с распределенными транзакциями
* **FBDA** – Flashback Data Archiever, архивирование ретроспективных данных

## **14. Процесс-слушатель Oracle и его основные параметры.**

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

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

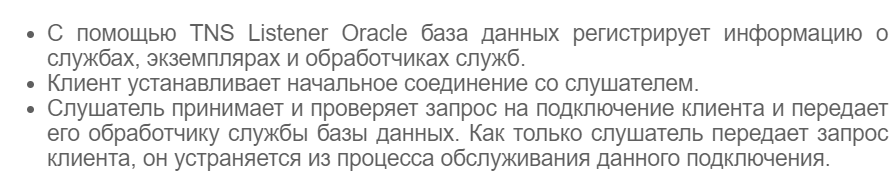
* По умолчанию используется порт **1521**.
* Конфигурационный файл - **Oracle\_Home\Network\Admin\listener.ora** считывается при старте Listener. Может быть сконфигурирован **только один** файл listener.ora, на сервере может быть настроено несколько прослушивателей, и этот единственный файл может обслуживать их все. Как правило, если на одном сервере настроено несколько прослушивателей, то это сделано либо с целью обеспечения отказоустойчивости, либо для балансировки обращений к базе данных. Несколько прослушивателей, так же настраиваются при использовании Real Application Cluster.
* Экземпляр может иметь несколько точек подключения (сервисов)
* При инсталляции создается автоматически два сервиса:

**SYS$USERS** (по умолчанию, указывается SID в параметрах соединения)

**сервис и именем инстанса** ( указывается сервис)

* При создании PDB для нее автоматически добавляется сервис с именем, совпадающим с PDB-именем

????????????



* Утилита управления процессом Listener - **lsnrctl.** Для прослушивателя доступны три основные команды: start, stop и status. Команда start служит для запуска процесса прослушивания, stop – остановка, status – показывает текущий статус прослушивателя.

**Старт экземпляра:**

* фоновый процесс LREG считывает имена и параметры зарегистрированных сервисов экземпляра
* регистрирует их в Listener
* если Listener не обнаружен, то попытки регистрации периодично повторяются
* Listener прослушивает запросы к сервисам

**Запрос на соединение:**

1)клиент выполняет запрос к Listener на соединение с сервисом экземпляра

2) Listener запрашивает соединение с сервером

3)сервер возвращает параметры соединения с обработчиком сервиса

4) Listener сообщает параметры соединения клиенту

5) клиент соединяется с обработчиком запросов сервиса для дальнейшей работы с сервисом в рамках соединения

## **15. Сетевые настройки Oracle. Установление соединения по сети. (tnsnames.ora ????????)**

Oracle-инстанс поддерживает два режима соединений: **dedicated server** (выделенный сервер) и **shared server** (разделяемый сервер). В зависимости от режима два вида обработчиков запросов к сервису. Могут работать оба режима. **Dedicated server** (режим работы инстанса по умолчанию) –

для каждого клиента выделяется отдельный выделенный серверный процесс (обработчик запросов, dedicated server process) который называется shadow process (теневой процесс). **Shared server** – в качестве обработчика запросов выступает программа **dispatcher**, которая: 1)получает запрос от клиента, 2)помещает их во входную очередь к разделяемым серверам; 3)незанятый разделяемый сервер извлекает и обрабатывает запрос; 4)после обработки разделяемый сервер помещает результат обработки в выходную очередь; 5) из очереди результат извлекает диспетчер; 6) диспетчер пересылает результат клиенту.

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

Состоит из двух компонентов: **Oracle Network Foundation Layer и Oracle Protocol Support**

Net Manager позволяет настроить следующие сетевые компоненты:

· **Слушатели:** Создание и настройка слушателей для получения клиентских подключений.

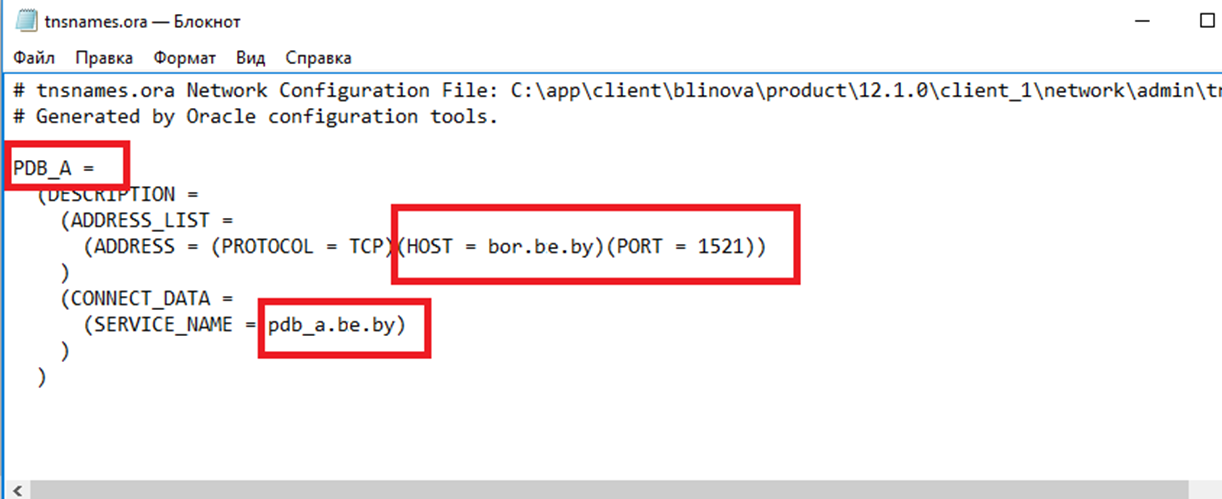
· **Нейминг:** Определить [идентификатор подключения](https://docs.oracle.com/cd/E11882_01/network.112/e41945/glossary.htm#BGBBGCEG) s и сопоставить их [подключить описатель](https://docs.oracle.com/cd/E11882_01/network.112/e41945/glossary.htm#BGBEAGEA) s для определения местоположения сети и идентификации службы. Oracle Net Manager поддерживает конфигурацию CONNECT дескрипторов в локальных tnsnames.ora файлах или централизованной службе каталогов.

· **Профили:** Настройка предпочтений для включения и настройки функции Oracle Net на клиенте или сервере.

Oracle Net Manager (Диспетчер сети Oracle) позволяет конфигурировать различные методы именования и слушателей. С помощью этого средства можно конфигурировать дескрипторы соединений в локальных файлах tnsnames.ora или в централизованном OID, а также легко добавлять и изменять методы подключения.

Для подключения необходимо написать в sqlplus: **connect имя/пароль@[//]host:[port][/имя службы]**

После создания соединения в файле tnsnames.ora будет информация:



**sqlnet.ora** is a plain-text [configuration file](https://en.wikipedia.org/wiki/Configuration_file) that contains the information (like tracing options, encryption, route of connections, external naming parameters etc.) on how both [Oracle](https://en.wikipedia.org/wiki/Oracle_Database) server and [Oracle](https://en.wikipedia.org/wiki/Oracle_Database) client have to use [Oracle Net](https://en.wikipedia.org/wiki/Oracle_Net_Services) (formerly Net8 or SQL\*Net) capabilities for networked database access.

NAMES.DIRECTORY\_PATH= (LDAP, TNSNAMES, HOSTNAME)

NAMES.DEFAULT\_DOMAIN = ORACLE.COM

TRACE\_LEVEL\_CLIENT = ON

SQLNET.EXPIRE\_TIME = 30

SQLNET.IDENTIX\_FINGERPRINT\_DATABASE = FINGRDB

AUTOMATIC\_IPC = ON

SQLNET.AUTHENTICATION\_SERVICES = (ALL)

SQLNET.CRYPTO\_CHECKSUM\_CLIENT = ACCEPTED

TNSPING.TRACE\_DIRECTORY = /oracle/traces

## **16. Табличные пространства СУБД Oracle и их основные параметры.**

**Табличное пространство** - логическая структура хранения данных, контейнер сегментов. Также это пограничный объект бд: с одной стороны логическая структура сервера, с другой файл или файлы ОС.

! С одним ТП(табличное пространство) мб связаны один или несколько файлов, с каждым файлом связано только одно ТП.

Предопределенные ТП:

* **SYSTEM** используется для управления БД, содержит словарь БД, стандартные пакеты процедур.
* **SYSAUX -** вспомогательное ТП
* **TEMP -** временное ТП по умолчанию
* **UNDOTBS1 -** ТП отката
* **USERS -** хранение пользовательских объектов БД
* **EXAMPLE -** демонстрационные схемы

SELECT \* FROM DBA\_TABLESPACES;

**Виды табличных пространств:**

* PERMANENT - хранение постоянных объектов (таблиц, индексов, кластеров…)
* TEMPORARY - временные данные
* UNDO - для отката

CREATE TABLESPACE SMALLFILE TABLESPACE\_1

DATAFILE ‘C:\...\name.dbf’ SIZE 10M AUTOEXTEND ON NEXT 2M MAXSIZE 100M,

LOGGING

ONLINE

……..

SMALLFILE - 1022 файла

BIGFILE - 1 файл

**LOGGING/ NOLOGGING/ FORCE LOGGING -·** журналирование изменений объектов базы данных хранящихся в табличном пространстве. Для конкретной операции можно отменить журналирование. Можно указать FORCE LOGGING – обязательное журналирование, которое не может быть отменено для конкретного объекта.

**ONLINE/ OFFLINE** -Табличное пространство недоступно непосредственно после создания, пока не будет переведено в состояние online.

**REUSE** Позволяет ораклу переиспользовать этот файл

## 

Чтобы пользователь имел возможность заполнять табличные пространства данными необходимо выдать ему квоту для данного табличного пространства.

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

**Роль** - это именованная группа связанных привилегий, которые могут быть предоставлены пользователю. Такой метод облегчает выдачу и отъем привилегий. Пользователь может иметь доступ к нескольким ролям, и нескольким пользователям может быть назначена одна и та же роль. Роли обычно создаются для приложения базы данных.

**Привилегия** - это право на выполнение конкретной команды SQL или право доступа к объектам других пользователей. Привилегии делятся на 2 категории: **системные и объектные.**

**Системные привилегии:**

**WITH ADMIN OPTION -** дает право пользователю также назначать/отбирать привилегии.

* **ALTER**
* **ANALYZE**
* **AUDIT**
* **BACKUP**
* **CREATE**
* **DROP**
* **SELECT**

**ANY- для любого объекта**

**ALL - для всех объектов**

**Объекты грантов для системных привилегий:**

* **DATABASE**
* **USER**
* **PROFILE**
* **TABLESPACES**
* **ROLE**
* **TABLE**
* **INDEX**
* **TRIGGER**
* **PROCEDURE**
* **SEQUENCE**
* **VIEW**

**Объектные привилегии:**

**WITH GRANT OPTION -** дает право пользователю также назначать/отбирать привилегии. ! Отбирает привилегии тот же, кто выдал.

* **ALTER**
* **DELETE**
* **EXECUTE**
* **INSERT**
* **UPDATE**
* **SELECT**
* **REFERENCES**

**Объекты грантов для объектных привилегий**

* **TABLE**
* **VIEW**
* **SEQUENCE**
* **PROCEDURE**

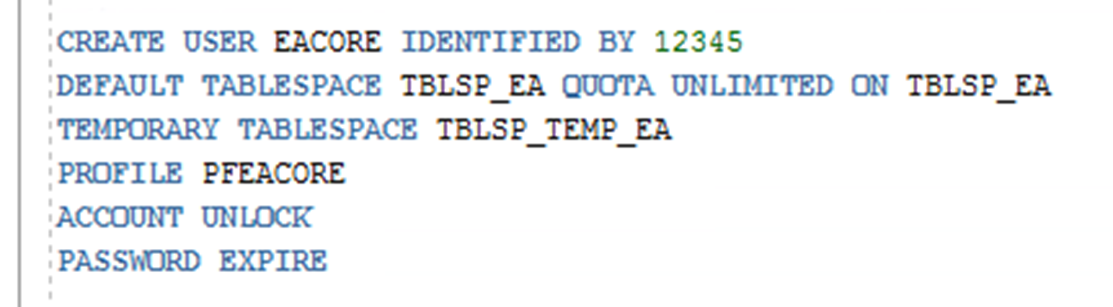
Привилегии назначаются/отнимаются роли с помощью:

**GRANT privileges ON object TO role\_name**

**REVOKE privileges ON object FROM role\_name**

Просмотреть системные привилегии : select \* from dba\_roles

## **18. Пользователь СУБД Oracle и его основные параметры.**



Пользователь БД (User) - это физическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации.

На каждом этапе развития базы данных (проектирование, реализация, эксплуатация, модернизация и развитие, полная реорганизация) с ней связаны разные категории пользователей.

Параметры пользователя, которые можно установить при его создании.

Создание имя и пароля

default tablespace табличное пространство (по умолчанию)

• temporary tablespace временное табличное пространство

• profile профиль безопасности

• account статус пользователя

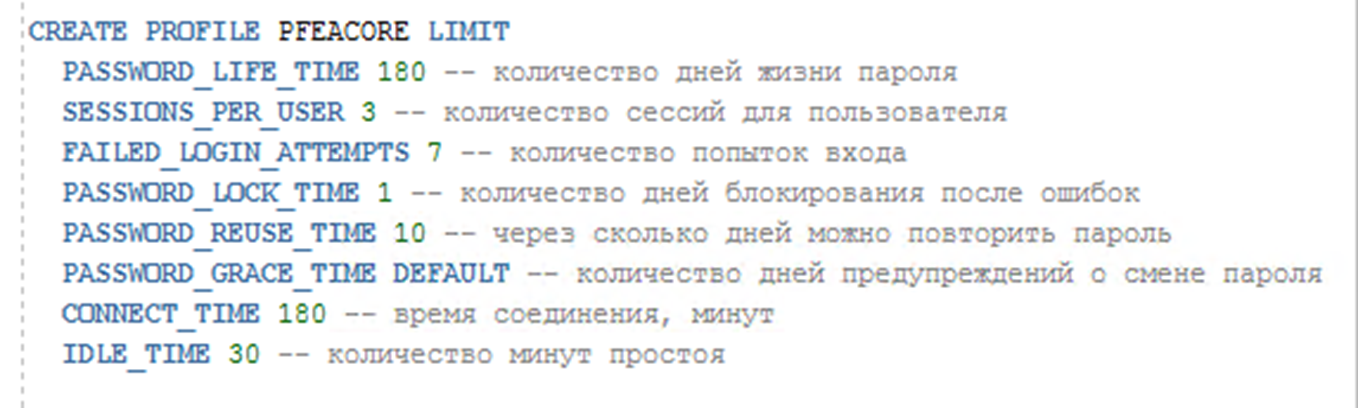
• password статус пароля пользователя

• quota объем памяти в табличном пространстве, которое может занимать пользователь

Просмотреть список пользователей базы данных: select \* from all\_users

Просмотреть список привилегий пользователя: SELECT \* FROM USER\_TAB\_PRIVS\_RECD;

## **19. Профиль безопасности СУБД Oracle и его основные параметры.**



Профиль безопасности – это системный объект базы данных, хранящий атрибуты безопасности.

