|  |
| --- |
| **Лабораторная работа №15**  **СУБД Oracle. ТЕСТ** |
| Настройка SQL\*Plus. Необходимо включить режим ECHO и вывести протокол лаб. работы в файл *<Фамилия студента>.txt*. Этот файл является отчетом о проделанной лаб. работе.  Пример:  Следующий пример включает режим ECHO и ведет файл протокола spool. txt  SQL> set echo on  SQL> spool c:\spool. txt  Задание №2.  <https://testserver.pro/run/test/1451/>  <https://onlinetestpad.com/ru/test/1799615-postgresql-10-dba1>  (тест DBA)  **Вопрос №1**  Дана таблица **DEPT** со следующей структурой:   |  |  |  | | --- | --- | --- | | **Name** | **NULL** | **Type** | | DEPTNO | NOT NULL | Number(25) | | DNAME |  | VARCHAR2(14) | | LOC |  | VARCHAR2(13) |   Приведён фрагмент кода:  https://testserver.pro/img/big/i-4L4.gif  Какое PL/SQL выражение отобразит местонахождение выделенного отдела?  DBMS\_OUTPUT.PUT\_LINE(dept\_rec.loc);+  DBMS\_OUTPUT.PUT\_LINE(dept\_rec(1).loc);  Не возможно отображение отдельных полей записи, так как они не были заданы в секции DECLARE  DBMS\_OUTPUT.PUT\_LINE(dept\_rec);  DBMS\_OUTPUT.PUT\_LINE(dept\_rec(0).)loc;  Пакет DBMS\_OUTPUT чаще всего используется для простого вывода информации на экран. трибут *%ROWTYPE* позволяет объявлять переменную типа "запись", соответствующую строке таблицы. Переменная такого типа имеет поля, совпадающие с полями таблицы по имени и типу.Такой тип значительно облегчает программирование операций со строками, позволяя выполнять выборку строки целиком в одну переменную типа "запись", а также предотвращает необходимость перепрограммирования *блоков* в случае изменения структуры таблицы.  В Oracle PL/SQL **атрибут %ROWTYPE** предоставляет тип записи, представляющий строку в таблице (или представлении) базы данных Oracle. Запись может хранить целую строку данных, выбранных из таблицы, или извлекаться из курсора или строго типизированной переменной курсора. Например:  var1 REAL(14,2);  var2 var1%TYPE;  - Переменная var2 будет иметь тип как var1  var\_f1 user1.tbl1.f1%TYPE;  /\* Переменная var\_f1 будет иметь тот же  тип, что и поле f1 таблицы tbl1  пользователя user1\*/  **Вопрос №2**  В базе данных были созданы пакет emp\_pack и процедура edit\_emp:  https://testserver.pro/img/big/i-4G1.gif  Вы внесли изменения в тело пакета emp\_pack и перекомпилировали его. Какое из следующих утверждений является верным?  Процедура emp\_pack станет недостоверной и будет перекомпилирована при первом вызове  Статус процедуры edit\_emp не изменится +  Процедура emp\_pack станет недостоверной и будет перекомпилирована при втором вызове  Спецификация пакета emp\_pack станет недостоверной и будет перекомпилирована при первом вызове  Спецификация пакета emp\_pack станет недостоверной и будет перекомпилирована при втором вызове  ***Пакет*** - это объект схемы, который объединяет логически зависимые типы PL/SQL, данные и *подпрограммы*. *Пакет* состоит из двух частей: *спецификации пакета* и тела *пакета* .  В *спецификации пакета* объявляются доступные типы, переменные, константы, исключения, курсоры и *подпрограммы*.  В теле *пакета* содержится определение курсоров и реализация *подпрограмм*. Все элементы, объявляемые в теле *пакета*, невидимы для приложения, что позволяет скрывать от пользователя детали реализации *подпрограмм*.  *Определение спецификации пакета* выполняется оператором *CREATE PACKAGE* , который может иметь следующее формальное описание:  - Спецификация (видимая часть)  CREATE PACKAGE name AS  - Объявление общедоступных типов  - и переменных  - Спецификация подпрограмм  END [name];  *Определение тела пакета* выполняется оператором *CREATE PACKAGE BODY* , который может иметь, с некоторыми сокращениями, следующее формальное описание:  - Тело пакета (скрытая часть)  CREATE PACKAGE BODY name AS  - Объявление локальных типов  - и переменных  - Тела подпрограмм  END [name];  **Вопрос №3**  Для осуществления контроля за пользователями был создан следующий триггер:  https://testserver.pro/img/big/i-4J2.gif  Каким должен быть тип триггера для эффективного решения поставленной задачи?  Уровня выражения (statement level)  INSTEAD OF  BEFORE  Уровня строки (row level)+  Oracle Forms триггер  *Триггер* - это выполняемый *модуль*, привязанный к *объекту базы данных* и событию, связанному с этим объектом. *Триггер* вызывается неявно при возникновении события над этим объектом. *Триггеры* имеют следующие характеристики -   * Тип триггера - *DDL* или *DML* * Объект - таблица, *VIEW*, *системный объект* для *DDL* *триггеров* * Событие - *insert*, *update*, *delete* для таблицы и *DML*, instead of для *VIEW* или *системное событие* для *DDL* *триггеров*.   Способ активации - для всего *оператора* или для каждой строки for each row   * Время активации - до или после выполнения *оператора*.   *Триггеры* в T-*SQL* по функциональности беднее *триггеров* в *Oracle*. В *SQL Server* существуют только after или instead of *триггеры*, вызываемые для всего *оператора*.  **Получение информации о триггерах**  select \* from user\_triggers;  select \* from user\_objects;  create or replace trigger ta\_ud1  after update  on test  for each row  begin  dbms\_output.put\_line('trigger update 1');  end;  **Вопрос №4**  В таблице **emp** хранится следующая информация о служащих: фамилия, номер подразделения, должность и зарплата. Вы не знаете, в каком регистре сохраняются строки, но хотите извлечь всех менеджеров с зарплатой больше 1000 и меньше 2500, отсортировав их по фамилии в порядке возрастания. Какой оператор позволит решить поставленную задачу?  https://testserver.pro/img/big/i-4JQ.gif  https://testserver.pro/img/big/i-4JR.gif  https://testserver.pro/img/big/i-4JP.gif  https://testserver.pro/img/big/i-4JS.gif+  https://testserver.pro/img/big/i-4JO.gif  Выполните манипуляции 3,2,1 1,2,3 2,3,1  SQL> select last\_name, email, hire\_date  2 from employees  3 order by 3, 2, 1  4 /  **Вопрос №5**  Следуя нижеперечисленным утверждениям можно уменьшить количество ошибочных перекомпиляций, возникающих в результате зависимостей. Какое из утверждений является неверным?  Объявлять записи используя атрибут %ROWTYPE  Использовать запросы с SELECT \*  Обращаться к функциям как имя\_пакета.имя\_функции +  Включать список полей в оператор INSERT  Объявлять переменные используя атрибут %TYPE  Обратите внимание на то, что при совпадении имени схемы и имени пакета может возникать путаница, например, если вы создадите пакет **scott** с процедурой **list**, а в схеме **scott** есть процедура с именем **list**, то вызов функции **scott.list** будет неоднозначным. Вывод: называйте свои пакеты не тривиально (например, с префиксами – в стандартном пакете Oracle это “**DBMS\_**”).  