Бесплатная электронная книга

УЧУСЬ

Oracle Database

Free unaffiliated eBook created from
Stack Overflow contributors.

#oracle

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: oracle-database

It is an unofficial and free Oracle Database ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Oracle Database.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с Oracle Database

замечания

Oracle - это система управления реляционными базами данных (RDBMS), первоначально созданная Ларри Эллисоном, Бобом Майнером и Эд Оутсом в конце 70-х. Он должен был быть совместим с IBM System R.

Версии

Версия	Дата выхода
Версия 1 (неизданная)	1978-01-01
Oracle V2	1979-01-01
Версия Oracle 3	1983-01-01
Версия Oracle 4	1984-01-01
Версия Oracle 5	1985-01-01
Версия Oracle 6	1988-01-01
Oracle7	1992-01-01
Oracle8	1997-07-01
Oracle8i	1999-02-01
Oracle9i	2001-06-01
Oracle 10g	2003-01-01
Oracle 11g	2007-01-01
Oracle 12c	2013-01-01

Examples

Привет, мир

```
SELECT 'Hello world!' FROM dual;
```

В духе Oracle SQL, «dual - это просто таблица состязаний» . Первоначально он был предназначен для двойных рядов через JOIN, но теперь содержит одну строку с значением `DUMMY` 'X'.

Привет, мир! из таблицы

Создайте простую таблицу

```
create table MY_table (
   what varchar2(10),
   who varchar2(10),
   mark varchar2(10)
);
```

Вставить значения (вы можете опустить целевые столбцы, если вы предоставляете значения для всех столбцов)

```
insert into my_table (what, who, mark) values ('Hello', 'world', '!' );
insert into my_table values ('Bye bye', 'ponies', '?' );
insert into my_table (what) values('Hey');
```

Не забудьте зафиксировать, потому что Oracle использует *транзакции*

```
commit;
```

Выберите свои данные:

```
select what, who, mark from my_table where what='Hello';
```

SQL-запрос

Список сотрудников, зарабатывающих более $ 50000, родился в этом веке. Укажите их имя, дату рождения и зарплату, отсортированную по алфавиту по имени.

```
SELECT employee_name, date_of_birth, salary
FROM   employees
WHERE  salary > 50000
   AND date_of_birth >= DATE '2000-01-01'
ORDER BY employee_name;
```

Покажите количество сотрудников в каждом отделе не менее 5 сотрудников. Сначала перечислите крупнейшие отделы.

```
SELECT department_id, COUNT(*)
FROM   employees
GROUP BY department_id
HAVING COUNT(*) >= 5
ORDER BY COUNT(*) DESC;
```

Hello World из PL / SQL

```
/* PL/SQL is a core Oracle Database technology, allowing you to build clean, secure,
   optimized APIs to SQL and business logic. */

set serveroutput on

BEGIN
   DBMS_OUTPUT.PUT_LINE ('Hello World!');
END;
```

Прочитайте Начало работы с Oracle Database онлайн:
https://riptutorial.com/ru/oracle/topic/558/начало-работы-с-oracle-database

глава 2: JOINS

Examples

ПЕРЕКРЕСТНЫЙ ПРИСОЕДИНЕНИЕ

`CROSS JOIN` выполняет соединение между двумя таблицами, которое не использует явное предложение о соединении и приводит к декартовому произведению двух таблиц. Декартово произведение означает, что каждая строка одной таблицы объединяется с каждой строкой второй таблицы в соединении. Например, если `TABLEA` имеет 20 строк и `TABLEB` имеет 20 строк, результатом будет `20*20 = 400` строк вывода.

Пример:

```
SELECT *
FROM TABLEA CROSS JOIN TABLEB;
```

Это также можно записать в виде:

```
SELECT *
FROM TABLEA, TABLEB;
```

Вот пример перекрестного соединения в SQL между двумя таблицами:

Пример таблицы: TABLEA

```
+-------+---------+
| VALUE |   NAME  |
+-------+---------+
|   1   |   ONE   |
|   2   |   TWO   |
+-------+---------+
```

Пример таблицы: TABLEB

```
+-------+--------+
| VALUE |  NAME  |
+-------+--------+
|   3   |  THREE |
|   4   |  FOUR  |
+-------+--------+
```