Профиль с именем DEFAULT: профиль с неограниченными ресурсами. По-умолчанию назначается всем пользователям и не имеет ограничений.

Если пользователю на назначен профиль безопасности, либо в назначенном профиле отсутствуют некоторые параметры, то они берутся из профиля по умолчанию

Назначить профиль безопасности юзеру:

**GRANT RLEACORE TO EACORE;**

## **20. Язык SQL. Основные операторы и их назначение.**

Доступ к данным осуществляется в виде запросов, которые формулируются на стандартном языке запросов. Сегодня для большинства СУБД таким языком является SQL.

SQL не является языком программирования в традиционном представлении. На нем пишутся не программы, а запросы к базе данных. Поэтому SQL —декларативный или непроцедурный язык. Это означает, что с его помощью можно сформулировать, что необходимо получить, но нельзя указать, как это следует сделать.

Язык SQL делится на подмножества:

1) Язык определения данных (DDL - Data Definition Language) предоставляет пользователям средства указания типа данных и их структуры, а также средства задания ограничений для информации, хранимой в базе данных.

Операторы –CREATE, ALTER, DROP.

2) Язык манипулирования данными (DML - Data Manipulation Language) позволяет вставлять, обновлять и извлекать информацию из базы данных.

Операторы –SELECT, INSERT, DELETE, UPDATE.

3) Язык управления данными (DCL - Data Control Language) состоит из управляющих операторов.

Операторы –GRANT, REVOKE.

4) Язык управления транзакциями.

Операторы –COMMIT, ROLLBACK, SAVEPOINT.

Запрос на языке SQL состоит из одного или нескольких операторов, следующих один за другим и разделенных точкой с запятой.

Каждый столбец в любой таблице хранит данные определенных типов. Различают базовые типы данных —строки символов фиксированной длины - CHAR(n), целые и вещественные числа - NUMERIC(n,m), DEC(n,m), INTEGER, SMALLINT, FLOAT(n), REAL, DOUBLE PRECISION , и дополнительные типы данных —строки символов переменной длины - VARCHAR(n), BIT VARYING(n), денежные единицы, дату и время –DATE, TIMESTAMP, INTERVAL, логические данные –BOOLEAN (два значения —"ИСТИНА" и "ЛОЖЬ").

Современные базы данных содержат различные типы объектов:

таблицы, индексы, представления, роли, синонимы, последовательности, кластеры, триггеры, процедуры, функции, пакеты, пользователи, профили и т.д.

Все эти объекты создаются, изменяются и удаляются с помощью операторов DDL.

## **21. Таблица и ее основные параметры.**

**Таблица** – основная структура сохранения информации в БД.

**Типы таблиц:**

Традиционные таблицы (heap organized table)

Индекс-таблицы (index organized table)

Кластеризованные индекс-таблицы (index clustered table)

Кластеризованные хэш-таблицы (hash clustered table)

Отсортированные кластеризованные хэш-таблицы (sorted hash clustered table)

Вложенные таблицы (nested table)

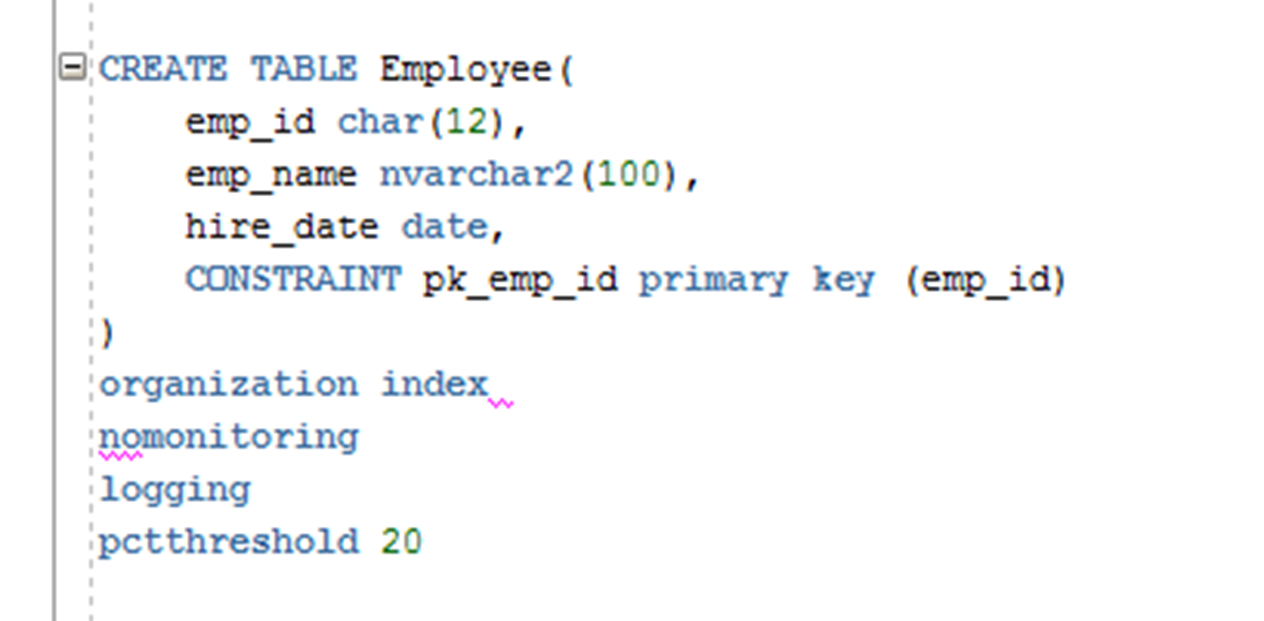
Временные таблицы (temporary table)

Объектные таблицы

Внешние таблицы

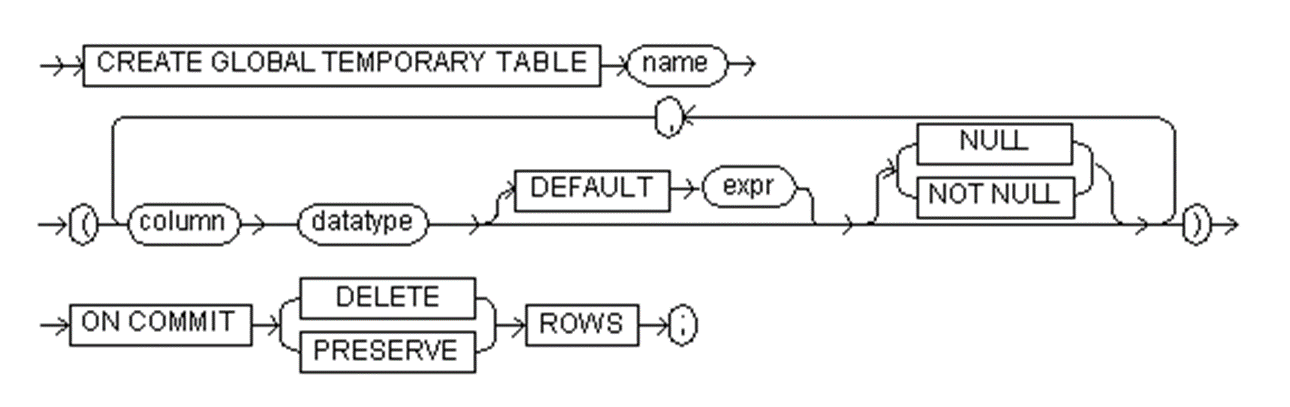
* Может иметь до 1000 столбцов (<254)
* Может иметь неограниченное число строк
* Может иметь неограниченное число индексов
* Нет ограничения на число таблиц

Параметры



## **22. Временные таблицы.**

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



Временные таблицы – глобальны. Привилегии для создания временной таблицы CREATE TABLE. Можно разместить временную таблицу в заданном табличном пространстве. Временные таблицы – это шаблон, хранящийся в словаре базы данных, для нее выделяется временный сегмент в (по умолчанию) TEMPORARY-табличном пространстве и для каждого пользователя свой. Каждый пользователь видит только свои данные (свой сегмент данных). Для временных таблиц можно создавать триггеры.

Для временных таблиц можно указать констрейны (ограничения).

Для временных таблиц можно создавать индексы. Не могут быть индексно-организованными, нельзя секционировать, размещать в кластере.

Данные повторного выполнения генерируются, но их количество пренебрежительно мало.

create global temporary table tablename

(

);

## **23. Ограничения целостности в таблицах.**

**Це́лостность ба́зы да́нных** — соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам. Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется ограничением целостности.

* Primary Key
* Foreign Key
* data type
* not null
* check
* unique
* Between

## **24. Типы данных базы данных.**

* char - символьное поле фиксированное длины до 2к байт
* nchar - поле фиксированной длины для набора символов, состоящих из нескольких байт. Максимальный размер - 2к символов или 2к байт в зависимости от набора символов.
* varchar2 - символьное поле переменной длины до 4к байт.
* nvarchar2 - поле переменной длины для набора символов, состоящих из нескольких байт. Максимальный размер - 4к символов или 4к байт.
* long - символьный, переменной длины, до 2gb, оставлен для совместимости
* raw(n) - переменной длины для бинарных данных n <= 2к байт, оставлен для совместимости
* long raw - бинарные данные до 2 gb
* clob - символьный большой объект до 4gb
* nlob - clob для многобайтных символов
* blob - большой двоичный объект до 4gb
* bfile - указатель на двоичный файл в ос
* date - 7 байтовое поле фиксированной длины, используемое для хранения даты и времени
* interval day to seconds – 11 байтовое поле фиксированной длины для интервала времени: дни часы минуты секунды
* interval year to month timestamp - 5 байтовое поле фикс. длины для интервала времени: годы и месяц
* timestamp with time zone - 13 байтовое поле фиксированной длины, дата время и настройки, связанные с часовым поясом.
* timestamp with local zone - 7-11 байтовое поле переменной длины, дата и время, приведенные к часовому поясу БД.
* number (n,s) - числовой тип переменной длины. Точность n <= 38, общее количество цифр. Масштаб s = [-84, 127], количество цифр после запятой.
* rowid - 16-ричный тип для уникального определения любой строки в БД.  
  Длина - 18 символов, которая делиться на 4 элемента (6 + 3 + 6 + 3)
  + OOOOOO - уникальный номер объекта в бд
  + FFF - уникальный номер файла данных в бд
  + BBBBBB - номер блока данных
  + RRR - адрес строки в блоке
* urowid - специальный 16-ричный тип данных для адресации строк в таблицах, организованных по индексу.

## 

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

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

Типы индексов:

**Табличный (B\*Tree) индекс**

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

**Битовый индекс**

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

Предназначен для индексирования столбцов с низкой селективностью

Не подходит для таблиц с частым обновлением

Хорошо подходят для хранилищ данных

**Функциональный индекс**

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

LOWER(NAME) / UPPER (NAME)

**Кластерный индекс**

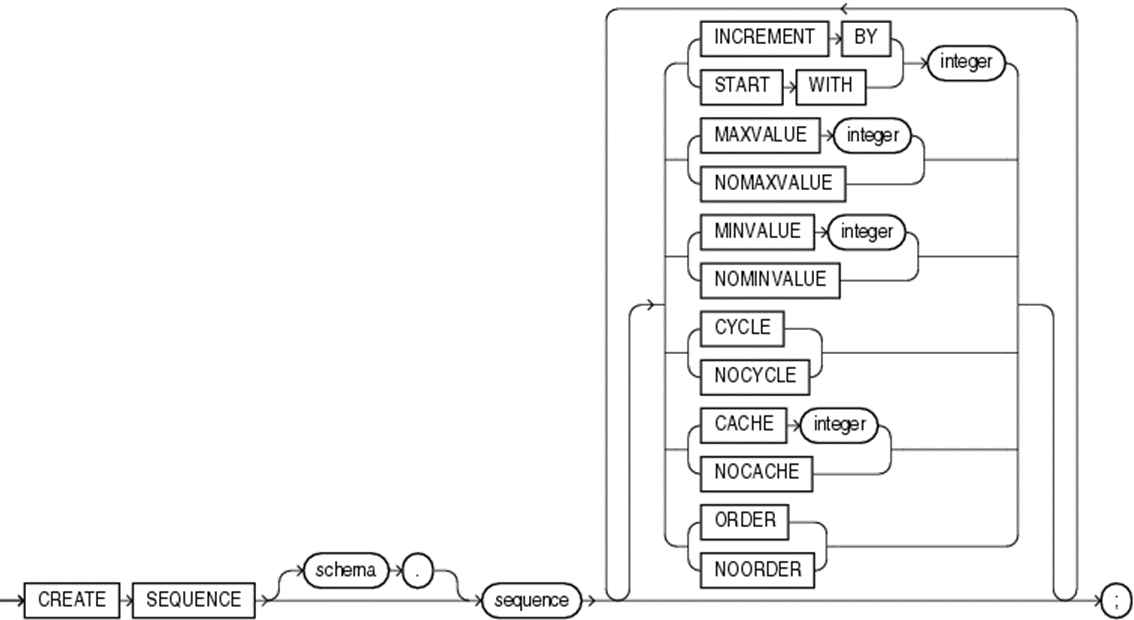
**Плотность запроса** – количество возвращаемых строк запроса

**Селективность запроса** – количество возвращаемых значений ключа (% разных ключей от общего количества)

create (bitmap|cluster|....) index indexname on table(column)

## **26. Последовательность СУБД Oracle и ее параметры.**

**Последовательность** – объект базы данных, предназначенный для генерации числовой последовательности (необходима привилегия create sequence)



create sequence S1

increment by 10

start with 1000

nomaxvalue – maxvalue 10

nominvalue – minvalue 0

nocycle – cycle

nocache – cache 20

noorder; – order

**start** **with —** позволяет создателю последовательности указать первое генерируемое ею значение. После создания последовательность генерирует указанное в start with значение при первой ссылке на ее виртуальный столбец **NEXTVAL**

**increment by n** — определяет приращение последовательности при каждой ссылке на виртуальный столбец **NEXVAL**. Если значение не указано явно, по умолчанию устанавливается 1. Для возрастающих последовательностей устанавливается положительное n, для убывающих, или последовательностей с обратным отсчетом — отрицательное

**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 -1026, 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 1027, 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. Если циклический режим нежелателен или не установлен явным образом, Oracle применяет значение по умолчанию – **NOCYCLE**. Указывать **CYCLE** вместе с **NOMAXVALUE** или **NOMINVALUE** нельзя. Если нужна циклическая последовательность, необходимо указать **MAXVALUE** для возрастающей последовательности или **MINVALUE** – для убывающей

**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. Эта опция может использоваться, к примеру, когда номера последовательности предстают в качестве отметок времени. Гарантирование порядка обычно не существенно для тех последовательностей, которые используются для генерации первичных ключей. Если упорядочение нежелательно или не установлено явным образом, Oracle применяет значение по умолчанию **NOORDER**

**noorder** — не гарантирует, что номера последовательности генерируются в порядке запросов

select s1.nextval from dual; – берет следующее значение из последовательности и изменяет текущее значение на следующее

select s1.currval from dual – берет текущее значение из последовательности

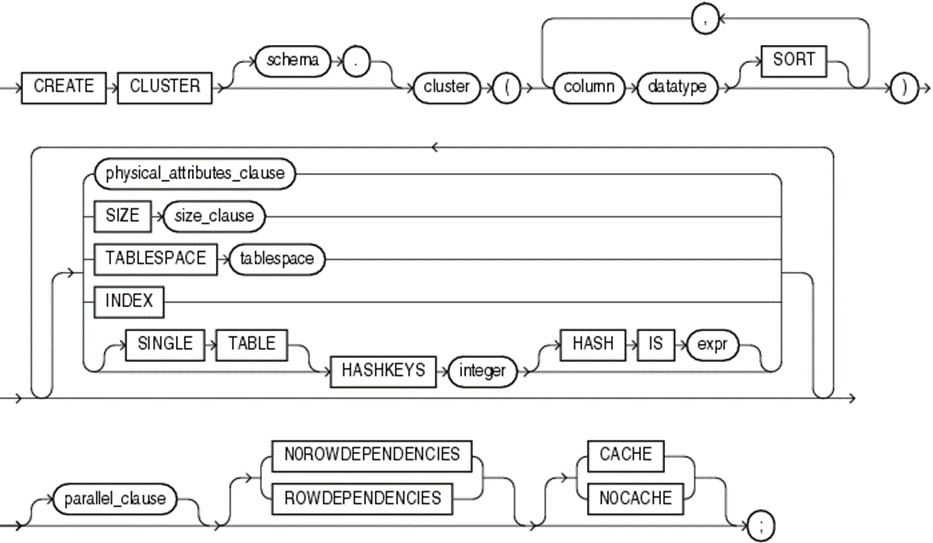
**sys.dba\_sequences**

**sys.all\_sequences**

**sys.user\_sequences**

## **27. Кластер и его параметры**

**Кластеры -** таблицы, с которыми часто работают совместно, можно физически хранить совместно. Для этого создается кластер, который будет их содержать. Строки из отдельных таблиц сохраняются в одних и тех же блоках, поэтому объединяющие запросы выполняются быстрее.