Функция может вызываться в любом месте программы, где допустимо использование выражения того же типа, а именно:   * В операторе присваивания:   **sales04 := tot\_sales(2004, ‘C’);**   * При задании значения по умолчанию:   **DECLARE**  **sales03 NUMBER DEFAULT tot\_sales(2003, ‘C’);**  **BEGIN**  **. . .**   * В логическом выражении:   **IF tot\_sales(2002, ‘C’) > 10000 THEN**  **. . .**   * В SQL-выражении:   **SELECT first\_name, surname**  **FROM sellers**  **WHERE tot\_sales(2001, ‘C’) > 1000**  **Вопрос №6**  Таблица **EMPLOYEES** содержит следующие значения:   |  |  | | --- | --- | | **EMPLOYEE\_ID** | **COMMISSION\_PCT** | | 145 | 0,4 | | 146 | 0,3 | | 147 |  | | 148 | 0,3 | | 149 | 0,2 |   Каким будет результат представленного ниже запроса:  https://testserver.pro/img/big/i-4I5.gif  Внимание! Ответ будет дан сразу при выборе одного из предложенных вариантов  0.4  0,24+  Ошибка  NULL  0,3  Функция AVG возвращает среднее значение выражения.  Чтобы преобразовать null-значение в фактическое значение, используйте **функцию NVL**.  СИНТАКСИС  NVL (*expr1*, *expr2*)  NVL (*expr1*, *expr2*)  В этом синтаксисе:   * *expr1* является исходным значением или выражением, которое может содержать null * *expr2* является целевым значением для того, чтобы преобразовать null   Можно использовать функцию NVL, чтобы преобразовать любой тип данных, но возвращаемое значение всегда является тем же самым, как и тип данных *expr1*.  ПРЕОБРАЗОВАНИЯ NVL ДЛЯ РАЗЛИЧНЫХ ТИПОВ ДАННЫХ   |  |  | | --- | --- | | **Тип Данных** | **Пример Преобразования** | | NUMBER | NVL(number\_column,9) | | DATE | NVL(date\_column, '01-JAN-95') | | CHAR or VARCHAR2 | NVL(character\_column, 'Недоступно') |   **Вопрос №7**  Каким будет результат выполнения приведенной ниже команды:  https://testserver.pro/img/big/i-4LR.gif  Эна очень деиннч ссч+  Эна очень длеиннч стросч  Эна нчень деиннч ссч  Эна очень еиннч сч  Эна нчень длеиннч стросч  Функция Oracle/PLSQL REPLACE заменяет последовательность символов в строке другим набором символов.  **Синтаксис**  Синтаксис функции Oracle/PLSQL REPLACE:  REPLACE( string1, string\_to\_replace, [ replacement\_string ] )  **Параметры или аргументы**  string1 строка для замены последовательности символов другим набором символов.  string\_to\_replace строка, которая будет искаться в string1.  replacement\_string не является обязательным. Все вхождения string\_to\_replace будут заменены string\_to\_replace в string1. Если параметр string\_to\_replace опущен, то функция REPLACE просто удалит все вхождения string\_to\_replace, и вернет получившуюся строку.  SQL> SELECT REPLACE('123123abcd', '123') FROM DUAL;    --Результат:   abcd    SQL> SELECT REPLACE('123abcd123', '123') FROM DUAL;    --Результат:   abcd    SQL> SELECT REPLACE('222abcd', '2', '3') FROM DUAL;    --Результат:   333abcd    SQL> SELECT REPLACE('0000123', '0') FROM DUAL;    --Результат:   123    SQL> SELECT REPLACE('0000123', '0', ' ') FROM DUAL;    --Результат:   123  Функция Oracle/PLSQL TRANSLATE заменяет последовательность символов в строке другим набором символов. Тем не менее, она заменяет один символ за один раз.  Например, заменится первый символ в string\_to\_replace с первого символа в replacement\_string. Тогда будет заменен второй символ в string\_to\_replace с вторым символом в replacement\_string, и так далее.  **Синтаксис**  Синтаксис функции Oracle/PLSQL TRANSLATE:  TRANSLATE( string1, string\_to\_replace, replacement\_string )  **Параметры или аргументы**  string1 строка для замены последовательности символов с другим набором символов.  string\_to\_replace строка, которая будет искаться в string1.  replacement\_string все символы в string\_to\_replace будут заменены на соответствующие символы в replacement\_string.  Функция TRANSLATE возвращает string значение.  **Применение**  **Функцию TRANSLATE** можно использовать в следующих версиях Oracle/PLSQL:   * Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i, Oracle 8i   **Пример**  Рассмотрим несколько примеров функции TRANSLATE и изучим, как использовать **функцию TRANSLATE** в Oracle/PLSQL.  Oracle PL/SQL     |  |  | | --- | --- | | 1  2  3  4  5  6  7 | SQL> SELECT TRANSLATE('1tech23', '123', '456') FROM DUAL;    --Результат:   4tech56    SQL> SELECT TRANSLATE('222tech', '2ec', '3it') FROM DUAL;    --Результат:   333tith |   **Вопрос №8**  Какое из утверждений о пакетах является верным?  Функции объявленные в спецификации пакета не могут быть использованы в SQL операторах  Команда DROP PACKAGE удаляет спецификацию и тело пакета+  Можно удалить функцию из пакета используя команду ALTER PACKAGE  Пакеты могут иметь параметры  Пакеты могут быть вложенными  **Вопрос №9**  В таблице **emp** хранится следующая информация о служащих: фамилия, номер подразделения, должность, зарплата и комиссия. Комиссию получают не все служащие. Вам поручено задание составить отчет содержащий фамилию, номер подразделения, должность и годовой заработок служащих, отсортировав их по номеру подразделения в порядке возрастания, и в порядке убывания зарплаты в каждом подразделении. Какой оператор позволит решить поставленную задачу?  https://testserver.pro/img/big/i-4LN.gif+  https://testserver.pro/img/big/i-4LM.gif  https://testserver.pro/img/big/i-4LO.gif  https://testserver.pro/img/big/i-4LP.gif  https://testserver.pro/img/big/i-4LL.gif  **Вопрос №10**  База данных содержит следующие таблицы и поля: employees(employee\_id, first\_name, last\_name, salary, manager\_id, department\_id), departments(department\_id, department\_name, manager\_id, location\_id). Необходимо составить отчет, содержащий имена всех сотрудников, и названия отделений, в которых они работают. В этот отчет не должны быть включены те сотрудники, которые не приписаны ни к одному из отделений. Однако отчет должен содержать названия отделений без сотрудников. Какой запрос позволит решить поставленную задачу?  https://testserver.pro/img/big/i-4HS.gif+  https://testserver.pro/img/big/i-4HU.gif  https://testserver.pro/img/big/i-4HT.gif  https://testserver.pro/img/big/i-4HV.gif  https://testserver.pro/img/big/i-4HR.gif  **Вопрос №11**  Что из перечисленного ниже НЕ является типом подзапроса?  Коррелируемый  Многострочный  Иерархический +  Многостолбцовый  Однострочный  **Подзапрос** – законченный оператор SELECT, внедрённый в тело другого оператора SELECT. **Внешний**(второй) оператор SELECT использует результат выполнения **внутреннего**(первого) оператора для определения содержания окончательного результата всей операции. Внутренние запросы могут находиться в конструкциях WHERE и HAVING внешнего оператора SELECT — в этом случае они получают название **подзапросов***,*или **вложенных запросов***.