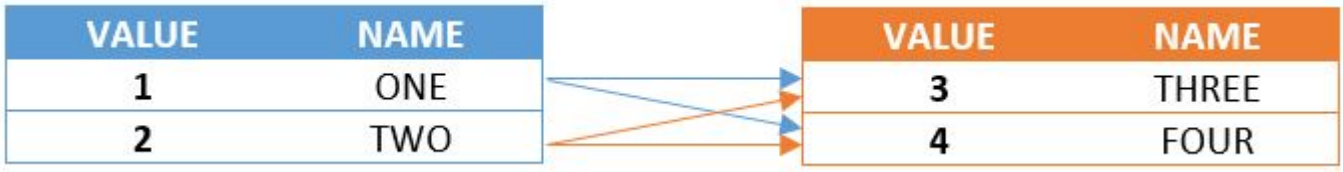
Теперь, если вы выполните запрос:

```
SELECT *
FROM TABLEA CROSS JOIN TABLEB;
```

Выход:

```
+-------+--------+-------+--------+
| VALUE |  NAME  | VALUE |  NAME  |
+-------+--------+-------+--------+
|   1   |   ONE  |   3   |  THREE |
|   1   |   ONE  |   4   |  FOUR  |
|   2   |   TWO  |   3   |  THREE |
|   2   |   TWO  |   4   |  FOUR  |
+-------+--------+-------+--------+
```

Вот как происходит перекрестное соединение между двумя таблицами:



Подробнее о Cross Join: документация Oracle

ВНУТРЕННЕЕ СОЕДИНЕНИЕ

INNER JOIN - операция JOIN, которая позволяет вам указать явное предложение о соединении.

Синтаксис

TableExpression [INNER] JOIN TableExpression {ON booleanExpression | ИСПОЛЬЗОВАНИЕ}

Вы можете указать предложение соединения, указав ON с булевым выражением.

Объем выражений в предложении ON включает текущие таблицы и любые таблицы во внешних блоках запросов к текущему SELECT. В следующем примере предложение ON относится к текущим таблицам:

```
-- Join the EMP_ACT and EMPLOYEE tables
-- select all the columns from the EMP_ACT table and
-- add the employee's surname (LASTNAME) from the EMPLOYEE table
-- to each row of the result
SELECT SAMP.EMP_ACT.*, LASTNAME
 FROM SAMP.EMP_ACT JOIN SAMP.EMPLOYEE
 ON EMP_ACT.EMPNO = EMPLOYEE.EMPNO
-- Join the EMPLOYEE and DEPARTMENT tables,
-- select the employee number (EMPNO),
-- employee surname (LASTNAME),
-- department number (WORKDEPT in the EMPLOYEE table and DEPTNO in the
-- DEPARTMENT table)
-- and department name (DEPTNAME)
-- of all employees who were born (BIRTHDATE) earlier than 1930.
SELECT EMPNO, LASTNAME, WORKDEPT, DEPTNAME
 FROM SAMP.EMPLOYEE JOIN SAMP.DEPARTMENT
 ON WORKDEPT = DEPTNO
```

```
 AND YEAR(BIRTHDATE) < 1930

-- Another example of "generating" new data values,
-- using a query which selects from a VALUES clause (which is an
-- alternate form of a fullselect).
-- This query shows how a table can be derived called "X"
-- having 2 columns "R1" and "R2" and 1 row of data
SELECT *
FROM (VALUES (3, 4), (1, 5), (2, 6))
AS VALUESTABLE1(C1, C2)
JOIN (VALUES (3, 2), (1, 2),
(0, 3)) AS VALUESTABLE2(c1, c2)
ON VALUESTABLE1.c1 = VALUESTABLE2.c1
-- This results in:
-- C1          |C2          |C1          |2
-- -----------------------------------------
-- 3           |4           |3           |2
-- 1           |5           |1           |2


-- List every department with the employee number and
-- last name of the manager

SELECT DEPTNO, DEPTNAME, EMPNO, LASTNAME
FROM DEPARTMENT INNER JOIN EMPLOYEE
ON MGRNO = EMPNO

-- List every employee number and last name
-- with the employee number and last name of their manager
SELECT E.EMPNO, E.LASTNAME, M.EMPNO, M.LASTNAME
FROM EMPLOYEE E INNER JOIN
DEPARTMENT INNER JOIN EMPLOYEE M
    ON MGRNO = M.EMPNO
    ON E.WORKDEPT = DEPTNO
```

ЛЕВЫЙ ВНЕШНИЙ ВСТУПИТЕЛЬ

`LEFT OUTER JOIN` выполняет соединение между двумя таблицами, для которых требуется явное предложение соединения, но не исключает непревзойденные строки из первой таблицы.