Уменьшается количество операций ввода-вывода. Производительность операций вставки, обновления и удаления может быть ниже, чем для обычных таблиц.

Связанные столбцы называются кластерным ключом.

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

Наибольшие преимущества – в запросах, использующих операции равенства:

select Name from STUDENT where Id = 999;

create cluster ABC(

X number(10),

V varchar(12)

) hashkeys 200;

**hashkeys -** создание хэш-кластера на n значений.

create table A(

XA number(10),

VA varchar(12),

i int

)cluster ABC(XA, VA);

create table B(

XB number(10),

VB varchar(12),

iB int

)cluster ABC(XB, VB);

select \* from user\_clusters;

select \* from user\_tables order by TABLE\_NAME;

## **28. Представление и его параметры.**

**Представление** – хранимый запрос. Можно обращаться, как к обычной таблице. Данные хранятся в таблице. Добавляют уровень защиты данных. Скрывают сложность данных. Скрывают имена столбцов таблиц

**FORCE** – создает представление, независимо от того, существуют ли таблицы и есть ли права

**NOFORCE** – по умолчанию

**WITH CHECK OPTION** – указывает, что будут вставлены или изменены строки, которые будут выбираться через это представление

**READ ONLY**

create view viewname

as select column1, column2 from tablename;

select \* from viewname;

alter view viename compile (пересоздание представления при изменении таблицы)

## **29. Материализованное представление и его параметры.**

**Материализованные представления -** представления, хранящиеся в памяти, которые обновляются в определенное время. Материализованные представления можно даже секционировать и при необходимости создавать на них индексы.

**BUILD IMMEDIATE** – создает представление в момент выполнения оператора

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

**NEXT** – показывает, когда выполнится в следующий раз

Далее – в разницу времени между START WITH и NEXT

create materialized view viewname

build immediate

refresh complete start with to\_date(‘07-10-2021’,’DD-MM-YYYY’)

next to\_date(‘10-10-2021’, ‘DD-MM-YYYY’)

as select \* from tablename;

## **30. Частные и публичные синонимы СУБД Oracle.**

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

Частный синоним принадлежит пользователю, который его создал.

Публичный синоним используется совместно всеми пользователями базы данных.

Привилегия - create (public) synonym

Синоним может указывать на:

* Таблицы
* Процедуры
* Представления
* Функции
* Последовательности
* Пакеты

create synonym t1 for teacher;

create public synonym t2 for teacher;

select \* from t1;

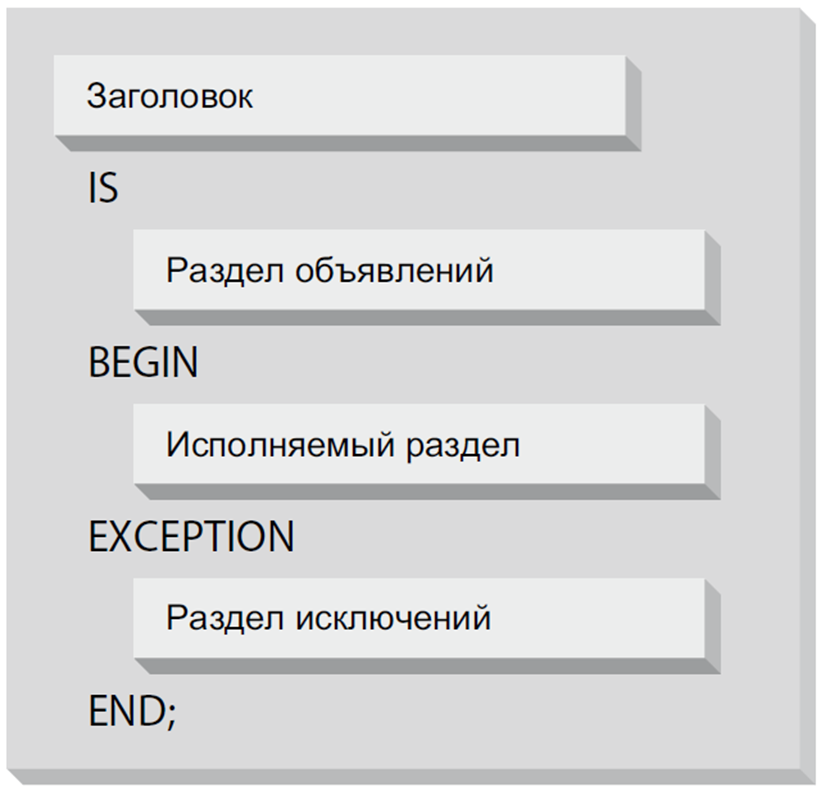
## **31. Основные характеристики языка PL/SQL.**

PL/SQL - Procedural Language extensions to SQL:

* основной язык для программирования хранимых процедур
* не чувствителен к регистру
* интегрирован с БД Oracle
* производительность серверных модулей
* приложение может быть проще в реализации при написании бизнес-логики на основе хранимых процедур
* отсутствие накладных расходов на приведение типов
* может выполняться независимо от пользователя
* PL/SQL функции можно вызывать из select запросов
* взаимодействие с пользователем
* внутренний язык
* содержит элементы ООП
* позволяет использовать объектные типы
* интерпретация (режим по умолчанию)
* компиляция (промежуточный код на С и конечный объектный код процессора)
* среда выполнения: SQL\*Plus, SQL Developer, TOAD

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

Схема блока PL/SQL



Анонимный блок:

* не имеет секции заголовка
* используется как скрипт для выполнения PL/SQL выражений
* не может быть вызван из другого блока
* начинается с declare или begin
* варианты использования:

1. триггер на стороне клиента
2. триггер БД
3. SQL-скрипт (описание процедур, функций, execute)
4. откомпилированная программа (блок в execute клманде, выполняющийся на сервере)

begin

--тело--

end;

Именованные блоки – процедуры и функции.

declare

x nuber(3):=3;

begin

dbms\_output.put\_line(‘x = ’ || x);

end;

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

Типы данных PL/SQL:

* скалярные (Scalar):
* символ (char(n), nchar(n)), строка (varcahr(n), varchar2(n), nvarchar2(n))
* число (number(n, m), binary\_float, binary\_double, binary\_integer)
* булев (boolean)
* дата/время (date, timestamp)
* ссылочные (Reference)
* составные (Composite)
* большие объекты (LOB)

Константы объявляются с помощью ключевого слова CONSTANT, в нем обязательно указывается значение по умолчанию (которое на самом деле является не значением по умолчанию, а единственно возможным значением).

declare n1 constant number(5) := 5;

Значение константы задается в момент объявления и впоследствии не может быть изменено.

Основные операции (?):

* if

if 8 > 10

then dbms\_output.put\_line(‘sosat’);

end if;

* case

case x

when 1 then …;

when 2 then …;

else …;

end case;

—-----------------

case

when 8 > 10 then …;

when 8 < 10 then …;

else …;

end case;

* loop

loop

x:= x + 1;

exit when x > 5;

end loop;

* for

for x in 1..5

loop

…

end loop;

* while

while (x > 0)

loop

x:= x - 1;

end loop;

* арифметические операции (+-:=/\*)

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

NLS - National Language Support (Globalization Support)

Можно хранить данные множества национальных языков, используя Unicode или специальные кодировки - наборы символов.

Символы хранятся как коды символов, зависящие от выбранного набора символов.

В одной БД могут использоваться два **набора символов**: основной (database character set) и дополнительный (national character set). Устанавливаются при создании БД. Изменяются alter database (national) character set.

Основной набор символов используется для:

* хранения символьных типов char, varchar2, clob, 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.

Переменная окружения NLS\_LANG:

NLS\_LANG = language\_territory.charset

* Язык (LANGUAGE) – имена месяцев, дней, направление текста, сокращения для времени и дат. По умолчанию AMERICAN.
* Территория (TERRITORY) – настройка календаря, формат даты, денежной единицы. Если не указан, то будет взято значение, соответствующее языку
* Набор символов (CHARSET) – отображение символов, отображение и конвертация заглавных букв, порядок замещения символов при преобразовании. Каждому языку поставлен в соответствие набор символов по умолчанию.

Представления словаря:

* nls\_session\_parameters
* nls\_instance\_parameters
* nls\_database\_parameters

Байтовая семантика рассматривает строки как последовательность байт, символьная – как последовательность символов.

Задается параметром NLS\_LENGTH\_SEMANTICS. По умолчанию BYTE.

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

Связанное объявление переменных – тип переменных основан на известной структуре данных.

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

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

subject svvcore.subject.subject%type;

faculty\_rec svvcore.faculty%rowtype;

## **36. Локальные процедуры и функции языка PL/SQL.**

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

Объявление локальных процедур и функций должно размещаться в конце секции декларации после всех типов, записей, курсоров, переменных и исключений.

Локальные процедуры и функции могут быть использованы только в рамках блока, в котором они объявлены.

Локальные процедуры и функции могут быть перегружены.

**Перегрузка программных модулей:**

* 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

**Локальная процедура:**

declare

x number := 4;

procedure sumself (x1) is

z number := 5;

begin

x3 := x1 + x1;

end sumself;

begin

sumself(x);

end;

**Локальная функция:**

declare

x number := 4;

s number;

function sumself (x1) return number is

z number := 5;

begin

x3 := x1 + x1;

return x3;

end sumself;

begin

s:= sumself(x);

end;

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

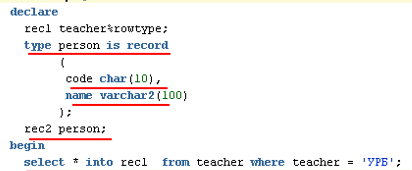
Запись – структура данных, составленная из нескольких частей информации (полей).

Для объявления записи надо определить как тип и объявить переменную типа “запись”.

Типы записей:

* табличные
* курсорные
* программно-определенные

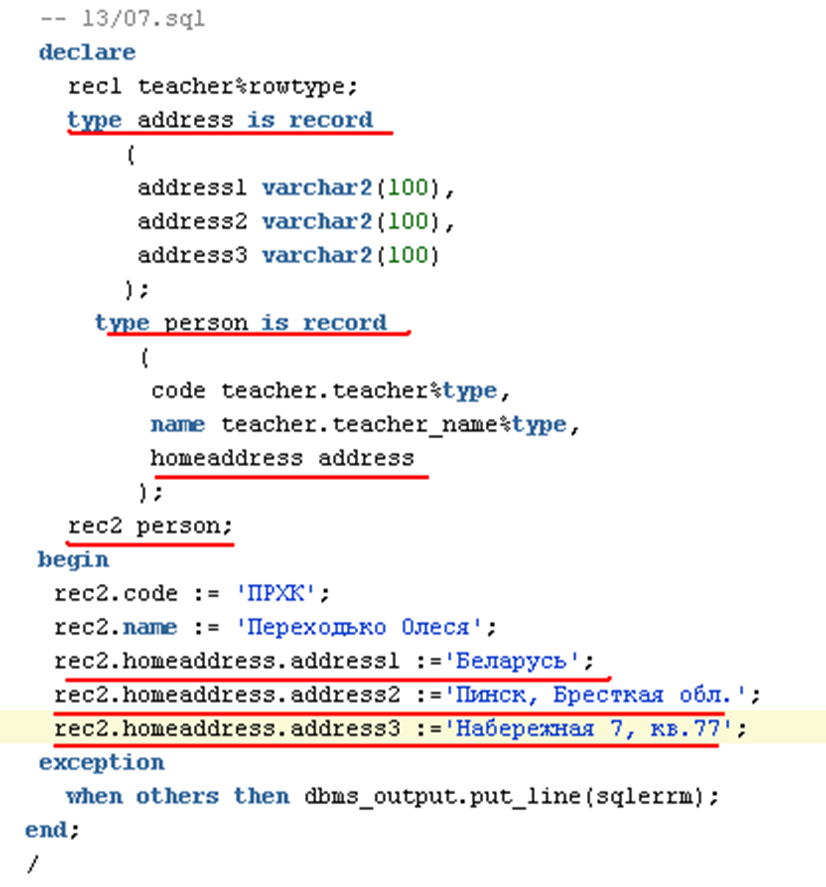
**Объявление:**

1. 
2. 

**Использование полей записи:**

* сравнение производится по полям записи
* присвоение: для отдельного поля, select into в запись целиком или в отдельные поля, присвоение значения одной записи другой записи – для одного и того же объявления TYPE

**Вложенные записи** – одна запись является полем другой записи.

****

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

**Операторы управления:**

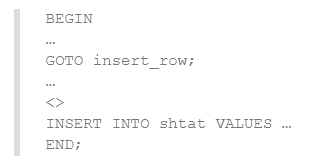
* **условного перехода**: if-then

if 8 > 10

then dbms\_output.put\_line(‘sosat’);

end if;

* if-then-else
* if-then-elsif
* **безусловного перехода** GOTO – в любом месте программы может быть поставлена метка, имеющая синтаксис: <<имя\_метки>>. Оператор GOTO позволяет осуществить безусловный переход к метке, имя которой должно быть уникальным внутри программы или блока PL/SQL



* **NULL** – пустой оператор; он передает управление к следующему за ним оператору.