*Кроме того, внутренние операторы SELECT могут использоваться в операторах INSERT, UPDATE и DELETE**.**Существуют **три типа подзапросов**.  • **Скалярный подзапрос**возвращает значение, выбираемое из пересечения одного столбца с одной строкой, т.е. единственное значение. В принципе скалярный подзапрос может использоваться везде, где требуется указать единственное значение. **Пример.** Выбрать всех работников, у которых зарплата выше, чем у работника 20.   |  |  |  |  | | --- | --- | --- | --- | | **Запрос** | **Таблица Emp** | | | | SELECT Emp\_id, Dep\_id, salary  FROM emp  WHERE salary >  (SELECT salary  FROM emp  WHERE emp\_id = 20); | Dep\_id | Emp\_id | Salary | | 1 | 10 | 100 | | 1 | 15 | 150 | | 2 | 20 | 200 | | 3 | 30 | 300 |   Подзапрос выбирает значение зарплаты у сотрудника 20, а внешний запрос выбирает всех работников, у которых зарплата больше полученной из подзапроса. Результат подзапроса: 200. Результат всего запроса:   |  |  |  | | --- | --- | --- | | Emp\_id | Dep\_id | Salary | | 30 | 3 | 300 |   • **Строковый (однострочный) подзапрос**возвращает значения нескольких столбцов таблицы, но в виде единственной строки. Примечание: в презентациях преподавателя два вышеприведённых типа рассматриваются как один. Я не вижу смысла приводить пример для этого типа, так как он будет аналогичен примеру ниже с тем лишь отличием, что возвращаться будет лишь одна строка.  • **Табличный (многострочный) подзапрос**возвращает значения одного или нескольких столбцов таблицы, размещенные в более чем одной строке. Табличный подзапрос может использоваться везде, где допускается указывать таблицу, например как операнд предиката IN, ANY, ALL. **Пример.** Выбрать работников, у которых зарплата ниже любой зарплаты сотрудника из департамента 2.   |  |  |  |  | | --- | --- | --- | --- | | **Запрос** | **Таблица Emp** | | | | SELECT Emp\_id, Dep\_id, salary  FROM emp  WHERE salary < ANY  (SELECT salary  FROM emp  WHERE Dep\_id = 2); | Dep\_id | Emp\_id | Salary | | 1 | 10 | 100 | | 1 | 15 | 150 | | 2 | 20 | 200 | | 2 | 25 | 250 | | 3 | 30 | 300 |   Подзапрос выбирает значения зарплат из департамента 2. Их два: 200 и 250. Внешний запрос выбирает тех работников, у которых зарплата меньше максимальной из выбранных подзапросом (поскольку у нас используется условие МЕНЬШЕ и ключевое слово ANY), в данном случае – меньше 250. Результат запроса:   |  |  |  | | --- | --- | --- | | Emp\_id | Dep\_id | Salary | | 10 | 1 | 100 | | 15 | 1 | 150 | | 20 | 2 | 200 |   Подзапросы любого из этих типов могут быть либо **коррелированными** (повторяющимися), либо **некоррелированными**.  **Некоррелированный подзапрос** может вычисляться как независимый запрос. Иначе говоря, результаты подзапроса подставляются в основной оператор (или внешний запрос). Это **не значит**, что SQL-сервер именно**так** выполняет операторы с подзапросами. Некорреляционные подзапросы могут быть заменены соединением и будут выполняться как соединения SQL-сервером. Все примеры запросов выше - некоррелированные.  **Коррелированные подзапросы** не могут выполняться как независимые запросы, поскольку они могут обращаться к данным, находящихся в столбцах таблицы, указанной в списке fromвнешнего запроса. В запросах, которые требуют повторного вычисления подзапроса, результаты возвращаемые подзапросом зависят от значений, передаваемых внешним запросом. В этом случае подзапрос выполняется повторно для каждой строки, которая выбирается во внешнем запросе.  К подзапросам применяютсяследующие **правила** **и ограничения.**  1. В подзапросах не должна использоваться конструкция ORDER BY, хотя она может присутствовать во внешнем операторе SELECT.  2. Список выборки SELECT подзапроса должен состоять из имен отдельных столбцов или составленных из них выражений, за исключением случая, когда в подзапросе используется ключевое слово EXISTS.  3. По умолчанию имена столбцов в подзапросе относятся к таблице, имя которой указано в конструкции FROM подзапроса. Однако разрешается ссылаться и на столбцы таблицы, указанной в конструкции FROM внешнего запроса.  4. Если подзапрос является одним из двух операндов, участвующих в операции сравнения, то подзапрос должен указываться в правой части этой операции.  **Вопрос №12**  База данных содержит следующие таблицы: **EMPLOYEES**   |  |  |  | | --- | --- | --- | | **Имя** | **Null** | **Тип** | | LAST\_NAME | NOT NULL | VARCHAR2(25) | | JOB\_ID | NOT NULL | VARCHAR2(10) | | SALARY | NOT NULL | NUMBER(8,2) | | COMMISSION\_PCT |  | NUMBER(2,2) | | DEPARTMENT\_ID |  | NUMBER(4) | | EMPLOYEE\_ID | NOT NULL | NUMBER(6) |   **DEPARTMENTS**   |  |  |  | | --- | --- | --- | | **Имя** | **Null** | **Тип** | | DEPARTMENT\_ID | NOT NULL | NUMBER(4) | | DEPARTMENT\_NAME | NOT NULL | VARCHAR2(30) | | LOCATION\_ID |  | NUMBER(4) |   **LOCATIONS**   |  |  |  | | --- | --- | --- | | **Имя** | **Null** | **Тип** | | DEPARTMENT\_ID | NOT NULL | NUMBER(4) | | DEPARTMENT\_NAME | NOT NULL | VARCHAR2(30) | | LOCATION\_ID |  | NUMBER(4) |   Какой запрос возвращает среднюю зарплату в отделении Shipping?  https://testserver.pro/img/big/i-4J_.gif  https://testserver.pro/img/big/i-4K-.gif+  https://testserver.pro/img/big/i-4K2.gif  https://testserver.pro/img/big/i-4K0.gif  https://testserver.pro/img/big/i-4K1.gif  **Вопрос №13**  Создана процедура process\_me для обработки данных. Возникла необходимость автоматического выполнения этой процедуры каждый день в 18:00. Какой фрагмент кода позволит решить поставленную задачу?  https://testserver.pro/img/big/i-4L1.gif+  https://testserver.pro/img/big/i-4K_.gif  https://testserver.pro/img/big/i-4L0.gif  https://testserver.pro/img/big/i-4L2.gif  https://testserver.pro/img/big/i-4L-.gif  В Oracle есть специальный механизм запланировать выполнение определённой программы, на заданное время, это может быть оператор SQL, программа на языке PLSQL, либо даже внешняя программа. Этот механизм называется механизмом заданий Job  Данный механизм может применяться, например, для планирования запуска тяжелых запросов в ночное время или на выходные дни, распределения задач построения отчетности.  **Теория и практика**  Для управления заданиями в Oracle существует специальные пакет DBMS\_JOB: Создание заданий Можно использовать механизм dbms\_job Или механизм dbms\_scheduler – более современный способ  DBMS\_JOB.SUBMIT(  JOB OUT BINARY\_INTEGER,  WHAT IN VARCHAR2,  NEXT\_DATE IN DATE DEFAULT SYSDATE,  INTERVAL IN VARCHAR2 DEFAULT NULL,  NO\_PARSE IN BOOLEAN DEFAULT FALSE,  INSTANCE IN BINARY\_INTEGER DEFAULT any\_instance,  FORCE IN BOOLEAN DEFAULT FALSE  );  Здесь: JOB – Входной параметр уникальный идентификатор задания. Идентификатор генерируется специальной системной последовательностью. WHAT - анонимный PL/SQL блок, в данном блоке указывается последовательность команд которая будет выполнена в процессе работы задания. В же в этом параметре можно также писать команды вставки удаления редактирования (inser update delete), а так же команды для создания создания индексов таблиц, создания индексов, ограничений. NEXT\_DATE – дата время следующего выполнения задания. Если будет указана дата меньше чем текущую дата, то выполнение задания будет начато немедленно. INTERVAL – Вычисляемая дата следующего выполнения задания в столбце NEXT\_DATE. Примеры интервала задания: NULL Задание выполнится однократно и удалится.  TRUNC(SYSDATE+1)+10/24  Задание будет выполняться ровно в 10 часов каждого дня  TRUNC(SYSDATE+1)+(11+(15/60))/24  Задание будет выполняться ровно в 11 часов 15 минут каждого дня.  TRUNC(LAST\_DAY(SYSDATE))+(n+(m/60))/24  Задание будет выполняться ровно в n часов m минут последнего дня каждого месяца.  TRUNC(LAST\_DAY(SYSDATE)+1)+(n+(m/60))/24  Задание будет выполняться ровно в n часов m минут первого дня каждого месяца.  NO\_PARSE - флаг разбора PL/SQL выражения. Если его значение равно FALSE разбор происходит в момент установки задания. Иначе, в момент выполнения задания.  INSTANCE - какой экземпляр производит выполнение задания  FORCE – если значение этого параметра истинно тогда в качестве INSTANCE может выступать любое положительное целое число. В противном случае экземпляр, указанный в INSTANCE должен быть обязательно запущен, иначе будет вызвано исключение. Пример2 создания задания  DECLARE jobno NUMBER  BEGIN  DBMS\_JOB.SUBMIT(:jobno,  'DELETE AUTO;',  TO\_DATE('01.01.2015 01:05','DD.MM.YYYY HH24:MI'), 'TRUNC(SYSDATE+1)+(1+(5/60))/24');  COMMIT;  END;  Таблица будет очищаться каждый день в один час пять минут  Пример2 создания задания  DECLARE jobno NUMBER  BEGIN  DBMS\_JOB.SUBMIT(:jobno,  'DELETE AUTO; INSERT INTO AUTO SELECT \* FROM AUTO1;',  TO\_DATE('01.01.2015 01:05','DD.MM.YYYY HH24:MI'), 'TRUNC(SYSDATE+1)+(3+(5/60))/24');  COMMIT;  END;  Таблица будет очищаться каждый день в три часа пять минут и заполняться данными из AUTO1 Пример3 создания задания  DECLARE jobno NUMBER  BEGIN  DBMS\_JOB.SUBMIT(:jobno,  'DROP AUTO1; CREATE TABLE AUTO1 AS SELECT \* FROM AUTO;',  TO\_DATE('01.01.2015 01:05','DD.MM.YYYY HH24:MI'), 'TRUNC(SYSDATE+1)+(3+(5/60))/24');  COMMIT;  END;  Таблица будет очищаться каждый день в три часа пять минут и заполняться данными из AUTO1 Удаления задания можно сделать следующей процедурой:  DBMS\_JOB.REMOVE ( JOB IN BINARY\_INTEGER );  Здесь JOB – идентификатор задачи Выключение задания Бывают случаи, когда задание временно не должно выполняться. Для этого совсем необязательно его удалять. Достаточно его просто выключить. Выключение (включение) задания производится установкой специального флага состояния - BROKEN. Делается это с помощью следующей процедуры:  DBMS\_JOB.BROKEN (  JOB IN BINARY\_INTEGER,  BROKEN IN BOOLEAN,  NEXT\_DATE IN DATE DEFAULT SYSDATE);  Просмотр всех заданий Для просмотра всех заданий используются следующие таблицы DBA\_JOBS, ALL\_JOBS и USER\_JOBS. • DBA\_JOBS – показывает все задания (JOB) • ALL\_JOBS -показвает задания (JOB) текущего пользователя • USER\_JOBS -показывает задания (JOB) текущего пользователя  **Важные замечания** Вычисление NEXT\_DATE с помощью формулы интервала происходит после выполнения задания. Поэтому, всегда учитывайте это время и старайтесь не ставить их на время близкое к окончанию суток из-за возможного неправильного расчёта следующей даты выполнения.  При создании задания или изменения его параметров ORACLE записывает текущие параметры NLS владельца. Эти параметры каждый раз восстанавливаются при выполнении задания. Это может приводить к некоторым ошибкам в случае ожидания других значений. Поэтому если необходимо лучше производить установку нужных NLS значений с помощью команды ALTER SESSION в параметре WHAT задания.  Задания в теле завершаются COMMIT;  **Вопросы**  Есть ли еще способы создать задние используя планировщик заданий. Да есть более современный метод DBMS\_SCHEDULLER Как создать задание которое бы выполнялось каждые полдня Пример задания  DECLARE jobno NUMBER  BEGIN  DBMS\_JOB.SUBMIT(:jobno,  'DELETE AUTO; INSERT INTO AUTO SELECT \* FROM AUTO1;',  TO\_DATE('01.01.2015 01:05','DD.MM.YYYY HH24:MI'), 'TRUNC(SYSDATE+1/2)’);  COMMIT;  END;  Синтаксис функции **trunc**:  trunc ( date, [ format ] )  *date* — усекаемя дата.  *format* — единица измерения применяемая для усечения. Если параметр *format* пропущен, функция **trunc** усечет дату до дня, то есть часы, минуты, и секунды будут отсечены.  Ниже приведены параметры, используемые в формате усечения:   |  |  | | --- | --- | | **Единица** | **Действующий параметры формата** | | Год | SYYYY, YYYY, YEAR SYEAR, YYY, YY, Y | | ISO Год | IYYY, IY, I | | Четверть | Q | | Месяц | MONTH, MON, MM, RM | | Нделя | WW | | IW | IW | | W | W | | День | DDD, DD, J | | Первый день недели | DAY, DY, D | | Час | HH, HH12, HH24 | | Минута | MI |   Применяется:   * Oracle 8i, Oracle 9i, Oracle 10g, Oracle 11g   Например:   |  |  | | --- | --- | | trunc(to\_date(’22-AUG-03′), ‘YEAR’) | вернет ‘01-JAN-03′ | | trunc(to\_date(’22-AUG-03′), ‘Q’) | вернет ‘01-JUL-03′ | | trunc(to\_date(’22-AUG-03′), ‘MONTH’) | вернет ‘01-AUG-03′ | | trunc(to\_date(’22-AUG-03′), ‘DDD’) | вернет ‘22-AUG-03′ | | trunc(to\_date(’22-AUG-03′), ‘DAY’) | вернет ‘17-AUG-3′ |   **Вопрос №14**  Какой пакет позволяет выполнять программы PL/SQL по расписанию?  DBMS\_RUN  DBMS\_AUTO  DBMS\_LOB  DBMS\_OUTPUT  DBMS\_JOB+ (**См. вопрос 13)**  **Вопрос №15**  Дана таблица **EMP**, содержащая поля:   |  |  | | --- | --- | | **Имя** | **Примечание** | | First\_name | Имя | | Last\_name | Фамилия |   Необходимо просмотреть информацию обо всех служащих с фамилией Smith, но Вы не знаете точно, в каком регистре она сохранена. Какое выражение следует использовать в этом случае?  https://testserver.pro/img/big/i-4KB.gif+  https://testserver.pro/img/big/i-4KC.gif  https://testserver.pro/img/big/i-4K8.gif  https://testserver.pro/img/big/i-4K9.gif  https://testserver.pro/img/big/i-4KA.gif  **Вопрос №16**  Каким образом можно разрешить использование ранее запрещенного триггера auth\_emp, выполняющегося для таблицы **emp**?  