Пример:

```
SELECT
      ENAME,
      DNAME,
      EMP.DEPTNO,
      DEPT.DEPTNO
FROM
      SCOTT.EMP LEFT OUTER JOIN SCOTT.DEPT
      ON EMP.DEPTNO = DEPT.DEPTNO;
```

Несмотря на то, что синтаксис ANSI является рекомендуемым , он может часто встречаться с устаревшим синтаксисом. Используя `(+)` внутри условия, определите, какую сторону уравнения рассматривать как *внешнюю* .

```
SELECT
      ENAME,
      DNAME,
      EMP.DEPTNO,
      DEPT.DEPTNO
FROM
      SCOTT.EMP,
      SCOTT.DEPT
WHERE
      EMP.DEPTNO = DEPT.DEPTNO(+);
```

Ниже приведен пример левой внешней связи между двумя таблицами:

Пример таблицы: EMPLOYEE

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     A     |    2    |
|     B     |    1    |
|     C     |    3    |
|     D     |    2    |
|     E     |    1    |
|     F     |    1    |
|     G     |    4    |
|     H     |    4    |
+-----------+---------+
```

Пример таблицы: DEPT

```
+---------+--------------+
|  DEPTNO |   DEPTNAME   |
+---------+--------------+
|    1    |  ACCOUNTING  |
|    2    |    FINANCE   |
|    5    |   MARKETING  |
|    6    |      HR      |
+---------+--------------+
```

Теперь, если вы выполните запрос:

```
SELECT
      *
FROM
      EMPLOYEE LEFT OUTER JOIN DEPT
      ON EMPLOYEE.DEPTNO = DEPT.DEPTNO;
```

Выход:

```
+-----------+---------+---------+--------------+
|    NAME   |  DEPTNO |  DEPTNO |   DEPTNAME   |
+-----------+---------+---------+--------------+
|     F     |    1    |    1    |  ACCOUNTING  |
|     E     |    1    |    1    |  ACCOUNTING  |
|     B     |    1    |    1    |  ACCOUNTING  |
```

```
|     D     |    2    |    2    |    FINANCE   |
|     A     |    2    |    2    |    FINANCE   |
|     C     |    3    |         |              |
|     H     |    4    |         |              |
|     G     |    4    |         |              |
+-----------+---------+---------+--------------+
```

ПРАВО НА ВСТУПЛЕНИЕ

`RIGHT OUTER JOIN` выполняет соединение между двумя таблицами, для которых требуется явное предложение соединения, но не исключает непревзойденные строки из второй таблицы.

Пример:

```
SELECT
      ENAME,
      DNAME,
      EMP.DEPTNO,
      DEPT.DEPTNO
FROM
      SCOTT.EMP RIGHT OUTER JOIN SCOTT.DEPT
      ON EMP.DEPTNO = DEPT.DEPTNO;
```

Поскольку включены `SCOTT.DEPT` строки `SCOTT.DEPT` , но несогласованные строки `SCOTT.EMP` не являются, приведенное выше эквивалентно следующему выражению, использующему `LEFT OUTER JOIN` .

```
SELECT
      ENAME,
      DNAME,
      EMP.DEPTNO,
      DEPT.DEPTNO
FROM
      SCOTT.DEPT RIGHT OUTER JOIN SCOTT.EMP
      ON DEPT.DEPTNO = EMP.DEPTNO;
```

Ниже приведен пример правой внешней связи между двумя таблицами:

Пример таблицы: EMPLOYEE

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     A     |    2    |
|     B     |    1    |
|     C     |    3    |
|     D     |    2    |
|     E     |    1    |
|     F     |    1    |
|     G     |    4    |
|     H     |    4    |
+-----------+---------+
```

Пример таблицы: DEPT

```
+---------+--------------+
|  DEPTNO |   DEPTNAME   |
+---------+--------------+
|    1    |  ACCOUNTING  |
|    2    |    FINANCE   |
|    5    |   MARKETING  |
|    6    |      HR      |
+---------+--------------+
```