**Операторы цикла:**

* loop

loop

x:= x + 1;

exit when x > 5;

end loop;

* for

for x in 1..5

loop

…

end loop;

* while

while (x > 0)

loop

x:= x - 1;

end loop;

* **EXIT** – завершение цикла, когда дальнейшая обработка нежелательна или невозможна. Внутри цикла можно помещать один или большее количество операторов EXIT. По оператору EXIT цикл завершается немедленно и управление переходит к следующему за END LOOP оператору.
* **EXIT-WHEN** – цикл завершиться только в том случае, когда становится истинным условие в предложении WHEN

## **39. Курсоры. Виды курсоров. Схемы обработки курсора.**

**Курсор** – объект БД, который позволяет работать с записями построчно.

**Типы курсоров:**

* явный – объявляется пользователем
* неявный – не требует объявления

Курсор можетс возвращать одну, несколько или ни одной строки.

**Неявный курсор** – выполнение SQL выражения в секции исполнения или в секции исключений блока. Не требуют объявления, не требуют OPEN, FETCH, CLOSE.

Когда неявный курсор не возвращает строк вообще, PL/SQL генерирует исключение NO\_DATA\_FOUND и передает управление в секцию исключений.

Когда SELECT возвращает более одной строки, PL/SQL генерирует исключение TOO\_MANY\_ROWS и также передает управление в секцию исключений.

SELECT INTO предназначен исключительно для того, чтобы возвращать ровно 1 строку – точную выборку.

**Виды курсоров:**

1. Статические – выражение для них определяется на этапе компиляции, используются для DML команд, могут быть явно объявлены и именованы
2. Динамические – выражение для них определяется на этапе выполнения, могут использоваться для любых SQL выражений, включая DDL и DCL команды, реализуются с помощью оператора EXECUTE IMMEDIATE или пакета dbms\_sql

**Операторы управления явным курсором:**

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

**Этапы открытия явного курсора:**

1. разбор
2. связывание переменных
3. выполнение
4. построение плана выполнения запроса
5. связывание внешних переменных и курсорных параметров
6. определение набора данных
7. выставление указателя текущей строки на первую строку в результирующем наборе данных

## **40. Курсоры. Атрибуты курсора. Курсоры с параметрами.**

Атрибуты курсора:

* %ISOPEN — возвращает значение TRUE, если курсор открыт, для неявного всегда FALSE
* %FOUND — определяет, найдена ли строка, удовлетворяющая условию, NULL перед выполнением, TRUE, если одна или более строк были вставлены, изменены, удалены или одна строка выбрана, иначе FALSE
* %NOTFOUND — возвращает TRUE, если строка не найдена, NULL перед выполнением, FALSE, если одна или более строк были вставлены, изменены, удалены или одна строка выбрана.
* %ROWCOUNT — возвращает номер текущей строки

Явные курсоры объявляются явно в секции декларации блока или в спецификации пакета, могут объявляться с параметрами или без:

CURSOR company\_\_cur (id\_in IN NUMBER) IS

SELECT name FROM company WHERE company\_id = id\_in;

…

for rec in company\_\_cur (10)

loop …

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

Курсорные переменные – структуры данных, которые указывают на курсорный объект.

Используются для:

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



Курсорная переменная, объявленная с помощью REF CURSOR без указания RETURN может быть связана с любым запросом.

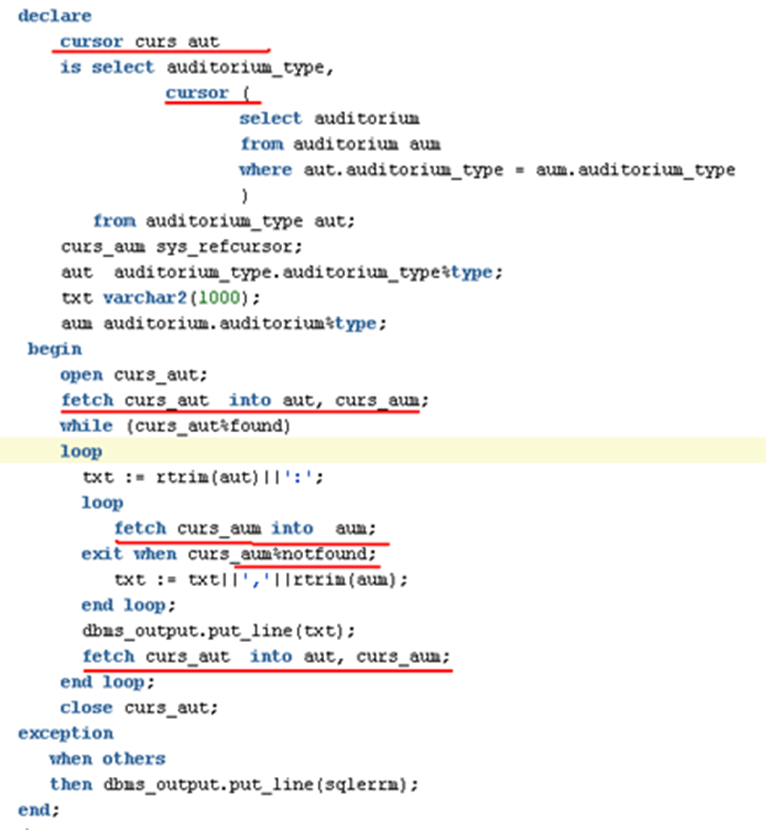
Курсорная переменная, объявленная с помощью REF CURSOR с указанием RETURN может быть связана только с запросом, который возвращает результат точно соответствующий числу и типам данных в записи после фразы RETURN во время выполнения.

Параметры Oracle, связанные с курсорами:

* cursor\_space\_for\_time = {TRUE|FALSE} – больший объем памяти для курсоров и никогда не освобождается. Применяется для увеличения скорости работы курсоров при наличии памяти для разделяемого пула.
* cursor\_sharing = {EXACT|SIMILAR|FORCE}
* open\_cursors – максимальное количество открытых курсоров
* session\_cached\_cursors – максимальное количество кэшируемых курсоров для сессии

## **42. Курсоры. Курсорные подзапросы.**

Курсор в курсоре.



## **43. Курсоры. Использование конструкции CURRENT OF в курсорах.**

CURRENT OF используется если нужно получить текущую строку, выбранную в курсоре

…

if cty > 90

then

cty := cty \* 1.1;

update auditorium set auditorium\_capacity = cty where current of curs\_auditorium;

end if;

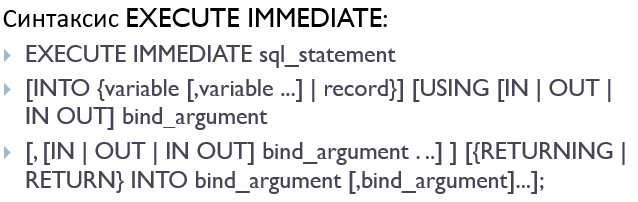
…

## **44. Курсоры. Динамические курсоры.**

Динамические курсоры – выражение для них определяется на этапе выполнения, могут использоваться для любых SQL выражений, включая DDL и DCL команды, реализуются с помощью оператора EXECUTE IMMEDIATE или пакета dbms\_sql.

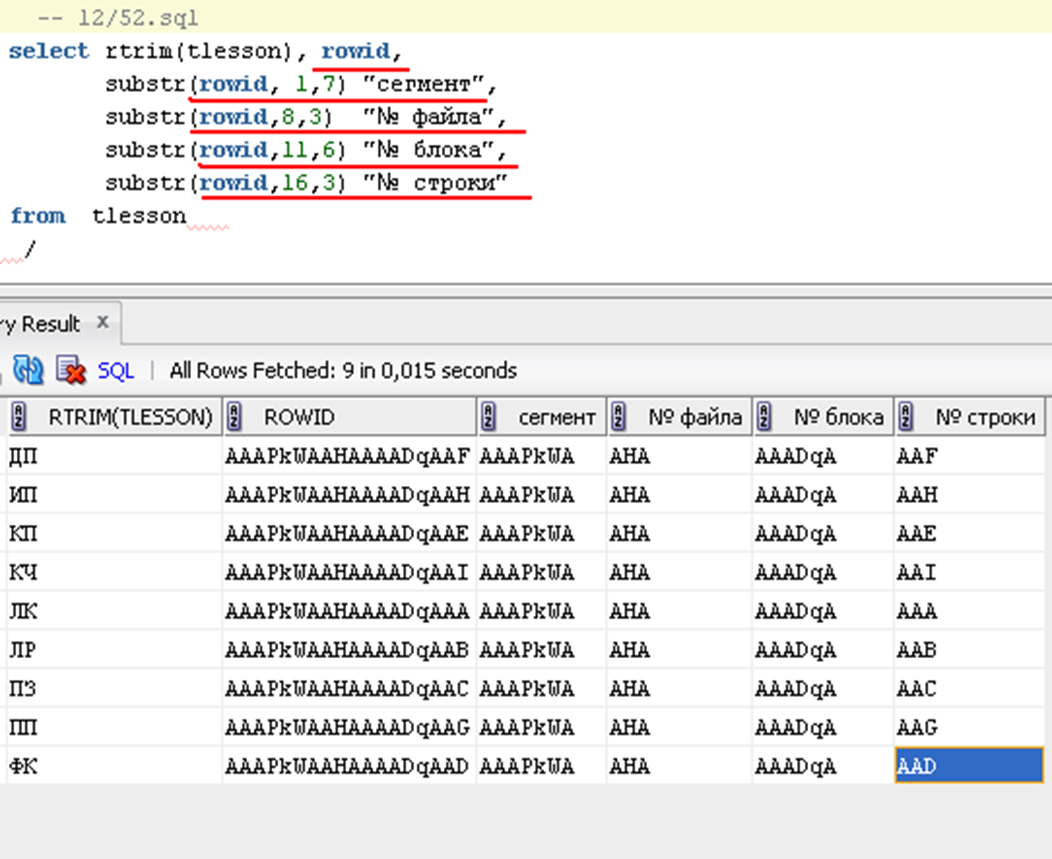
EXECUTE IMMEDIATE – однострочные запросы и DDL команды

OPEN, FOR, FETCH, CLOSE – динамические многострочные запросы



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

**ROWID** – уникальный идентификатор строки в таблице, то же применение что и current of, но в select.



**ROWNUM** – логический номер записи в запросе, для ограничения количества возвращаемых значений.



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

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

Типы исключений:

1. программно-определяемые
2. предопределенные (стандартные)

Секция исключений – необязательная секция в PL/SQL блоке, которая содержит один или несколько обработчиков исключения.

Обработка исключений – перехват ошибки в секции исключений

Область действия – часть кода, в рамках которого может быть сгенерировано исключение

Необработанное исключение – исключение становится необработанным, если оно не обработано блоком самого верхнего уровня.

Неименованное исключение – исключение, которое имеет код ошибки и сообщение, но не имеет наименования, не может быть использовано в команде RAISE или в секции WHEN

Именованное исключение – исключение с именованием.

**Стандартные исключения:**

* NOT\_LOGGED\_ON
* NO\_DATA\_FOUND
* TOO\_MANY\_ROWS
* INVALID\_NUMBER
* CURSOR\_ALREADY\_OPEN
* DELETE\_FAILED
* и т.д.

sqlerrm – сообщение ошибки

sqlcode – код ошибки

**Именованные исключения:**

* объявление исключения, которое не относится к определенным сервером: exception\_name EXCEPTION;
* имена исключений могут быть использованы только для генерации исключения при помощи RAISE и для перехвата исключения в секции WHEN

**Генерация исключения:**

… if n = 0 then raise ERROR\_NAME; end if;

**Обработка исключения:**

блок exception

…

exception

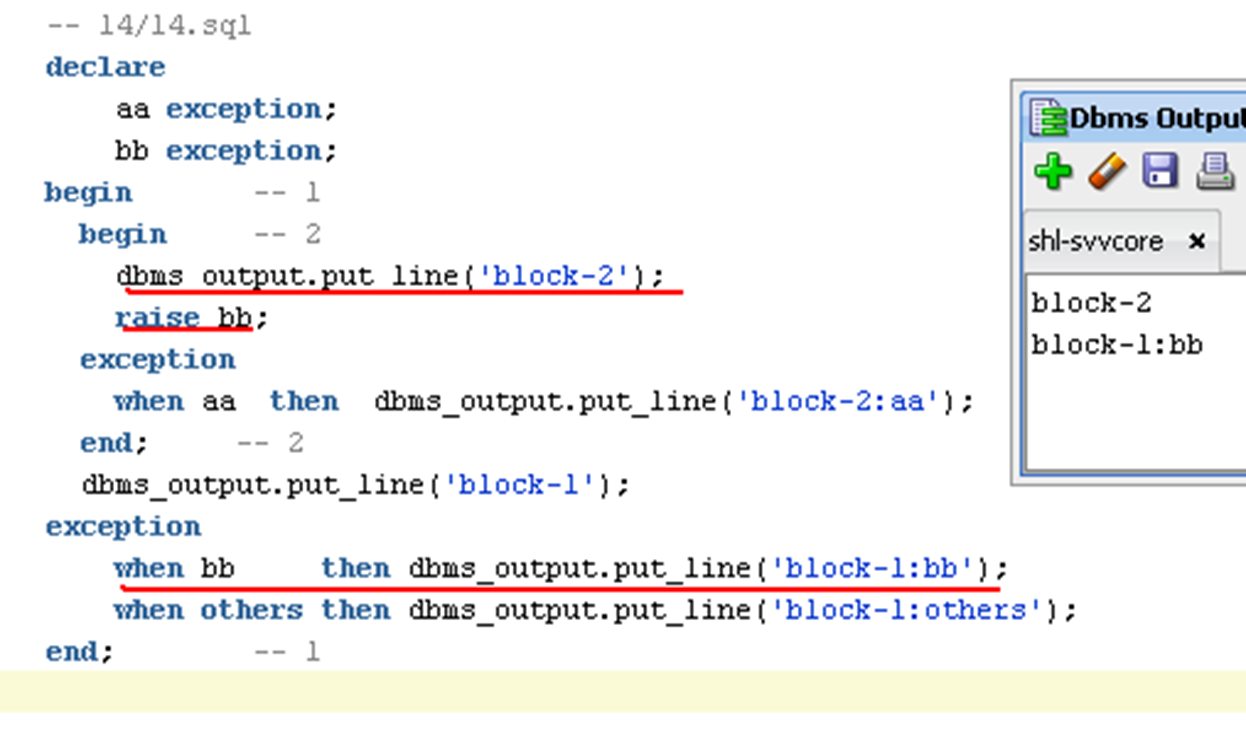
when no\_data\_found then …

when others then …;

end;

## **47. Принцип распространения исключений в PL/SQL. Инструкция RAISE\_APPLICATION\_ERROR.**

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



RAISE\_APPLICATION\_ERROR:

* определена в пакете DBMS\_STANDART
* можно присвоить сообщение об ошибке
* при выполнении процедуры:
* выполнение блока прерывается
* любые изменения в аргументах IN OUT и OUT откатываются
* изменения в глобальных структурах (пакетные переменные, объекты БД) не откатываются – для отката выполнить явно ROLLBACK

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

**Встроенные функции:**

* числовые
* символьные
* функции по работе с датами
* конвертирование
* функции обработки ошибок