https://testserver.pro/img/big/i-4IA.gif  https://testserver.pro/img/big/i-4I9.gif  https://testserver.pro/img/big/i-4I7.gif  https://testserver.pro/img/big/i-4IB.gif  https://testserver.pro/img/big/i-4I8.gif+  После создания триггер по умолчанию включен. Если по какой-то причине его необходимо временно отключить, воспользуйтесь следующим оператором:  SQL> ALTER TRIGGER test DISABLE;  Повторно включить триггер можно с помощью такой команды:  SQL> ALTER TRIGGER test ENABLE;  **Вопрос №17**  Вы создали процедуру:  https://testserver.pro/img/big/i-4KI.gif  Какое из приведённых ниже утверждений является верным?  Для того, чтобы процедура была создана без ошибок, необходимо использовать пакет DBMS\_SQL для выполнения DDL оператора +  Процедура будет создана без ошибок  Для того, чтобы процедура была создана без ошибок, необходимо использовать пакет DBMS\_SQL для выполнения DML оператора  Для того, чтобы процедура была создана без ошибок, необходимо использовать пакет DBMS\_DDL для выполнения DDL оператора  Для того, чтобы процедура была создана без ошибок, необходимо использовать пакет MS\_DDL для выполнения DML оператора  После написания пакета вы можете разрабатывать приложения, которые ссылаются на его типы, вызывать его подпрограммы, использовать его курсор и вызывать его исключение. Когда вы создаете пакет, он сохраняется в базе данных Oracle для общего пользования.  при установке сервера **Oracle** - происходит установка очень большого количества пакетов. Для работы с динамическими запросами **DBMS\_SQL**, для работы с **Web** пакет **DBMS\_WEB** и т.д. Все это огромное количество готового кода, который разработчики фирмы **Oracle** предоставляют в ваше распоряжение  **Вопрос №17**  Какое из приведённых ниже утверждений является верным?  Формальные параметры, передающиеся как IN OUT, не могут быть заданы выражением по умолчанию+  Функция должна содержать один OUT параметр  Параметры передающиеся как IN OUT могут быть как переменными так и значениями (литералами)  Существует 2 способа передачи параметров в подпрограммы: IN OUT и OUT  Процедура не может иметь OUT параметров  Передача значений через параметры OUT и IN OUT  Параметры с модификаторами OUT и IN OUT обычно передаются в вызываемую процедуру и обратно по значению, то есть,   * значения фактических параметров, с которыми вызывается процедура, копируются при вызове в параметры процедуры, и * значения параметров процедуры, установленные в ходе ее работы, копируются обратно в фактические параметры при успешном завершении работы процедуры.   А если в ходе выполнения процедуры возникает исключение, то фактические параметры остаются без изменений:  SQL> DECLARE  l\_out VARCHAR2(50) := 'не важно';  l\_inout VARCHAR2(50) := 'не важно';    PROCEDURE x(p\_out OUT VARCHAR2, p\_inout IN OUT VARCHAR2) IS  BEGIN  dbms\_output.put\_line('1 p\_out: ' || p\_out);  dbms\_output.put\_line('1 p\_inout: ' || p\_inout);  p\_out := 'результат';  p\_inout := 'результат';  dbms\_output.put\_line('2 l\_out: ' || l\_out);  dbms\_output.put\_line('2 l\_inout: ' || l\_inout);  RAISE no\_data\_found;  END x;  BEGIN  x(l\_out, l\_inout);  EXCEPTION  WHEN no\_data\_found THEN  dbms\_output.put\_line('3 l\_out: ' || l\_out);  dbms\_output.put\_line('3 l\_inout: ' || l\_inout);  END;  /  1 p\_out:  1 p\_inout: не важно  2 l\_out: не важно  2 l\_inout: не важно  3 l\_out: не важно  3 l\_inout: не важно  PL/SQL procedure successfully completed  Из примера видно следующее:   1. Параметр p\_out OUT в процедуре x инициализирован значением NULL перед выполнением кода процедуры - так всегда происходит с OUT параметрами. Значение фактического параметра l\_out в вызванной процедуре недоступно через "внутренюю переменную" p\_out. Тогда как значение переменной l\_inout доступно в процедуре через параметр p\_inout IN OUT. 2. При аварийном завершении процедуры (в результате исключения) значения формальных параметров p\_out и p\_inout не копируются в формальные параметры l\_out и l\_inout.   Итак, PL/SQL заботливо предохраняет внешние по отношению к процедуре переменные от изменений вплоть до момента успешного завершения процедуры.  Платой за эту предосторожность является удваивание объема памяти, необходимого для хранения значений OUT и IN OUT параметров. Если эти параметры имеют составной тип данных (коллекция, запись, объект) и занимают много места в памяти, то их копирование при входе и выходе из процедуры неизбежно снижает производительность.  Подсказка (hint) NOCOPY рекомендует виртуальной машине PL/SQL передавать параметры в процедуру по ссылке, а не по значению, что не требует копирования. При этом код процедуры напрямую изменяет значения фактических параметров:  **Вопрос №18**  Какая из приведенных ниже команд отменяет определение переменной подстановки?  CANCEL  ACCEPT  DEFINE  UNDEFINE+  REMOVE  При написании и использовании сценариев SQL иногда бывает необходимо указывать переменные и их значения. Команда *DEFINE* позволяет создавать свои собственные переменные (пользовательские переменные), продолжающие хранить заданные для них значения либо до конца всего данного сеанса SQL\*Plus, либо до тех пор, пока не будет применена команда *UNDEFINE*, которая сбрасывает значения переменных. Ниже приведен пример, показывающий, как применять команды *DEFINE* и *UNDEFINE*:  SQL> DEFINE dept = finance  SQL> UNDEFINE dept  В этом примере все выглядит довольно просто. На практике команда *DEFINE* довольно часто применяется для подстановки значений переменных. В таких случаях она обычно используется не с пользовательскими переменными, а с переменными подстановки (substitution variables). Такие переменные определяются за счет добавления к пользовательской переменной символа амперсанда (*&*), например: *&VARIABLE*.  В листинге ниже показан простой пример команды *DEFINE* с переменной подстановки.  SQL> col segment\_name for a27  DEFINE owner = '&1'  SELECT segment\_name,segment\_type,extents  FROM dba\_segments  WHERE owner = upper ('&owner')  AND extents > 10  AND segment\_name NOT LIKE 'TMP%'  ORDER BY segment\_type,extents desc  SQL> @extents.sql  Enter value for 1: system  SEGMENT\_NAME SEGMENT\_TYPE EXTENTS  HELP\_TOPIC\_SEQ INDEX 18  PRODUCT\_PROFILE TABLE 22  SQL>  В приведенном выше листинге  сценарии *extents.sql* была определена переменная *owner*, но вместо одного жестко закодированного значения было указано, что эта переменная должна принимать любое подставляемое значение, которое предоставит пользователь. Благодаря этому, сценарий может выполняться для любого пользователя в базе данных. Все, что потребуется — указывать при каждом его запуске другое имя для владельца схемы. **Вопрос №19** Дана таблица **ORDER**, содержащая поля:   |  |  |  | | --- | --- | --- | | **Имя** | **Тип** | **Примечаение** | | Ordid | Numeric | Идентификатор записи | | Shipdate | Date | Дата отгрузки | | Total | Numeric | Количество товара |   Необходимо создать отчет, отображающий дату отгрузки и количество товара. Если заказ не был отгружен или данного количество товара нет в наличии, то это должно быть отражено в отчете. Какой фрагмент кода позволит решить поставленную задачу?  