Теперь, если вы выполните запрос:

```
SELECT
      *
FROM
      EMPLOYEE RIGHT OUTER JOIN DEPT
      ON EMPLOYEE.DEPTNO = DEPT.DEPTNO;
```

Выход:

```
+-----------+---------+---------+--------------+
|    NAME   |  DEPTNO |  DEPTNO |   DEPTNAME   |
+-----------+---------+---------+--------------+
|     A     |    2    |    2    |    FINANCE   |
|     B     |    1    |    1    |  ACCOUNTING  |
|     D     |    2    |    2    |    FINANCE   |
|     E     |    1    |    1    |  ACCOUNTING  |
|     F     |    1    |    1    |  ACCOUNTING  |
|           |         |    5    |   MARKETING  |
|           |         |    6    |      HR      |
+-----------+---------+---------+--------------+
```

Эквивалент синтаксиса Oracle (+) для запроса:

```
SELECT *
FROM EMPLOYEE, DEPT
WHERE EMPLOYEE.DEPTNO(+) = DEPT.DEPTNO;
```

ПОЛНОЕ ВНЕШНЕЕ СОЕДИНЕНИЕ

`FULL OUTER JOIN` выполняет соединение между двумя таблицами, для которых требуется явное предложение соединения, но не исключает непревзойденные строки в любой таблице. Другими словами, он возвращает все строки в каждой таблице.

Пример:

```
SELECT
      *
FROM
      EMPLOYEE FULL OUTER JOIN DEPT
      ON EMPLOYEE.DEPTNO = DEPT.DEPTNO;
```

Вот пример полного внешнего соединения между двумя таблицами:

Пример таблицы: EMPLOYEE

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     A     |    2    |
|     B     |    1    |
|     C     |    3    |
|     D     |    2    |
|     E     |    1    |
|     F     |    1    |
|     G     |    4    |
|     H     |    4    |
+-----------+---------+
```

Пример таблицы: DEPT

```
+---------+--------------+
|  DEPTNO |   DEPTNAME   |
+---------+--------------+
|    1    |  ACCOUNTING  |
|    2    |    FINANCE   |
|    5    |   MARKETING  |
|    6    |      HR      |
+---------+--------------+
```

Теперь, если вы выполните запрос:

```
SELECT
      *
FROM
      EMPLOYEE FULL OUTER JOIN DEPT
      ON EMPLOYEE.DEPTNO = DEPT.DEPTNO;
```

Выход

```
+-----------+---------+---------+--------------+
|    NAME   |  DEPTNO |  DEPTNO |   DEPTNAME   |
+-----------+---------+---------+--------------+
|     A     |    2    |    2    |    FINANCE   |
|     B     |    1    |    1    |  ACCOUNTING  |
|     C     |    3    |         |              |
|     D     |    2    |    2    |    FINANCE   |
|     E     |    1    |    1    |  ACCOUNTING  |
|     F     |    1    |    1    |  ACCOUNTING  |
|     G     |    4    |         |              |
|     H     |    4    |         |              |
|           |         |    6    |      HR      |
|           |         |    5    |   MARKETING  |
+-----------+---------+---------+--------------+
```

Здесь столбцы, которые не совпадают, были сохранены NULL.

ANTIJOIN

Antijoin возвращает строки с левой стороны предиката, для которых нет соответствующих строк в правой части предиката. Он возвращает строки, которые не соответствуют (НЕ В) подзапросам с правой стороны.

```
SELECT * FROM employees
   WHERE department_id NOT IN
   (SELECT department_id FROM departments
       WHERE location_id = 1700)
   ORDER BY last_name;
```

Вот пример Anti Join между двумя таблицами:

Пример таблицы: EMPLOYEE

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     A     |    2    |
|     B     |    1    |
|     C     |    3    |
|     D     |    2    |
|     E     |    1    |
|     F     |    1    |
|     G     |    4    |
|     H     |    4    |
+-----------+---------+
```

Пример таблицы: DEPT

```
+---------+--------------+
|  DEPTNO |   DEPTNAME   |
+---------+--------------+
|    1    |  ACCOUNTING  |
|    2    |    FINANCE   |
|    5    |   MARKETING  |
|    6    |      HR      |
+---------+--------------+
```

Теперь, если вы выполните запрос:

```
SELECT
      *
FROM
      EMPLOYEE WHERE DEPTNO NOT IN
      (SELECT DEPTNO FROM DEPT);
```

Выход:

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
```

```
|     C     |    3    |
|     H     |    4    |
|     G     |    4    |
+-----------+---------+
```

Вывод показывает, что только строки таблицы EMPLOYEE, из которых DEPTNO не присутствуют в таблице DEPT.

SEMIJOIN

Например, запрос semijoin можно найти, чтобы найти все отделы с хотя бы одним сотрудником, чья зарплата превышает 2500.