**Числовые функции:**

* abs - модуль
* ceil - округление вверх
* round - округление до определенного количества знаков после запятой
* trunc - усечение до определенного количества знаков после запятой
* floor - возвращает наибольшее целое значение, равное или меньшее, чем число
* remainder - остаток от деления
* mod
* bitand (expr1, expr2) - возвращает целое число, представляющее побитовую операцию AND над битами expr1 и expr2
* cos, acos, sin, asin, tan, atan
* exp
* power
* sqrt
* log
* sign - возвращает значение, определяющее знак числа (<0, то -1; >0, то 1; 0, то 0; null, то null)
* greatest - возвращает наибольшее значение в списке параметров

**Символьные функции:**

* ascii - возвращает числовое представление крайнего левого символа строки
* asciistr - преобразует строку любого набора символов к ASCII строке, используя набор символов базы данных
* chr - возвращает символ, который основан на числовом коде, противоположна ascii
* unistr - преобразует в национальный формат
* concat - соединение строк
* initcap - устанавливает первый символ каждого слова в верхний регистр, а остальные в нижний регистр
* instr - возвращает n-е вхождение подстроки в строке

**Функции для работы с датами:**

* current\_date - возвращает текущую дату в часовом поясе текущей сессии SQL как установлено с помощью команды ALTER SESSION
* current\_timestamp - возвращает текущую дату и время в часовом поясе текущей сессии SQL как установлено с помощью команды ALTER SESSION. Она возвращает дату/время со значением часового пояса.
* sysdate - возвращает текущую системную дату и время на вашей локальной базе данных.
* localtimestamp - возвращает текущую дату и время в часовом поясе из текущей сессии SQL, как установлено командой ALTER SESSION. Это возвратит значение TIMESTAMP.
* next\_day - возвращает первый день недели, который больше date
* last\_day - возвращает последний день месяца на основе значения даты
* dbtimezone - возвращает часовой пояс базы данных как смещения часового пояса (в следующем формате: '[+|-]TZH:TZM') или название региона часового пояса
* extract(year/month/day from date ‘yyyy-mm-dd’) - извлекает значение из даты или значения интервала
* months\_between - возвращает количество месяцев между date1 и date2

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

REGEXP\_LIKE(исходная\_строка, шаблон[, параметр\_сопоставления]) - Используется подобно оператору like в части where или же при определении ограничения на таблицу (constraint) .

REGEXP\_INSTR(исходная\_строка, шаблон[, начальная\_позиция [, вхождение [, опция\_возврата [, параметр\_сопоставления ] ] ] ] ) - Функция возвращает позицию символа, находящегося в начале или конце соответствия для шаблона, так же как и ее аналог instr.

REGEXP\_SUBSTR(исходная\_строка, шаблон[, позиция [, вхождение [,параметр\_сопоставления]]]) - Функция REGEXP\_SUBSTR возвращает подстроку, которая соответствует шаблону.

REGEXP\_REPLACE(исходная\_строка, шаблон [, строка\_замены [, позиция[,вхождение, [параметр\_сопоставления]]]]) - REGEXP\_REPLACE возвращает измененную входную строку, в которой все вхождения шаблона заменены значением, переданным в параметре строка\_замены.

## **50. Коллекции. Массивы переменной длины.**

**Коллекция** – структура данных, содержащая элементы одного типа. Элементом коллекции может быть как скалярная величина, так и композитные данные. Элементы коллекций можно сравнивать между собой на эквивалентность. Можно передавать параметром. Одномерная, но можно создавать коллекции коллекций.

Коллекция состоит из **набора** элементов, причем каждый элемент находится в определенной позиции (имеется **индекс** элемента). Необходимо объявить **тип коллекции** – командой TYPE. Необходимо объявить **коллекцию** – переменную этого типа для дальнейшего использования.

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

Коллекции типа VARRAY всегда ограничены

Вложенные таблицы и ассоциативные массивы неограничены.

**Объявление коллекций**

*Инициализация коллекций*

* Явно с помощью конструктора
* Неявно при выборке из базы данных
* Прямым присвоением переменной с другой коллекции такого же типа

*Добавление и удаление элементов*

* Вложенные таблицы и массивы переменной длины – сначала увеличить размер при помощи функции EXTEND, а затем присвоить значения новым элементам
* Ассоциативный массив – присвоение значения новому элементу

**Массивы переменной длины** – одномерные, связанные коллекции однотипных элементов. Доступны в рамках PL/SQL и в БД. Являются плотными

declare

type myarraytype is varray(3) of number

va myarray := myarraytype(null, null, null)

begin

for i in 1..va.count() loop

va(i) := 1;

end loop;

end;

va.extends - инициализирует доп ячейку в массиве (до n, объявленных при создании типа)

## **51. Коллекции. Вложенные таблицы.**

**Вложенные таблицы** – одномерные, несвязанные коллекции однотипных элементов. Доступны в рамках PL/SQL и как поля таблицы в БД. Изначально являются плотными, но могут впоследствии становиться разреженными.

declare

type mytabletype is table of number

tb mytabletype := mytabletype ()

begin

for i in 1..va.count() loop

tb.extend

tb(i) := 1;

end loop;

end;

## **52. Коллекции. Ассоциативные массивы.**

**Ассоциативные массивы** – одномерные, неограниченные (по максимальному количеству элементов при создании) коллекции элементов. Доступны только в рамках PL/SQL. Изначально являются разреженными, индекс может принимать непоследовательные значения.

declare

type tperson is record (name teacher.teacher\_name%type, pulpit teacher.pulpit%type)

type mytabletype is table of tpersonindex by teacher.teacher\_name%type

tb mytable;

begin

tb(‘УРБ’).name := ‘Урбанович П. П,’

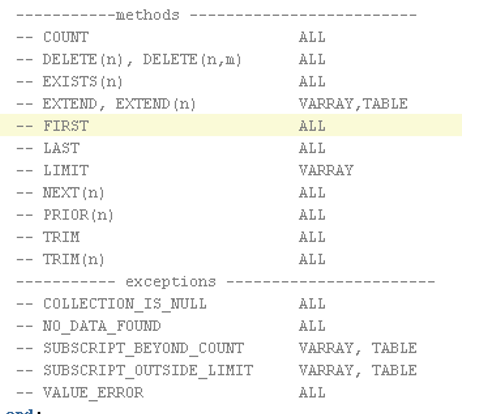
tb(‘УРБ’).pulpit := ‘ИСиТ’

end loop;

end;

tb.first - возврат первого элемента массива

tb.next(o) - возврат следующего после o элемента массива



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

**Хранимая процедура** — это определенный набор инструкций, написанных на языке PL/SQL. Вызов процедуры приводит к выполнению содержащихся в ней инструкций. Процедура хранится в базе данных, поэтому и называется хранимой. Процедура состоит из двух основных частей: спецификации и тела*.*

**Спецификация процедуры (procedure specification)** включает в себя имя процедуры и описание ее входных и выходных данных. Эти входные и выходные данные называются формальными параметрами (formal parameters) или формальными аргументами (formal arguments). Если при вызове процедуры указываются параметры командной строки или другие входные данные, эти значения называются фактическими (actual) параметрами или фактическими аргументами.

При создании процедуры или функции, вы можете определить три типа параметров, которые могут быть объявлены:

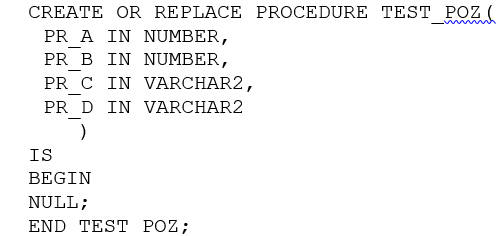
**IN** – Параметр может ссылаться на процедуру или функцию. Значение параметра не может быть изменено процедурой или функцией.

**OUT** – параметр не может ссылаться на процедуру или функцию, но значение параметра может быть изменено процедурой или функцией.

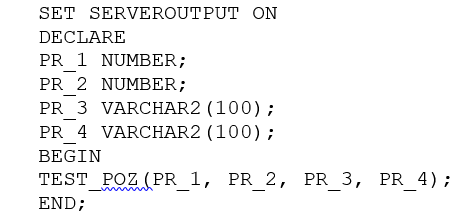
**IN OUT** – Параметр может ссылаться на процедуру или функцию и значения параметра может быть изменено процедурой или функцией.

**Позиционный формат** - когда мы передаем параметры строго в указанном при создании процедуры порядке

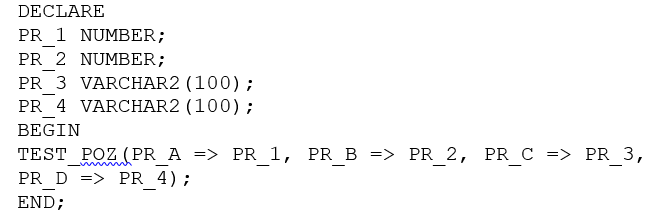
Например, есть процедура:



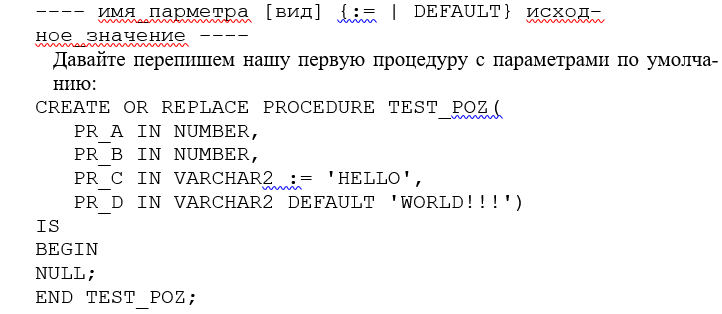
Тогда позиционный формат:

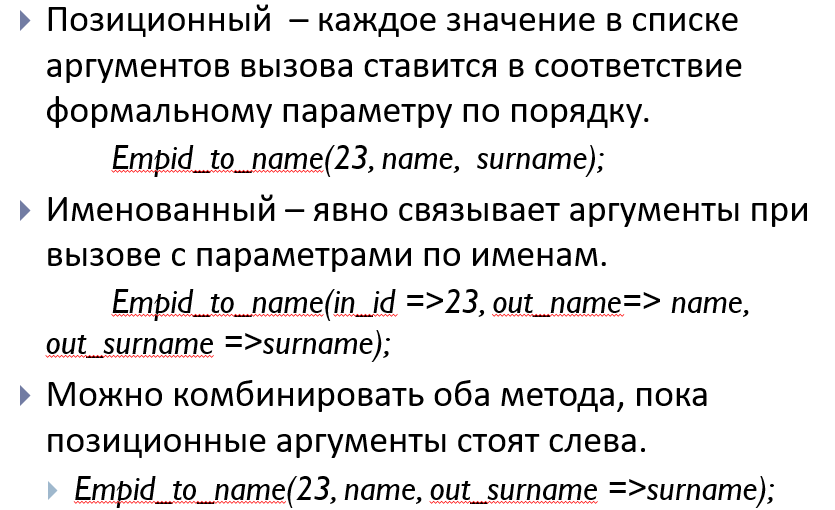


**Параметрический формат** - когда мы передаем не в указанном порядке, а по имени параметра:



**Значение параметров по умолчанию.** Дело в том, что как и все переменные формальные параметры процедуры могут иметь значения по умолчанию. В таком случае значение параметру, имеющему такое определение можно не передавать. Если же фактический параметр все-таки передан, то принимается именно его значение. Итак, значение по умолчанию указывается вот так:





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

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

Главное различие между процедурой и функцией в том, что функция предназначена для возврата значения, которое может использоваться в более крупном SQL-Операторе.

Функция вызывается в анонимном блоке.

Функция называется детерминированной тогда, когда для одних и тех же входных аргументов она возвращает один и тот же результат.

**Pipeline** - это конвейерная табличная функция. Конвейерная обработка отменяет надобность в создании огромных наборов, передавая строки по каналу из функции по мере их создания, сохраняя память и позволяя запустить последующую обработку еще до окончания генерации всех строк. Конвейерные табличные функции включают фразу **PIPELINED** и используют вызов **PIPE ROW**, чтобы вытолкнуть строки из функции, как только они создадутся, вместо построения табличной коллекции. **Конвейерными функциями** называются табличные функции, которые возвращают данные в виде коллекции, но делают это асинхронно, то есть получена одна запись коллекции и сразу же эта запись отдается в вызывающий код в котором она сразу же обрабатывается. В этом случае память сохраняется, простой по времени ликвидируется.

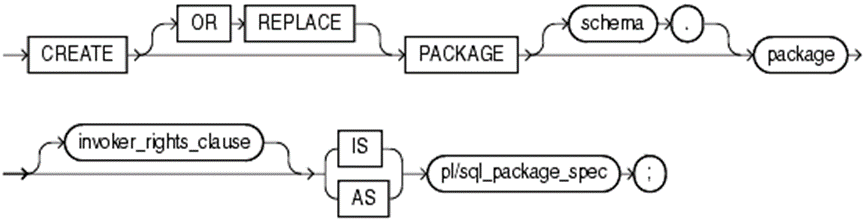
## **55. Процедурные объекты. Пакеты. Спецификация и реализация пакета. ХЗ нужен ли весь этот код**

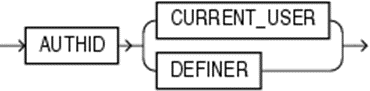
**Пакеты** - коллекция PL/SQL объектов, сгруппированных вместе.