https://testserver.pro/img/big/i-4GO.gif  https://testserver.pro/img/big/i-4GP.gif  https://testserver.pro/img/big/i-4GR.gif  https://testserver.pro/img/big/i-4GS.gif  https://testserver.pro/img/big/i-4GQ.gif+  **Функция TO\_CHAR**  [Функция TO\_CHAR](https://oracle-patches.com/db/sql/3388-%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8-%D0%B8-%D0%B0%D1%80%D0%B3%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B-%D1%82%D0%B8%D0%BF%D0%B0-char-%D0%B2-pl-sql) выполняет задачу, обратную функции TO\_NUMBER: она преобразует число в его символьное представление. Используя необязательную маску форматирования, можно подробно указать, каким должно быть представление. Функция TO\_CHAR вызывается следующим образом:  TO\_CHAR(число [,формат [,параметры\_nls]])  Здесь число — это число, которое требуется представить в символьной форме. Оно может относиться к любому из числовых типов PL/SQL: NUMBER, PLS\_INTEGER, BINARY\_INTEGER, BINARY\_FLOAT, BINARY\_DOUBLE, SIMPLE\_INTEGER, SIMPLE\_FLOAT или SIMPLE\_DOUBLE. Параметр формат содержит необязательную маску форматирования, определяющую способ представления числа в символьной форме; необязательная строка параметры\_nls содержит значения параметров NLS. Ее можно применить для замещения текущих установок параметров NLS уровня сеанса.  Если вы хотите, чтобы результат был представлен в национальном наборе символов, используйте вместо TO\_CHAR функцию TO\_NCHAR. При этом помните, что строка форматирования числа должна быть представлена символами национального набора; в противном случае полученная строка будет состоять из символов «#».    Использование TO\_CHAR без маски форматирования  Функция TO\_CHAR, как и TO\_NUMBER, может вызываться без маски форматирования:  DECLARE  b VARCHAR2(30);  BEGIN  b := TO\_CHAR(123456789.01);  DBMS\_OUTPUT.PUT\_LINE(b);  END;  Результат выглядит так:  123456789.01  В отличие от TO\_NUMBER форма TO\_CHAR особой пользы не приносит. Чтобы число лучше читалось, нужно задать как минимум разделитель групп разрядов.    Использование функции TO\_CHAR с маской форматирования  При преобразовании числа в символьное представление функция TO\_CHAR используется чаще всего с маской форматирования. Например, с ее помощью можно вывести денежную сумму:  DECLARE  b VARCHAR2(30);  BEGIN  b := TO\_CHAR(123456789.01,'L999G999G999D99');  DBMS\_OUTPUT.PUT\_LINE(b);  END;  В локальном контексте США результат будет выглядеть так:  $123,456,789.01  **Вопрос №20**  Какое из нижеперечисленных представлений НЕ содержит информации о зависимостях?  IDEPTREE  USER\_OBJECTS+  USER\_DEPENDENCIES  DEPTREE\_TEMPTAB  DEPTREE  PL/SQL-программы является проверка ее зависимостей. Зависимость в PL/SQL представляет собой вид связи между программой и некоторым объектом Oracle, существующим вне этой программы. Серверные программы PL/SQL могут зависеть от таблиц, представлений, типов данных, процедур, функций, последовательностей и спецификаций пакетов, но не от тела пакетов или типов данных (последние относятся к «скрытой» реализации).  **Оглавление статьи**[[Показать](https://oracle-patches.com/db/sql/3666-%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B7%D0%B0%D0%B2%D0%B8%D1%81%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D1%8F%D0%BC%D0%B8-%D0%B8-%D0%BF%D0%B5%D1%80%D0%B5%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%86%D0%B8%D1%8F-%D0%B2-pl-sql)]  **Основная цель проверки зависимостей в PL/SQL** — *не допустить выполнения программы*, если *хоть один* из объектов, от которых она зависит, *изменился* с момента ее последней компиляции.  К счастью, управление зависимостями производится автоматически, от отслеживания зависимостей до их перекомпиляции в случае необходимости. Тем не менее некоторая ответственность за синхронизацию кода лежит на программистах, и в следующих разделах рассказывается, как, когда и для чего им следует воздействовать на этот процесс.  В Oracle10g и более ранних версиях зависимости отслеживались на уровне программных модулей. Если процедура зависела от функции пакета или столбца таблицы, то зависимой единицей становился пакет или таблица. Такой уровень детализации считался стандартным с первых дней PL/SQL и до недавнего времени.  В Oracle11g детализация отслеживания зависимостей была улучшена. Зависимости теперь отслеживаются не до уровня пакетов или таблиц, а до отдельных элементов (например, столбцов таблицы или программ пакета вместе с формальными параметрами вызова и режимами передачи). Точное отслеживание зависимостей означает, что программа останется действительной в случае добавления или перегрузки существующей программы в существующем пакете. Аналогичным образом при добавлении нового столбца в таблицу база данных не объявит недействительными все программы PL/ SQL, ссылающиеся на эту таблицу, — только те программы, которые ссылаются на все столбцы (например, с использованием конструкции SELECT \* или объявления %ROWTYPE). В следующих разделах эта ситуация рассматривается более подробно.  В подразделе «[Уточнение ссылок на переменные и столбцы в командах SQL](https://oracle-patches.com/db/sql/3174-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0-pl-sql-%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0-%D0%B1%D0%BB%D0%BE%D0%BA%D0%B0-pl-sql-%D0%B8-%D0%BE%D0%B1%D0%BB%D0%B0%D1%81%D1%82%D1%8C-%D0%B2%D0%B8%D0%B4%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D0%B8#h5-1-utochnenie-ssylok-na-peremennye-i-stolbtsy-v-komandakh-sql)» блога приведен пример точного управления зависимостями.  К сожалению, в Oracle11g Release 2 эти данные все еще недоступны в представлениях словарей данных. Хочется верить, что в будущем информация станет доступной.  А пока использование любых версий, предшествующих Oracle11g, означает, что при любых изменениях в базах данных многие объекты будут автоматически становиться недействительными.  **Анализ зависимостей с использованием представлений словаря данных**  Для анализа зависимостей можно использовать некоторые представления словаря данных. Рассмотрим простой пример. Допустим, на сервере имеется пакет bookworm, а в нем имеется функция, извлекающая данные из таблицы books. Непосредственно после создания и таблица, и пакет действительны (valid):  SELECT object\_name, object\_type, status  FROM USER\_OBJECTS  WHERE object\_name = 'BOOKWORM';  OBJECT\_NAME OBJECT\_TYPE STATUS  ------------------------------ ------------------ -------  BOOKWORM PACKAGE VALID  BOOKWORM PACKAGE BODY VALID  При компиляции программы PL/SQL база данных формирует список объектов, необходимых для успешной компиляции пакета BOOKWORM. Для определения всех зависимостей между объектами можно построить граф зависимостей при помощи запроса к представлению USER\_DEPENDENCIES:  SELECT name, type, referenced\_name, referenced\_type  FROM USER\_DEPENDENCIES  WHERE name = 'BOOKWORM';  NAME TYPE REFERENCED\_NAME REFERENCED\_TYPE  --------------- -------------- --------------- ---------------  BOOKWORM PACKAGE STANDARD PACKAGE  BOOKWORM PACKAGE BODY STANDARD PACKAGE  BOOKWORM PACKAGE BODY BOOKS TABLE  BOOKWORM PACKAGE BODY BOOKWORM PACKAGE  **Вопрос №22**  На базе таблицы **EMPLOYEE** создано представление **ANN\_SAL**:   |  |  |  | | --- | --- | --- | | **NAME** | **NULL** | **TYPE** | | EMPNO | NOT NULL | NUMBER(4) | | YEARLY\_SAL |  | NUMBER(9,2) | | MONTHLY\_SAL |  | NUMBER(9,2) |     С помощью какого выражения можно извлечь данные из представления **ANN\_SAL**?  SELECT \* FROM EMPLOYEE  SELECT \* FROM VIEW ANN\_SAL  SELECT \* FROM VIEW ANN\_SAL IS DON EMPLOYEE  SELECT \* FROM VIEW ANN\_SAL FROM EMPLOYEE  SELECT \* FROM ANN\_SAL+  **Вопрос №26**  В течение какого времени остаются определенными переменные подстановки, заданные командой DEFINE?  До окончания сессии+  До остановки базы данных  До завершения выполнения запроса  До окончания транзакции  До выключения компьютера  **Вопрос №27**  Какое из приведенных ниже утверждений верно относительно скрипта, представленного ниже:  https://testserver.pro/img/big/i-4FS.gif  Отчет будет иметь заголовок, выровненный по левому краю  Между заголовком и данными отчета будет промежуток в три строки  Каждая страница отчета будет иметь заголовок, выровненный по центру  Отчет будет иметь заголовок, выровненный по центру+  Каждая страница отчета будет иметь концевик, выровненный по центру  **Вопрос №28**  Принято решение применить явный курсор в своей программе. Какой последовательности действий следует придерживаться при работе с явными курсорами?  Объявить курсор, загрузить данные, закрыть курсор  Открыть курсор, объявить курсор, загрузить данные, закрыть курсор  Объявить курсор, инициализировать курсор, открыть курсор, загрузить данные, закрыть курсор+  Объявить курсор, открыть курсор, загрузить данные, закрыть курсор  Объявить курсор, открыть курсор, закрыть курсор, загрузить данные  Одной из важнейших характеристик PL/SQL является тесная интеграция с базой данных Oracle в отношении как изменения данных в таблицах, так и выборки данных из таблиц. В этом блоге рассматриваются элементы PL/SQL, связанные с выборкой информации из базы данных и ее обработкой в программах PL/SQL.  При выполнении команды SQL из PL/SQL РСУБД Oracle назначает ей приватную рабочую область, а некоторые данные записывает в системную глобальную область (SGA, System Global Area). В приватной рабочей области содержится информация о [команде SQL](https://oracle-patches.com/oracle/develop/2924-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5-%D1%80%D0%B0%D0%B1%D0%BE%D1%87%D0%B8%D0%B5-%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D1%8B-sql-plus) и набор данных, возвращаемых или обрабатываемых этой командой. [PL/SQL](https://oracle-patches.com/db/sql/3125-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0-pl-sql) предоставляет программистам несколько механизмов доступа к этой рабочей области и содержащейся в ней информации; все они так или иначе связаны с опреде­лением курсоров и выполнением операций с ними.  **Оглавление статьи**[[Показать](https://oracle-patches.com/db/sql/3560-%D0%BA%D1%83%D1%80%D1%81%D0%BE%D1%80%D1%8B-%D0%B2-pl-sql-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5-%D0%BF%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF%D1%8B-%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)]   |  | | --- | |  |  * **Неявные курсоры**. Команда SELECT.. .INTO считывает одну строку данных и при­сваивает ее в качестве значения локальной переменной программы. Это простейший (и зачастую наиболее эффективный) способ доступа к данным, но он часто ведет к написанию сходных и даже одинаковых SQL-команд SELECT во многих местах программы. * **Явные курсоры**. Запрос можно явно объявить как курсор в разделе объявлений локального блока или пакета. После этого такой курсор можно будет открывать и выбирать из него данные в одной или нескольких программах, причем возмож­ности управления явным курсором шире, чем у неявного. * **Курсорные переменные**. Курсорные переменные (в объявлении которых задается тип REF CURSOR) позволяют передавать из программы в программу указатель на результирующий набор строк запроса. Любая программа, для которой доступна такая переменная, может открыть курсор, извлечь из него необходимые данные и закрыть его. * **Курсорные выражения**. Ключевое слово CURSOR превращает команду SELECT в набор [REF CURSOR](https://oracle-patches.com/db/sql/3574-%D0%BA%D1%83%D1%80%D1%81%D0%BE%D1%80%D0%BD%D1%8B%D0%B5-%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5-%D0%B8-ref-cursor-%D0%B2-pl-sql), который может использоваться совместно с табличными функциями для повышения производительности приложения. * **Динамические SQL-запросы**. Oracle позволяет динамически конструировать и вы­полнять запросы с использованием либо встроенного динамического SQL либо программ пакета DMBS\_SQL. Этот встроенный пакет описывается в документации Oracle, а также в [книге Oracle Built-in Packages](https://oracle-patches.com/%D0%BA%D0%BD%D0%B8%D0%B3%D0%B8/38-%D0%B1%D0%B0%D0%B7%D1%8B-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85-%D0%BE%D1%80%D0%B0%D0%BA%D0%BB/842-oracle-built-in-packages) (O’Reilly).     **Основные принципы работы с курсорами**  Курсор проще всего представить себе как указатель на таблицу в базе данных. Напри­мер, следующее объявление связывает всю таблицу employee с курсором employee\_cur:  CURSOR employee\_cur IS SELECT \* FROM employee;  Объявленный курсор можно открыть:  OPEN employee\_cur;  Далее из него можно выбирать строки:  FETCH employee\_cur INTO employee\_rec;  Завершив работу с курсором, его следует закрыть:  CLOSE employee\_cur;  В этом случае каждая выбранная из курсора запись представляет строку таблицы employee. Однако с курсором можно связать любую допустимую команду SELECT. В сле­дующем примере в объявлении курсора объединяются три таблицы:  DECLARE  CURSOR joke\_feedback\_cur IS  SELECT J.name, R.laugh\_volume, C.name FROM Joke J, response R, comedian C WHERE J.joke\_id = R.joke\_id AND R.joker\_id = C.joker\_id;  BEGIN  END;  В данном случае курсор действует не как указатель на конкретную таблицу базы дан­ных — он указывает на виртуальную таблицу или неявное представление, определяемое командой SELECT. (Такая таблица называется виртуальной, потому что команда SELECT генерирует данные с табличной структурой, но эта таблица существует только вре­менно, пока программа работает с возвращенными командой данными.) Если тройное объединение возвращает таблицу из 20 строк и 3 столбцов, то курсор действует как указатель на эти 20 строк  **Вопрос №30**  База данных содержит таблицу **EMPLOYEES**:   |  |  |  | | --- | --- | --- | | **Имя** | **NULL** | **Тип** | | FIRST\_NAME |  | VARCHAR2(20) | | LAST\_NAME | NOT NULL | VARCHAR2(25) | | EMAIL | NOT NULL | VARCHAR2(25) | | PHONE\_NUMBER |  | VARCHAR2(20) | | HIRE\_DATE | NOT NULL | DATE | | JOB\_ID | NOT NULL | VARCHAR2(10) | | SALARY | NOT NULL | NUMBER(8,2) | | COMMISSION\_PCT |  | NUMBER(2,2) | | MANAGER\_ID |  | NUMBER(6) | | DEPARTMENT\_ID |  | NUMBER(4) |   Необходимо создать индекс с названием name\_ind на столбцах **first\_name** и **last\_name**. Какое выражение позволит решить поставленную задачу?  https://testserver.pro/img/big/i-4MG.gif  https://testserver.pro/img/big/i-4ME.gif  https://testserver.pro/img/big/i-4MF.gif+  https://testserver.pro/img/big/i-4MD.gif  https://testserver.pro/img/big/i-4MC.gif  **Вопрос №31**  Вы выполнили следующие команды:  https://testserver.pro/img/big/i-4MA.gif  Каким будет результат выполнения приведенного ниже оператора?  https://testserver.pro/img/big/i-4MB.gif  1035  1010  1005  1040  1020+  **Вопрос №32**  Вы решили полностью очистить таблицу **STUDENT**, но структура таблицы вам понадобится в будущем. Ограничения ссылочной целостности присутствуют. Какая из перечисленных ниже команд позволит решить поставленную задачу?  **TRUNCATE TABLE** student **KEEP STRUCTURE**;  **DROP TABLE** student;  **DELETE** \* **FROM** student;  **DELETE** \* **FROM** student **KEEP STRUCTURE**;  **TRUNCATE TABLE** student;+  **Вопрос №33**  Рассмотрите структуру таблиц **EMPLOYEES** и **EMP**: **EMPLOYEES**   |  |  |  | | --- | --- | --- | | **Имя** | **NULL** | **Тип** | | EMPLOYEE\_ID | NOT NULL | NUMBER(6) | | FIRST\_NAME |  | VARCHAR2(20) | | LAST\_NAME | NOT NULL | VARCHAR2(25) | | HIRE\_DATE | NOT NULL | DATE | | JOB\_ID | NOT NULL | VARCHAR2(10) | | SALARY | NOT NULL | NUMBER(8,2) | | COMMISSION\_PCT |  | NUMBER(2,2) | | DEPARTMENT\_ID |  | NUMBER(4) |   **EMP**   |  |  |  | | --- | --- | --- | | **Имя** | **NULL** | **Тип** | | EMP\_ID | NOT NULL | NUMBER | | ENAME |  | VARCHAR2(20) |   Какое из перечисленных ниже выражений выполняется верно?  https://testserver.pro/img/big/i-4JD.gif  https://testserver.pro/img/big/i-4JA.gif+  https://testserver.pro/img/big/i-4JB.gif  https://testserver.pro/img/big/i-4JE.gif  https://testserver.pro/img/big/i-4JC.gif  **Вопрос №34**  С какими привилегиями выполняется процедура DBMS\_DDL.ALTER\_COMPILE?  С привилегией ALTER ANY TABLE  С привилегиями владельца пакета DBMS\_DDL SYS  C привилегиями вызвавшего процедуру пользователя+  С привилегиями DBA  С привилегией ALTER ANY PROCEDURE  **Вопрос №35**  База данных содержит следующие таблицы: **EMPLOYEES**   |  |  |  | | --- | --- | --- | | **Имя** | **NOT NULL** | **Тип** | | EMPLOYEE\_ID | NOT NULL | NUMBER(6) | | LAST\_NAME | NOT NULL | VARCHAR2(25) | | JOB\_ID | NOT NULL | VARCHAR2(10) | | SALARY | NOT NULL | NUMBER(8,2) | | COMMISSION\_PCT |  | NUMBER(2,2) | | DEPARTMENT\_ID |  | NUMBER(4) |     **DEPARTMENTS**   |  |  |  | | --- | --- | --- | | **Имя** | **NOT NULL** | **Тип** | | DEPARTMENT\_NAME | NOT NULL | VARCHAR2(30) | | LOCATION\_ID |  | NUMBER(4) | | DEPARTMENT\_ID | NOT NULL | NUMBER(4) |     **LOCATIONS**   |  |  |  | | --- | --- | --- | | **Имя** | **NULL** | **Тип** | | DEPARTMENT\_ID | NOT NULL | NUMBER(4) | | CITY | NOT NULL | VARCHAR2(30) | | LOCATION\_ID |  | NUMBER(4) |     Необходимо определить количество служащих, работающих в Оксфорде и получающих комиссию. Какой запрос позволит решить поставленную задачу?  Внимание! Ответ будет дан сразу при выборе одного из предложенных вариантов  https://testserver.pro/img/big/i-4IV.gif  https://testserver.pro/img/big/i-4IU.gif  https://testserver.pro/img/big/i-4IX.gif  https://testserver.pro/img/big/i-4IT.gif+  **Вопрос №36**  Какой командой можно удалить функцию calculate?  https://testserver.pro/img/big/i-4FL.gif  https://testserver.pro/img/big/i-4FI.gif+  https://testserver.pro/img/big/i-4FH.gif  https://testserver.pro/img/big/i-4FK.gif  https://testserver.pro/img/big/i-4FJ.gif  **Вопрос №37**  Какой из представленных ниже запросов использует внутреннее представление?  https://testserver.pro/img/big/i-4HZ.gif+  https://testserver.pro/img/big/i-4HY.gif  https://testserver.pro/img/big/i-4HW.gif  https://testserver.pro/img/big/i-4HX.gif  https://testserver.pro/img/big/i-4H_.gif  **Вопрос №38**  Какое из утверждений является верным?  Результат применения логического оператора NOT к NULL всегда TRUE  Применение логического оператора NOT к переменной, значение которой NULL, приводит к ошибке  Результат применения логического оператора NOT к переменной, значение которой NULL, зависит от типа этой переменной  Результат применения логического оператора NOT к NULL всегда NULL+  Результат применения логического оператора NOT к NULL всегда FALSE  **Вопрос №39**  База данных содержит следующие таблицы и поля: employees(employee\_id, first\_name, last\_name, salary, department\_id), departments(department\_id, department\_name, location\_id), locations(location\_id, street\_address, city). Необходимо выяснить название и адрес отделения, в котором работает сотрудник с номером 117. Какой запрос позволит решить поставленную задачу?  https://testserver.pro/img/big/i-4JM.gif  https://testserver.pro/img/big/i52P1.gif+  https://testserver.pro/img/big/i-4JL.gif  https://testserver.pro/img/big/i-4JK.gif **Вопрос №40** Какое из утверждений о пакетах является верным?  Можно удалить функцию из пакета используя команду ALTER PACKAGE  Функции объявленные в спецификации пакета не могут быть использованы в SQL операторах  Пакеты могут иметь параметры  Команда DROP PACKAGE удаляет спецификацию и тело пакета+  Пакеты могут быть вложенными  Завершить протокол лаб. работы (команда spool off) и показать результаты преподавателю.  Пример:  SQL> spool off |