```
SELECT * FROM departments
   WHERE EXISTS
   (SELECT 1 FROM employees
       WHERE departments.department_id = employees.department_id
       AND employees.salary > 2500)
   ORDER BY department_name;
```

Это более эффективно, чем варианты полного объединения, поскольку внутреннее присоединение к сотрудникам, а затем предоставление предложения о том, что зарплата должна быть больше 2500, может возвращать тот же отдел несколько раз. Скажем, если в отделе пожарной охраны есть `n` сотрудников с зарплатой 3000, `select * from departments, employees` с необходимым объединением в ids, а наше предложение where вернет пожарную службу `n` раз.

ПРИСОЕДИНИТЬСЯ

Операция `JOIN` выполняет соединение между двумя таблицами, исключая любые несогласованные строки из первой таблицы. Начиная с Oracle 9i forward, `JOIN` эквивалентен функции `INNER JOIN` . Эта операция требует явного условия соединения, в отличие от операторов `CROSS JOIN` и `NATURAL JOIN` .

Пример:

```
select t1.*,
       t2.DeptId
  from table_1 t1
  join table_2 t2 on t2.DeptNo = t1.DeptNo
```

Документация Oracle:

- 10г
- 11g
- 12g

ПРИРОДНОЕ СОЕДИНЕНИЕ

ПРИРОДНЫЙ ПРИСОЕДИНЕНИЕ не требует никакого объяснительного условия соединения; он строит один на основе всех полей с тем же именем в объединенных таблицах.

```
create table tab1(id number,  descr varchar2(100));
create table tab2(id number,  descr varchar2(100));
insert into tab1 values(1, 'one');
insert into tab1 values(2, 'two');
insert into tab1 values(3, 'three');
insert into tab2 values(1, 'ONE');
insert into tab2 values(3, 'three');
```

Соединение будет выполнено по идентификаторам полей и DESCR, общим для обеих таблиц:

```
SQL> select *
  2  from tab1
  3       natural join
  4      tab2;

        ID DESCR
---------- ----------
         3 three
```

Столбцы с разными именами не будут использоваться в состоянии JOIN:

```
SQL> select *
  2  from (select id as id, descr as descr1 from tab1)
  3       natural join
  4       (select id as id, descr as descr2 from tab2);

        ID DESCR1     DESCR2
---------- ---------- ----------
         1 one        ONE
         3 three      three
```

Если объединенные таблицы не имеют общих столбцов, будет выполнен JOIN без каких-либо условий:

```
SQL> select *
  2  from (select id as id1, descr as descr1 from tab1)
  3       natural join
  4       (select id as id2, descr as descr2 from tab2);