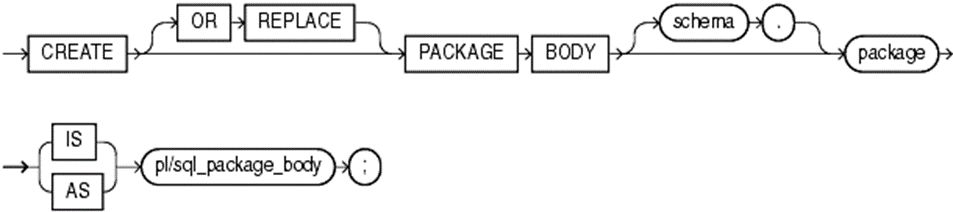
В пакет можно включать: **процедуры, функции, константы, исключения, курсоры, переменные, TYPE выражения, записи, REF курсоры.**

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

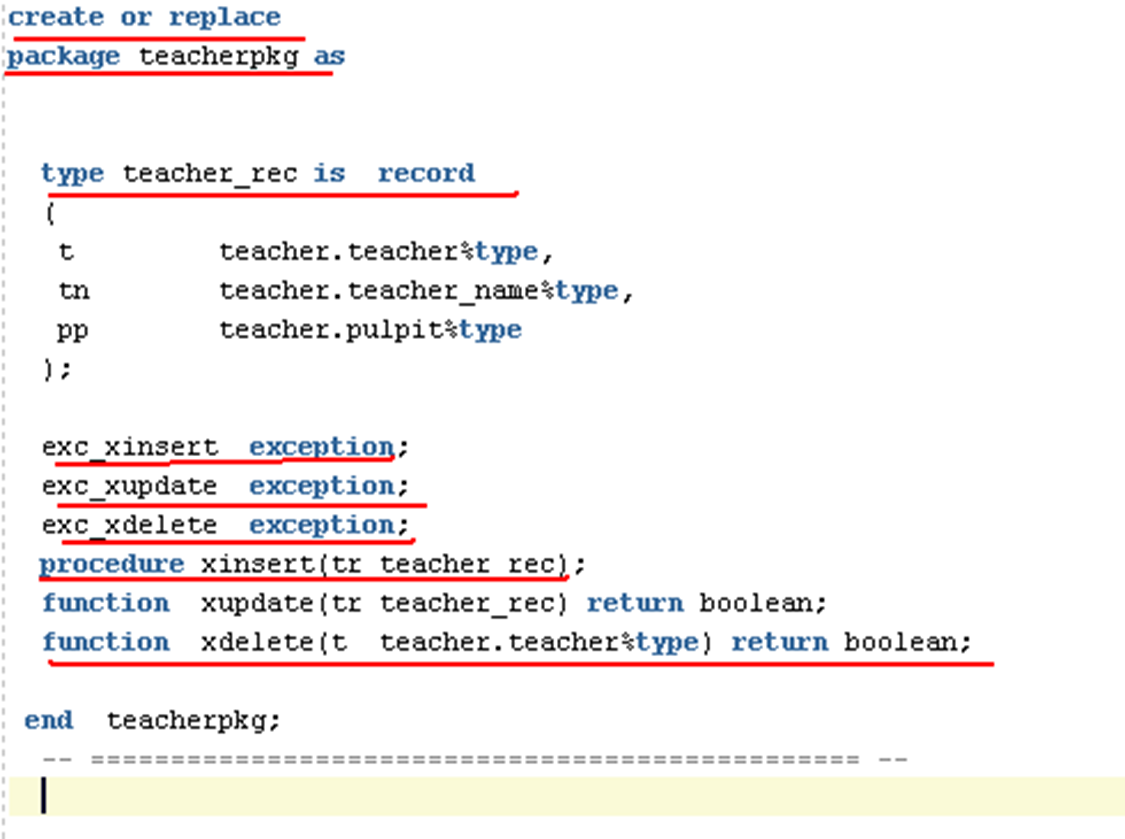
**Реализация пакета (package body)** - содержит весь программный код для реализации процедур и функций и спецификации, приватные объекты и секцию инициализации.



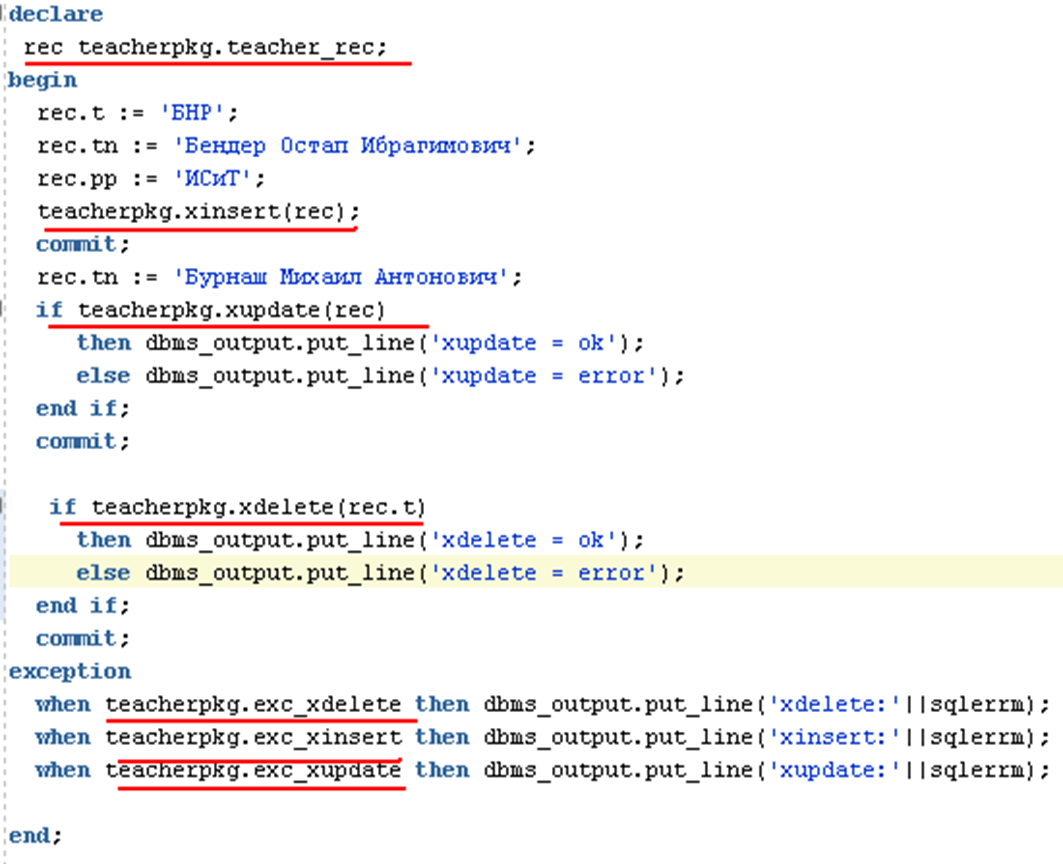




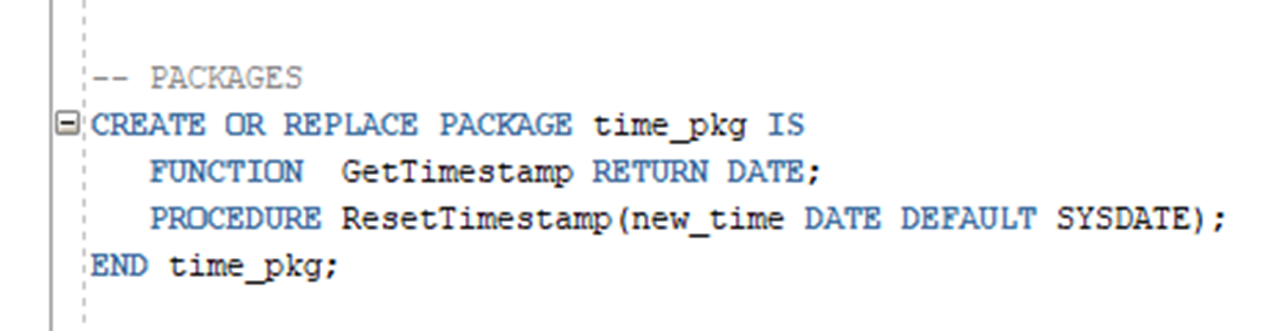
**Пример заголовка пакета**

****

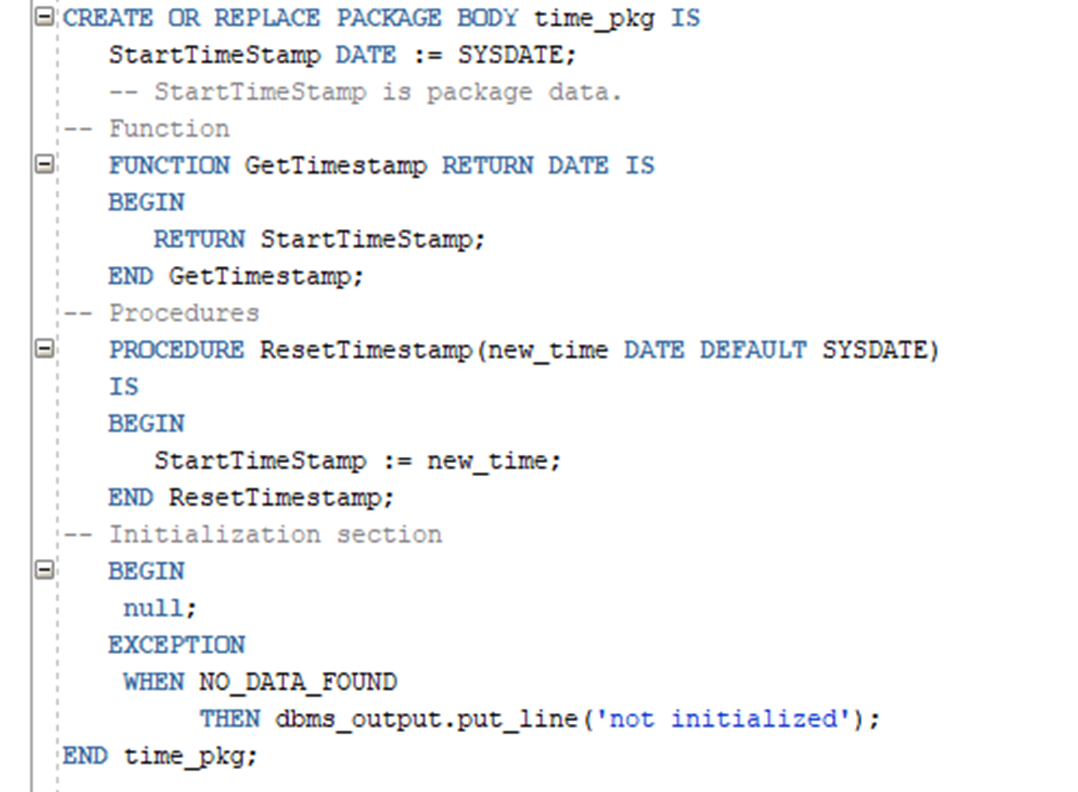
**Пример использования пакета:**

****

**Пример спецификации пакета**

****

**Пример реализации пакета:**

****

**Пример использования пакета:**

****

* **Вызов пакета: Package\_name.pachkage\_element;**
* **Пакетные данные -** структуры данных, объявленные в пакете
* Пакетные переменные сохраняют свое состояние от одной транзакции к другой и являются глобальными данными

## **56. Процедурные объекты. Триггеры. Виды триггеров. Классификация, порядок выполнения и предикаты триггеров. Триггеры замещения. Привилегии. Включение/отключение триггеров. Псевдозаписи old и new.**

Триггер - механизм, вызываемый в момент выполнения одного из DML операторов (insert, delete, update).

Виды триггеров:

* before (вызывается перед выполнением DML оператора)
* instead of (вызывается вместо выполнения DML оператора)
* after (вызывается после выполнения DML оператора)

Триггеры также могут быть строчные for each row (вызываются при изменении оператором каждой строки) или операторные (вызываются один раз на один оператор).

Порядок выполнения триггеров:

1. before оператор
2. before строка
3. after строка
4. after оператор

Необходимые привилегии:

* grant create trigger
* grant alter trigger
* grant create any trigger

Включение/отключение триггера - alter trigger [имя триггера] enable/disable

Псевдозаписи old и new используются для получения измененных данных.

## **57. Секционирование таблиц. Виды секционирования.**

Секционирование - метод, позволяющий хранить сегмент данных, такой как таблица, в виде отдельных сегментов, сохраняя логическую монолитную структуру (...\_tables, …\_part\_tables, …\_tab\_patritions, …\_segments) .

select \* from tablename partition (partitionname)

* **Диапазонное секционированние**

**partition by range (column)**

**(**

**partition partitionname1 values less than (...),**

**partition partitionname2 values less than (...),**

**partition partitionname3 values less than (maxvalue)**

**);**

***Недостаток:***

При загрузке новых данных в таблицу нужно постоянно расщеплять секцию maxvalue

* **Интервальное секционирование**

**partition by range (column)**

**interval (100) store in dict**

**( partition partname values less than (101)**

**);**

Создается единственная секция без maxvalue

Новые секции будут создаваться автоматически

* **Хэш-секционирование**

**partition by hash(column)**

**partitions 2^n**

**store in (partname1, partname2, partname3, partname4, …)**

* **Списочное секционирование**

**partition by list (column)**

**(**

**partition partitionname1 values (..., …, …)**

**partition partitionname2 values (..., …, …)**

**partition partitionname3 values (default)**

**);**

* **Композитное секционирование**

При секционировании большой таблицы сами секции могут оказаться достаточно крупными. Такие секции могут секционировать по другому критерии секционированию. Допускается 2 уровня секционирования (секции и подсекции).

**partition by range (column1)**

**subpartition by hash (column2)**

**subpartitions 4 store in (..., …, …, …)**

**(**

**partition partname1 values less than (),**

**partition partname2 values less than ()**

**);**

**Эквисекционирование -** объекты эквисекционированы, если они секционированы одинаковым образом.

* **Ссылочное секционирование**

Выполняется, если таблицы явным образом связаны ссылочными ограничениями целостности. Манипуляции с секциями главной таблицы автоматически отражаются на секциях подчиненной.

**partition by reference (refcolumn)**

•Секционирование повышает производительность обработки данных

•Отбрасывание секций, параллелизм, балансировка дисковой нагрузки

•Секционирование упрощает управление крупными объектами хранения

•Крупный объект разбивается на индивидуально-управляемые части

•Секционирование обеспечивает дополнительную надежность системы

•Сокращение времени обслуживания, нечувствительность к сбоям

## **58. Транзакции. Виды транзакций. Понятие автономной транзакции.**

Транзакция – группа операторов, которые либо все выполняются, либо все не выполняются.



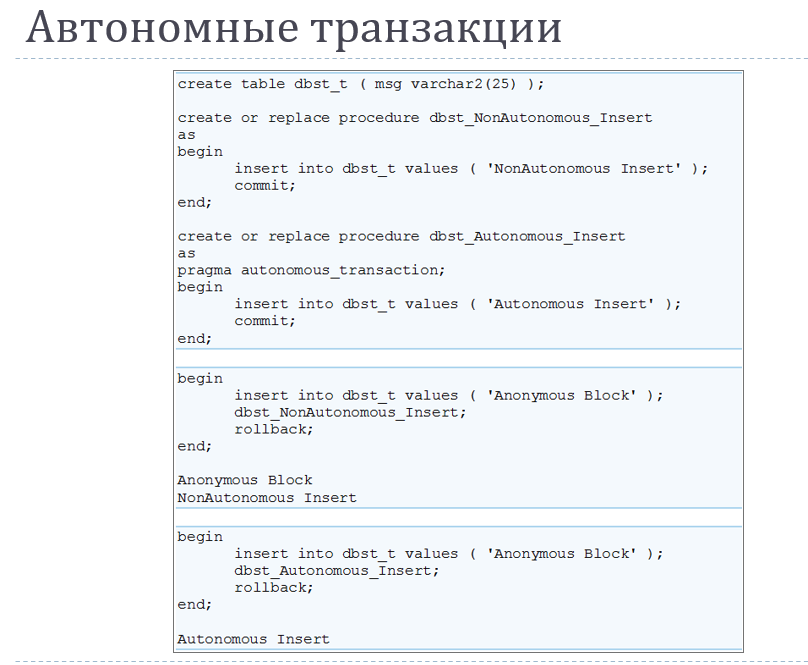
Грязное чтение – чтение одной транзакцией промежуточных данных другой транзакции перед тем, как она их зафиксирует

Невоспроизводимое (неповторяющееся) чтение – при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются измененными

Фантомное чтение – при повторном чтении в рамках одной транзакции одна и та же выборка дает разные множества строк

**Автономная транзакция** – независимая транзакция, начатая другой транзакцией, которая является главной. В блоке автономной транзакции главная транзакция приостанавливается. Вы выполня­ете SQL-операции, затем производите их фиксацию или откат и возобновляете главную транзакцию.