       ID1 DESCR1            ID2 DESCR2
---------- ---------- ---------- ----------
         1 one                 1 ONE
         2 two                 1 ONE
         3 three               1 ONE
         1 one                 3 three
         2 two                 3 three
         3 three               3 three
```

Прочитайте JOINS онлайн: https://riptutorial.com/ru/oracle/topic/4192/joins

глава 3: Oracle Advanced Queuing (AQ)

замечания

- Никогда не используйте DDL или DML для таблиц, созданных `dbms_aqadm.create_queue_table` . Используйте только dbms_aqadm и dbms_aq для работы с этими таблицами. Oracle может создавать несколько поддерживающих таблиц, индексов и т. Д., О которых вы не будете знать. Ручное управление DDL или DML с таблицей может привести к сценарию, в котором Oracle Support вам понадобится, чтобы удалить и воссоздать таблицу и очереди, чтобы решить эту проблему.
- Настоятельно рекомендуется не использовать параметр `dbms_aq.forever` для ожидания. Это вызвало проблемы в прошлом, поскольку Oracle может начать планировать чрезмерное количество рабочих заданий для работы ненужных очередей (см. Oracle Doc ID 2001165.1).
- Рекомендуется не устанавливать параметр AQ_TM_PROCESSES в версии 10.1 и новее. Особенно избегайте установки этого значения в ноль, поскольку это отключит фоновое задание QMON, которое необходимо для поддержания очередей. Вы можете сбросить это значение до значения по умолчанию Oracle, используя следующую команду и перезагрузив базу данных. `alter system reset aq_tm_processes scope=spfile sid='*';`

Examples

Простой производитель / потребитель

обзор

Создайте очередь, на которую мы можем отправить сообщение. Oracle уведомит нашу хранимую процедуру о том, что сообщение было помещено в очередь и должно быть обработано. Мы также добавим некоторые подпрограммы, которые мы можем использовать в экстренной ситуации, чтобы не допустить, чтобы сообщения были отклонены, снова разрешить переопределение и запустить простое пакетное задание для работы через все сообщения.

Эти примеры были протестированы на Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production.

Создать очередь

Мы создадим тип сообщения, таблицу очередей, в которой могут храниться сообщения, и

очереди. Сообщения в очереди будут сначала дезертированы по приоритету, а затем их время очереди. Если что-то пойдет не так, работая над сообщением, a dequeue будет откат, AQ сделает сообщение доступным для dequeue 3600 секунд спустя. Он будет делать это 48 раз, прежде чем перемещать его в очередь исключений.

```
create type message_t as object
   (
   sender  varchar2 ( 50 ),
   message varchar2 ( 512 )
   );
/
-- Type MESSAGE_T compiled
begin dbms_aqadm.create_queue_table(
     queue_table        => 'MESSAGE_Q_TBL',
     queue_payload_type => 'MESSAGE_T',
     sort_list          => 'PRIORITY,ENQ_TIME',
     multiple_consumers =>  false,
     compatible         => '10.0.0');
  end;
/
-- PL/SQL procedure successfully completed.
begin dbms_aqadm.create_queue(
     queue_name          => 'MESSAGE_Q',
     queue_table         => 'MESSAGE_Q_TBL',
     queue_type          =>  0,
     max_retries         =>  48,
     retry_delay         =>  3600,
     dependency_tracking =>  false);
  end;
/
-- PL/SQL procedure successfully completed.
```

Теперь, когда у нас есть место для размещения сообщений, вы можете создать пакет для управления и работы сообщений в очереди.

```
create or replace package message_worker_pkg
is
   queue_name_c constant varchar2(20) := 'MESSAGE_Q';

   -- allows the workers to process messages in the queue
   procedure enable_dequeue;

   -- prevents messages from being worked but will still allow them to be created and enqueued
   procedure disable_dequeue;

   -- called only by Oracle Advanced Queueing.  Do not call anywhere else.
   procedure on_message_enqueued (context        in raw,
                                  reginfo        in sys.aq$_reg_info,
                                  descr          in sys.aq$_descriptor,
                                  payload        in raw,
                                  payloadl       in number);

   -- allows messages to be worked if we missed the notification (or a retry
   -- is pending)
   procedure work_old_messages;

end;
```

```
/

create or replace package body message_worker_pkg
is
   -- raised by Oracle when we try to dequeue but no more messages are ready to
   -- be dequeued at this moment
   no_more_messages_ex          exception;
   pragma exception_init (no_more_messages_ex,
                          -25228);

   -- allows the workers to process messages in the queue
   procedure enable_dequeue
   as
   begin
      dbms_aqadm.start_queue (queue_name => queue_name_c, dequeue => true);
   end enable_dequeue;

   -- prevents messages from being worked but will still allow them to be created and enqueued
   procedure disable_dequeue
   as
   begin
      dbms_aqadm.stop_queue (queue_name => queue_name_c, dequeue => true, enqueue => false);
   end disable_dequeue;

   procedure work_message (message_in in out nocopy message_t)
   as
   begin
      dbms_output.put_line ( message_in.sender || ' says ' || message_in.message );
   end work_message;

   -- called only by Oracle Advanced Queueing.  Do not call anywhere else.