Чтобы определить блок как автономную транзакцию, нужно включить в раздел объявлений директиву PRAGMA AUTONOMOUS\_TRANSACTION



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

**DBMS\_JOB –** поддержка управления заданиями

**Задание –** процедура, Pl-SQL блок, внешняя процедура

Выполняется в фоновом режиме, надо задать количество одновременно выполняемых процессов (**alter system set job\_queue\_processes = 9)**

**create or replace procedure procedjob is**

**begin**

**end;**

**declare job\_number user\_jobs.job%type;**

**begin**

**dbms\_job.submit(job\_number, ‘begin procedjob; end;’, sysdate, ‘sysdate + 60/86400’);**

**commit;**

**end;**

**dbms\_job.run(43);**

**dbms\_job.change(43, null, null, ‘sysdate + 360/86400’);**

**dbms\_job.next\_date(43, sysdate + 2/24);**

**dbms\_job.interval(43, sysdate + 3):**

**dbms\_job.broken(43, true, null);**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**dbms\_scheduler.create\_schedule(**

**schedule\_name =>**

**start\_date =>**

**repeat\_intervall => ‘freq=daily’**

**comments =>**

**);**

## **60. Системные пакеты Oracle.**

* **APEX** – Oracle Application Express – среда разработки веб-приложения
* **APEX\_CUSTOM\_AUTH** –проверка подлинности и управления сеансом
* **APEX\_APPLICATION** – использование глобальных переменных
* **APEX\_ITEM** – создание элементов форм на основе SQL-запроса
* **APEX\_UTIL** – различные утилиты состояния сеанса, файлов, авторизации и пр.
* **DBMS\_ADVANCED\_REWRITE** – перехват и замена SQL-запросов
* **DBMS\_ADVISOR** – часть набора экспертной системы для решения проблем производительности, связанных с компонентами сервера базы данных
* **DBMS\_SQLTUNE** – сбор статистики, используется при анализе производительности SQL-операторов
* **DBMS\_APPLICATION\_INFO** – присвоение имени процессу для удобства мониторинга и отладки
* **DBMS\_AQ** – пакет для Advanced Queuing – система обмена очередями сообщений
* **DBMS\_TRANSFORM** – пакет для Oracle Advanced Queuing – преобразования данных
* **DBMS\_DEFER** – вызов удаленных процедур
* **DBMS\_OFFLINE\_OG, DBMS\_REPCAT** – поддержка репликации
* **DBMS\_REFRESH** – создание группы материализованных представлений для обновления как единого целого
* **DBMS\_BACKUP\_RESTORE** – пакет для создания резервных копий для управляющих файлов, журналов повтора, архивных файлов
* **DBMS\_CRYPTO** – пакет шифрования данных

Поддерживаются алгоритмы шифрования - DES, AES, RC4, 3DES, 3DES-2KEY

* **DBMS\_CUBE** - поддерживает развертывание кубических материализованных представлений из существующих реляционных материализованных представлений
* **DBMS\_DIMENSION** - отображение информации об измерениях – иерархических связях между данными
* **Data Warehouse**  - предметно-ориентированная база данных, предназначенная для отчетов и/или OLAP
* **DBMS\_DEBUG** – пакет для отладки PL/SQL-кода на сервере
* **DBMS\_DDL** - эквиваленты команд DDL, которые нельзя использовать в PL/SQL непо­средственно:
* **DBMS\_DESCRIBE** – позволяет получить информацию о процедурах
* **DBMS\_FGA** – пакет настройки и использования политик аудита:

## 

# Список примерных задач к экзамену по АБДиП/БД

# Задачи попроще (в одной экзаменационной задаче может быть несколько связанных задач из этого списка):

--1 Определите размеры областей памяти SGA.

select \* from v$sga;

select \* from v$sga\_dynamic\_components;

--2 Получите список всех параметров экземпляра.

select \* from v$parameter;

show parameter instance;

--3 Получите список управляющих файлов.

select \* from v$controlfile;

--4 Сформируйте PFILE.

show parameter spfile;

create pfile = 'test\_exam\_pfile.ora' from spfile = 'SPFILEORCL.ORA';

--or

create pfile = 'test\_exam\_pfile\_from\_memory.ora' from memory;

--5 Создайте таблицу из двух столбцов, один из которых первичный ключ. Получите перечень всех сегментов. Вставьте данные в таблицу. Определите, сколько в сегменте таблицы экстентов, их размер в блоках и байтах.

create table T1(

n1 number(3) primary key,

ch1 nvarchar2(4)

);

create table T2(

n2 number(3),

ch2 nvarchar2(4),

constraint fk\_T1 foreign key (n2) references T1(n1)

);

drop table T1;

drop table T2;

declare

i number(3) :=1;

max\_i number(3) := 998;

begin

while i <= max\_i

loop

insert into T1 values (i, dbms\_random.string('x', 4));

i := i + 1;

end loop;

commit;

i := 1;

max\_i := 500;

while i <= max\_i

loop

insert into T2 values (dbms\_random.value(1,999), dbms\_random.string('x', 4));

i := i + 1;

end loop;

end;

declare

i number(3) :=1;

max\_i number(3) := 500;

begin

while i <= max\_i

loop

insert into T2 values (dbms\_random.value(1,998), dbms\_random.string('x', 4));

i := i + 1;

end loop;

end;

select \* from t1;

select \* from t2;

select segment\_name, segment\_type, extents, blocks, bytes from user\_segments where segment\_name in ('T1', 'T2');

-- 6 Получите перечень всех процессов СУБД Oracle. Для серверных процессов укажите режим подключения. Для фоновых укажите работающие в настоящий момент.

select \* from v$bgprocess where paddr != '00'; -- активные

select \* from v$process; -- вообще все

select name, network\_name, pdb from v$services; -- активные подключения

select username, status, server from v$session where username is not null; --активные серверные подключения

--прикол в самих процессах, ибо серверный процесс по сути является основой для пользовательского

select username, status, server from v$session where server = ‘DEDICATED’ -- режим подключения - выделенный

--7 Получите перечень всех табличных пространств и их файлов.

select \* from dba\_tablespaces;

select tablespace\_name, file\_name from dba\_data\_files;

select \* from dba\_temp\_files;

--8 Получите перечень всех ролей.

select \* from dba\_roles;

--9 Получите перечень привилегий для определенной роли.

select granted\_role, privilege from dba\_role\_privs pr

join dba\_sys\_privs p on pr.grantee = p.grantee

where granted\_role = 'RL\_SYMCORE';

select \* from dba\_role\_privs;

select \* from dba\_sys\_privs;

--10 Получите перечень всех пользователей.

select \* from dba\_users;

--11 Создайте роль

create role RL\_SYMCORE;

--drop role RL\_SYMCORE;

grant create session,

create table,

create procedure,

create view

to RL\_SYMCORE;

--12 Создайте пользователя

create user C##TEST identified by "191279"

default tablespace users

quota unlimited on users;

--13 перечень всех профилей безопасности

select distinct profile from dba\_profiles;

--14 параметры профиля безопасности

select \* from dba\_profiles where profile = 'PF\_SYMCORE';

--15 Создание профиля безопасности

create profile PF\_SYMCORE limit

password\_life\_time 180

sessions\_per\_user 3

failed\_login\_attempts 7

password\_lock\_time 1

password\_reuse\_time 10

password\_grace\_time default

connect\_time 180

idle\_time 30

--16 Создайте последовательность S1, со следующими характеристиками: начальное значение 1000; приращение 10; минимальное значение 0; максимальное значение 10000; циклическую; кэширующую 30 значений в памяти; гарантирующую хронологию значений. Создайте таблицу T1 с тремя столбцами и введите (INSERT) 10 строк, со значениями из S1.

create sequence S1

increment by 10

start with 1000

maxvalue 10000

minvalue 0

cycle

cache 30

order;

--drop sequence S1;

create table T1 (C1 number(10), C2 number(10), C3 number(10))

declare

val1 int;

val2 int;

val3 int;

max\_i int := 10;

begin

for i in 0..max\_i

loop

select S1.nextval into val1 from dual;

select S1.nextval into val2 from dual;

select S1.nextval into val3 from dual;

insert into T1 values (val1, val2, val3);

end loop;

commit;

end;

select \* from T1;

--17 Создайте частный и публичный синоним для одной из таблиц и продемонстрируйте его область видимости. Найдите созданные синонимы в представлениях словаря Oracle.

create synonym LS for T3;

create public synonym PS for T3;

select \* from dba\_synonyms where table\_name = 'T3';

--18 Разработайте анонимный блок, демонстрирующий возникновение и обработку исключений WHEN TO\_MANY\_ROWS и NO\_DATA\_FOUND.

select \* from orders;

declare

order\_row orders%rowtype;

begin

select \* into order\_row

from orders;

dbms\_output.put\_line(order\_row.order\_num || ' ' || order\_row.order\_date);

exception

when too\_many\_rows

then dbms\_output.put\_line('Error number: ' ||sqlcode);

dbms\_output.put\_line('Error text: ' ||sqlerrm);

end;

declare

order\_row orders%rowtype;

begin

select \* into order\_row

from orders

where order\_num = '000000000';

dbms\_output.put\_line(order\_row.order\_num || ' ' || order\_row.order\_date);

exception

when no\_data\_found

then dbms\_output.put\_line('Error number: ' ||sqlcode);

dbms\_output.put\_line('Error text: ' ||sqlerrm);

end;

--19 Получите перечень всех групп журналов повтора.

select \* from v$log;

--20 Определите текущую группу журналов повтора.

select \* from v$log where status = 'CURRENT';

--21 Получите перечень файлов всех журналов повтора.

select \* from v$logfile;

--22 Создайте таблицу и вставьте в нее 100 записей. Найдите таблицу и ее свойства в представлениях словаря.

create table T4 (x number(3));

--drop table T4;

declare

i number(3) :=1;

max\_i number(3) := 100;

begin

while i <= max\_i

loop

insert into T4 values (dbms\_random.value(1,998));

i := i + 1;

end loop;

end;

select \* from T4;

select \* from user\_segments where segment\_name = 'T4';

select \* from dba\_segments where segment\_name = 'T4';

select \* from user\_objects where object\_name = 'T4';

select \* from user\_tables where table\_name = 'T4';

--23 Получите список сегментов табличного пространства.

select segment\_name from dba\_segments where tablespace\_name = 'SYSTEM';

--24 Подсчитайте размер данных в таблице.

select bytes from dba\_segments where segment\_name = 'T4';

--25 Вычислите количество блоков, занятых таблицей.

select blocks from dba\_segments where segment\_name = 'T4';

--26 Выведите список сессий.

select \* from v$session;

--27 Выведите, производится ли архивирование журналов повтора.

select log\_mode from v$database;

--28 Создайте представление с определенными параметрами.

тут я малость не поняла что за представление они хотят, ибо для параметризованного нужен отдельный пакет, а мы это не проходили

--скорее всего имеется в виду создайте для таблицы X с таким-то предикатом…

вперед

--29 Создайте database link с определенными параметрами.

тут тоже. с какими параметрами?

--30 Продемонстрируйте эскалацию исключения.

declare

order\_row orders%rowtype;

begin

select \* into order\_row

from orders;

begin

dbms\_output.put\_line(order\_row.order\_num || ' ' || order\_row.order\_date);

exception

when too\_many\_rows

then dbms\_output.put\_line('привет из внутреннего блока');

end;

exception

when too\_many\_rows

then dbms\_output.put\_line('привет из внешнего блока');

end;

# Задачи среднего уровня:

не осуждайте женщину за отсутствие фантазии в названиях процедур и функций….

1. Создайте процедуру, которая выводит список заказов и их итоговую среднюю стоимость для определенного покупателя. Параметр – наименование покупателя. Обработайте возможные ошибки.

| create or replace procedure orderd\_cost(cust\_name customers.company%type)  is  cursor cust\_orders is select order\_num, amount from orders o  join customers cust on o.cust = cust.cust\_num  where cust.company = cust\_name;  amount\_sum DECIMAL(9,2) := 0;  res cust\_orders%rowtype;  begin  for res in cust\_orders  loop  dbms\_output.put\_line(res.order\_num|| ' ' || res.amount);  amount\_sum := amount\_sum + res.amount;  end loop;  dbms\_output.put\_line('Sum: ' || amount\_sum);  exception  when others then  dbms\_output.put\_line('exception :' || sqlerrm);  end;  begin  orderd\_cost('JCP Inc.');  end;  -- блять, я конечно всё понимаю, но не понимаю:  1) зачем fetch, 2) when no\_data\_found наивно юзать с явными курсорами, у меня вроде бы не прокало, но рекомендую потестить.  я пыталась хуйню одну сделать и забыла убрать. Ты сам то веришь, чтобы я fetch в for написала?  обидно кстати да…  Ну бывает такое, да  qty - количество которое заказали  amount - цена за все  amaunt = qty \* price  тут ещё не до конца четко описано что есть qty, что amount, я считал cost как их произведение, но я думаю ей не принципиально  а  не, так не надо там все достаточно логично  price в таблице товаров  ты ж связи посмотри  извиняю  У меня вот:  то есть price = amount/qty? невероятно, ладно, тогда извиняюсь, зря быканул  create or replace procedure ord\_cust\_1(comp\_name customers.company%type) **is**  cursor orders\_curs **is**  select order\_num, company, amount from orders inner join customers on cust\_num = cust  where company = comp\_name;    curs\_row orders%rowtype;  cost\_sum int := 0;  rows\_count int := 0;  begin  for curs\_row in orders\_curs  loop  dbms\_output.put\_line(curs\_row.order\_num || ' amount: ' || curs\_row.amount);  rows\_count := rows\_count + 1;  cost\_sum := cost\_sum + curs\_row.amount;  end loop;    if cost\_sum =0 then  dbms\_output.put\_line('no orders');  else  dbms\_output.put\_line('avg: ' || cost\_sum / rows\_count);  end if;    exception  when others then dbms\_output.put\_line(sqlerrm);  end;  begin  ord\_cust\_1('J.P. Sinclair');  end;  -- с увожением ||<3 |
| --- |

2. Создайте функцию, которая подсчитывает количество заказов покупателя за определенный период. Параметры – покупатель, дата начала периода, дата окончания периода.

| create or replace function orders\_count( cust\_name customers.company%type, start\_date orders.order\_date%type, end\_date orders.order\_date%type)  return number is results number(3);  begin  select count(\*) into results from orders o  join customers cust on o.cust = cust.cust\_num  where cust.company = cust\_name and order\_date between start\_date and end\_date;  return results;  end;  select orders\_count('JCP Inc.', to\_date('17.07.07'), to\_date('01.11.08')) from dual; |
| --- |

3. Создайте процедуру, которая выводит список всех товаров, приобретенных покупателем, с указанием суммы продаж по убыванию. Параметр – наименование покупателя. Обработайте возможные ошибки.

| **цена по которой продали товар или цена самого заказа спросим завтра**  create or replace procedure cust\_orders(cust\_name customers.company%type)  is  cursor cur is select distinct product, p.price from orders o  join products p on p.product\_id = o.product  join customers cust on cust.cust\_num = o.cust  where cust.company = cust\_name  order by price desc;    res cur%rowtype;  begin  open cur;  fetch cur into res;  while cur%found  loop  dbms\_output.put\_line(res.product || ' ' || res.price);  fetch cur into res;  end loop;  close cur;  exception  when others then  dbms\_output.put\_line('exception :' || sqlerrm);  end;  begin  cust\_orders('JCP Inc.');  end; |
| --- |

4. Создайте функцию, которая подсчитывает количество заказов за определенный период. Параметры – дата начала периода, дата окончания периода.

| select \* from orders;  create or replace function orders\_count(start\_date orders.order\_date%type, end\_date orders.order\_date%type)  return number is results number(3);  begin  select count(\*) into results from orders  where order\_date between start\_date and end\_date;  return results;  end;  select orders\_count(to\_date('17.07.07'), to\_date('01.11.08')) from dual; |
| --- |

5. Создайте процедуру, которая выводит список покупателей, в порядке убывания общей стоимости заказов. Параметры – дата начала периода, дата окончания периода. Обработайте возможные ошибки.

| create or replace procedure customers\_desc(start\_date orders.order\_date%type, end\_date orders.order\_date%type)  is  cursor cur is select company, sum(amount) amount\_sum from customers cus  join orders o on o.cust = cus.cust\_num  where order\_date between start\_date and end\_date  group by company  order by sum(amount) desc;    res cur%rowtype;  begin  for res in cur  loop  dbms\_output.put\_line(res.company || ' ' || res.amount\_sum);  end loop;  exception  when others then  dbms\_output.put\_line('exception :' || sqlerrm);  end;  begin  customers\_desc(to\_date('17.07.07'), to\_date('01.11.08'));  end; |
| --- |

6. Создайте функцию, которая подсчитывает количество заказанных товаров за определенный период. Параметры – дата начала периода, дата окончания периода.

| create or replace function prod\_count(start\_date orders.order\_date%type, end\_date orders.order\_date%type)  return number is results number(3);  begin  select distinct count(\*) into results from orders o  join products p on p.product\_id = o.product  where order\_date between start\_date and end\_date;  return results;  end;  -- справочная информация от ||<3: to\_date(‘dd.mm.yy’) не обязателен, преобразование неявное, везде где встречается  select prod\_count('17.07.07','01.11.08') from dual;  Не всегда. У оракла бывает шиза с датами и пока явно не приведешь- работать не будет. Саша подтвердит, у неё таблицы не заполнялись без явного приведения. |
| --- |

7. Создайте процедуру, которая выводит список покупателей, у которых есть заказы в этом временном периоде. Параметры – дата начала периода, дата окончания периода. Обработайте возможные ошибки

| create or replace procedure cust(start\_date orders.order\_date%type, end\_date orders.order\_date%type)  is  cursor cur is select distinct company from customers cus  join orders o on o.cust = cus.cust\_num  where order\_date between start\_date and end\_date;  res cur%rowtype;  begin  for res in cur  loop  dbms\_output.put\_line(res.company);  end loop;  exception  when others then  dbms\_output.put\_line('exception :' || sqlerrm);  end;  begin  cust(to\_date('17.07.07'), to\_date('01.02.08'));  end;  ну, я очень вскользь, средние задачи вроде простые, но вот тут например: скорее всего если ты крутой, то when others не совсем достаточно, а вот неплохо было бы например if (end\_date < start\_date) чекнуть  сек, где-то есть у меня, но дело не совсем в рейзах, а просто в том что тут просто нужно чекать 2 вещи по идее: отсутствие заказов и неправильные параметры, но 1 хватит по идее.  Согласен, да, думал об этом. ну, можно всегда на похуй в селект добавить count, или как у меня переменная row\_count.  Ну щас чекну  raise самый простой вариант: raise\_application\_error(-20001,'Credit is too high');  пишите дети raise\_application\_error и будьте счастливы  пишем кастомный эхепшн через прагму :)  дитя ООП артем орлов хуйни не посоветует  все я спать  завтра теорию еще вычитать стоило бы, а то там весело  артем это я тебе говорю помогать будешь  ;0  declare  exception\_name EXCEPTION;  PRAGMA EXCEPTION\_INIT (exception\_name, error\_code);  begin  RAISE exception\_name;  end -- сообщение хочеццааааа  сейчас насчет курсоров чекну  да, походу либо отказываться от for - я не собираюсь, либо заводить доп переменную. Тут как бы справедливо, курсор открыт только внутри фора, а если ничего нет - сразу закроется  здрасьте, вопрос. а почему бы не через while? вместо фор  ставишь вайл до тех пор пока в курсоре есть что-то и оно пашет  Я пробовала в первом задании, но работает хуйово. Придумаешь как слелать красиво - можно будет глянуть.    хуя гусь скажи гагага гагага гуся:  вроде как работает  а смысл? с raise работает криво. тут же прикол не в самом курсоре, а в правильной обработке ошибок.  ну просто если так все задания писать, лишь бы raise работал, то хуйта получается  это часть задания так-то. на when others можно выехать в принципе. я к тому что смысл от более сложной конструкции, если ты все равно не знаешь как ее применить с обработчиком  пиздец блять  нихуя агрессия  глянь остальные, а то я могла и хуйни написать  тут ао всех when others не прокатит нужно делать rase, но я хер знает как их правильно полднимать  по хорошему это должно работать через no\_data\_found но курсор сказал соси  я хотела проверять его на пустоту, но не прокатило  row\_count кстати неплохое, не подумала хотя курсорными переменными должно тоже обрабатываться |
| --- |

8. Создайте функцию, которая подсчитывает количество покупателей определенного товара. Параметры – наименование товара.

| create or replace function cust\_by\_prod(prod\_name products.description%type)  return number as res number;  begin  select count(\*) into res from orders o  join products p on p.product\_id = o.product  where p.description = prod\_name;  return res;  end;  select cust\_by\_prod('Size 4 Widget') from dual; |
| --- |

9. Создайте процедуру, которая увеличивает на 10% стоимость определенного товара. Параметр – наименование товара. Обработайте возможные ошибки

| reate or replace procedure up\_price(product products.product\_id%type)  is  p products.price%type;  begin  select price into p from products where product\_id = product;  dbms\_output.put\_line(p);  update products set price = price + price\*0.1  where product\_id = product;  select price into p from products where product\_id = product;  dbms\_output.put\_line(p);  end;  begin  up\_price('2A45C');  end;  rollback; |
| --- |

10. Создайте функцию, которая вычисляет количество заказов, выполненных в определенном году для определенного покупателя. Параметры – покупатель, год. товара.

| create or replace function orders\_count(cust\_name customers.company%type, order\_year number)  return number is res number;  begin  select count(order\_num) into res from orders o  join customers cus on cus.cust\_num = o.cust  where company = cust\_name  and extract(year from o.order\_date) = order\_year;  return res;  end;  select orders\_count('Acme Mfg.', 2007) from dual; |
| --- |

# Задачи посложнее:

эти завтра, если надо

Сложна

1. Создайте процедуру, которая добавляет заказ. Обработайте возможные ошибки. Создайте триггер, который контролирует целостность данных при добавлении заказа.

| create or replace procedure add\_order\_check\_ingegrity(mfr products.mfr\_id%type, product products.product\_id%type, rep orders.rep%type, cust orders.cust%type) is  elem\_count int;  begin  select count(\*) into elem\_count from products where product\_id = product AND mfr\_id = mfr;  if elem\_count = 0 then  raise\_application\_error(-20001, 'Product does not exist!');  end if;    select count(\*) into elem\_count from customers where cust\_num = cust;  if elem\_count = 0 then  raise\_application\_error(-20001, 'Customer does not exist!');  end if;    select count(\*) into elem\_count from salesreps where rep = empl\_num;  if elem\_count = 0 then  raise\_application\_error(-20001, 'Repository does not exist!');  end if;  end;  create or replace procedure add\_order(mfr products.mfr\_id%type, product products.product\_id%type, rep orders.rep%type, cust orders.cust%type, qty orders.qty%type) is  new\_order\_num int;  price int;  begin  add\_order\_check\_ingegrity(mfr, product, rep, cust);  select max(order\_num) + 1 into new\_order\_num from Orders;  select price into price from products where product\_id = product AND mfr\_id = mfr;  insert into Orders  (order\_num, order\_date, cust, rep, mfr, product, qty, amount)  values (new\_order\_num, sysdate, cust, rep, mfr, product, qty, price\*qty);  end;  create or replace trigger tr\_order\_integrity\_check before insert on Orders for each row  begin  add\_order\_check\_ingegrity(:new.mfr, :new.product, :new.rep, :new.cust);  end;  т.е я зря пыталась проверять все явно и можно было забить хуй и сделать обычный стандартный триггер  exec add\_order('REI', '2A44L', 106, 2117, 12);  exec add\_order('REI', '2A44L', 106, 21317, 12); -- wrong cust  insert into Orders  (order\_num, order\_date, cust, rep, mfr, product, qty, amount)  values (124241, sysdate, 2117, 106, 'aEI', '2A44L', 12, 21412); -- wrong mfr, should proc trigger error  -- комментарий: задание хуйни, откровенно говоря. Начиная с того, что id заказа не является identity, что автоматически посылает меня нахуй, ну и другие мелочи. Тк ошибки по сути обрабатывать нужно в двух местах, я вынес в отдельную процедуру. Алсо возможно нужно обрабатывать qty\_on\_hand, мол, если заказал больше чем на руках - идешь нахуй. (да, нужно)А ещё возможно qty\_on\_hand нужно уменьшать при этом. (тоже нужно) Спасибо заданию за очень подробную информацию по этому вопросу. Вообще говоря задание заняло у меня порядка 45 минут бтв  процедура хуня, делается за 10-15 минут. что с триггером делать не ебу  У меня что-то такое вышло  create or replace procedure add\_order(cust\_name customers.company%type, rep\_name salesreps.name%type, product\_type products.product\_id%type, prod\_qty orders.qty%type)  is  order\_row orders%rowtype;  order\_date order\_row.order\_date%type := sysdate;  product\_price products.price%type;  available\_qty products.qty\_on\_hand%type;    invalid\_qty exception;  begin  order\_row.order\_num := dbms\_random.value(114000, 115999);  select cust\_num into order\_row.cust from customers where company = cust\_name;  select empl\_num into order\_row.rep from salesreps where name = rep\_name;  select mfr\_id into order\_row.mfr from products where product\_id = product\_type;  select price into product\_price from products where product\_id = product\_type;  select qty\_on\_hand into available\_qty from products where product\_id = product\_type;  if prod\_qty > available\_qty then  raise invalid\_qty;  end if;  insert into orders values(order\_row.order\_num, order\_date, order\_row.cust, order\_row.rep, order\_row.mfr, product\_type, prod\_qty, prod\_qty\*product\_price);  update products set qty\_on\_hand = qty\_on\_hand - prod\_qty where product\_id = product\_type;  dbms\_output.put\_line('Заказ №' || order\_row.order\_num || ' добавлен.');  exception  when invalid\_qty then  dbms\_output.put\_line('Не хватает товара. Доступное для заказа количество: ' || available\_qty);  when no\_data\_found then  dbms\_output.put\_line('Проверьте правильность введенных данных.');  when others then  dbms\_output.put\_line('exception :' || sqlerrm);  end;  begin  add\_order('JCP Inc.', 'Sam Clark', 'XK47 ', 4);  end;  кстати за использование id продавца и покупателя дадут по жепе. за id товара не дадут, ибо описания повторяются |
| --- |

2. Создайте функцию, которая возвращает количество заказов покупателя помесячно за определенный период. Параметры – покупатель, дата начала периода, дата окончания периода. Обработайте возможные ошибки.

-- звучит кстати как pipeline функция, которая возвращала бы что-то типо

звучит как хуйня ибо у меня получается выводить только в одну ячейку

пайплайн возможно сработает, но я не пробовала. Зато пробовала псевдокурсором. хуета  
12 | 01.2001  
46 | 02.2001  
15 | 03.2001

результат верный, но некрасивый

| create or replace function orders\_by\_month(cust\_name customers.company%type, start\_date orders.order\_date%type, end\_date orders.order\_date%type)  return sys\_refcursor is result\_table sys\_refcursor;  begin  open result\_table for  select extract(month from order\_date), count(\*) from orders o  join customers cus on o.cust = cus.cust\_num  where cus.company = cust\_name  and o.order\_date between start\_date and end\_date  group by (extract(month from order\_date));  return result\_table;  end;  select orders\_by\_month('JCP Inc.', to\_date('17.07.07'), to\_date('01.11.08')) from dual; |
| --- |

Гигачад выкатывает пипелайновую функцию:

| --пайплайн не обязателен, можно просто вернуть курсор, потверждено Блиновой  grant create any type to C##OAACORE;  create type t\_month\_count as object (  month\_date date,  orders\_count int  );  create type t\_month\_count\_arr is table of t\_month\_count;  create or replace function orders\_by\_date return t\_month\_count\_arr pipelined as  cursor curs is  select  trunc(order\_date, 'MONTH') as month\_date,  count(\*) as orders\_count  from orders  group by trunc(order\_date, 'MONTH');  type t\_arr is table of curs%rowtype;  arr t\_arr;  begin  open curs;  fetch curs bulk collect into arr;  for i in 1..arr.Count  loop  PIPE ROW(t\_month\_count(arr(i).month\_date, arr(i).orders\_count));  end loop;  end;  select \* from TABLE(orders\_by\_date()) order by month\_date; |
| --- |

3. Создайте процедуру, которая выводит в консоль список всех товаров, не приобретенных ни одним покупателем в определенном году по убыванию количества на складе. Параметр – год. Обработайте возможные ошибки.

4. Создайте функцию, которая подсчитывает количество заказов покупателя за определенный год. Параметры – год, часть имени покупателя или код.

5. Создайте процедуру, которая сортирует таблицу по определенному столбцу Параметры – название столбца, порядок сортировки (ASC, DESC). Обработайте возможные ошибки.

-- копировать сортированный селект в массив, удалить из таблицы, вставить массив? -- Уже написал потерпите

| -- Для будущих поколений, в такой задаче лучше спросить чо от вас хотят, т.к. всё может закончиться тем что вас попросят просто dbms\_output сделать  create or replace procedure sort\_orders\_by(colname nvarchar2 default 'ORDER\_DATE', sort\_order nvarchar2 default 'ASC') is  type order\_arr is table of orders%rowtype;  arr order\_arr;  cur\_row orders%rowtype;  curs SYS\_REFCURSOR;  query\_str varchar2(60);  begin  query\_str := ('select \* from orders order by ' || colname || ' ' || sort\_order);  dbms\_output.put\_line(query\_str);  open curs for query\_str;  fetch curs bulk collect into arr;  close curs;    delete from orders;  FORALL i IN 1..arr.COUNT  insert into orders values arr(i);  end;  exec sort\_orders\_by('ORDER\_DATE', 'DESC');  select \* from orders; |
| --- |

вообще задачки выглядят не так ебано, но не на sql

# 