   procedure on_message_enqueued (context        in raw,
                                  reginfo        in sys.aq$_reg_info,
                                  descr          in sys.aq$_descriptor,
                                  payload        in raw,
                                  payloadl       in number)
   as
      pragma autonomous_transaction;
      dequeue_options_l      dbms_aq.dequeue_options_t;
      message_id_l           raw (16);
      message_l              message_t;
      message_properties_l   dbms_aq.message_properties_t;
   begin
      dequeue_options_l.msgid         := descr.msg_id;
      dequeue_options_l.consumer_name := descr.consumer_name;
      dequeue_options_l.wait          := dbms_aq.no_wait;
      dbms_aq.dequeue (queue_name           => descr.queue_name,
                       dequeue_options      => dequeue_options_l,
                       message_properties   => message_properties_l,
                       payload              => message_l,
                       msgid                => message_id_l);
      work_message (message_l);
      commit;
   exception
      when no_more_messages_ex
      then
         -- it's possible work_old_messages already dequeued the message
         commit;
      when others
      then
```

```
         -- we don't need to have a raise here.  I just wanted to point out that
         -- since this will be called by AQ throwing the exception back to it
         -- will have it put the message back on the queue and retry later
         raise;
   end on_message_enqueued;

   -- allows messages to be worked if we missed the notification (or a retry
   -- is pending)
   procedure work_old_messages
   as
      pragma autonomous_transaction;
      dequeue_options_l      dbms_aq.dequeue_options_t;
      message_id_l           raw (16);
      message_l              message_t;
      message_properties_l   dbms_aq.message_properties_t;
   begin
      dequeue_options_l.wait       := dbms_aq.no_wait;
      dequeue_options_l.navigation := dbms_aq.first_message;

      while (true) loop -- way out is no_more_messages_ex
         dbms_aq.dequeue (queue_name           => queue_name_c,
                          dequeue_options      => dequeue_options_l,
                          message_properties   => message_properties_l,
                          payload              => message_l,
                          msgid                => message_id_l);
         work_message (message_l);
         commit;
      end loop;
   exception
      when no_more_messages_ex
      then
         null;
   end work_old_messages;
end;
```

Затем скажите AQ, что, когда сообщение помещено в MESSAGE_Q (и зафиксировано), уведомление о нашей процедуре должно быть выполнено. AQ запустит работу на своем собственном сеансе, чтобы справиться с этим.

```
begin
  dbms_aq.register (
     sys.aq$_reg_info_list (
        sys.aq$_reg_info (user || '.' || message_worker_pkg.queue_name_c,
                          dbms_aq.namespace_aq,
                          'plsql://' || user || '.message_worker_pkg.on_message_enqueued',
                          hextoraw ('FF'))),
     1);
  commit;
end;
```

Запустить очередь и отправить сообщение

```
declare
   enqueue_options_l      dbms_aq.enqueue_options_t;
   message_properties_l   dbms_aq.message_properties_t;
   message_id_l           raw (16);
```

```
   message_l              message_t;
begin
   -- only need to do this next line ONCE
   dbms_aqadm.start_queue (queue_name => message_worker_pkg.queue_name_c, enqueue => true ,
dequeue => true);

   message_l := new message_t ( 'Jon', 'Hello, world!' );
   dbms_aq.enqueue (queue_name           => message_worker_pkg.queue_name_c,
                    enqueue_options      => enqueue_options_l,
                    message_properties   => message_properties_l,
                    payload              => message_l,
                    msgid                => message_id_l);
   commit;
end;
```

Прочитайте Oracle Advanced Queuing (AQ) онлайн:
https://riptutorial.com/ru/oracle/topic/4362/oracle-advanced-queuing--aq-

глава 4: Oracle MAF

Examples

Чтобы получить значение от привязки

```
        ValueExpression ve = AdfmfJavaUtilities.getValueExpression(<binding>, String.class);
       String <variable_name> = (String) ve.getValue(AdfmfJavaUtilities.getELContext());
```

Здесь «привязка» указывает выражение EL, из которого должно быть получено значение.

"variable_name" параметр, для которого сохраняется значение из привязки

Чтобы установить значение привязки

```
        ValueExpression ve = AdfmfJavaUtilities.getValueExpression(<binding>, String.class);
        ve.setValue(AdfmfJavaUtilities.getELContext(), <value>);
```

Здесь «привязка» указывает выражение EL, значение которого должно быть сохранено.

«значение» - это желаемое значение для добавления к привязке

Вызов метода из привязки

```
    AdfELContext adfELContext = AdfmfJavaUtilities.getAdfELContext();
    MethodExpression me;
    me = AdfmfJavaUtilities.getMethodExpression(<binding>, Object.class, new Class[] { });
    me.invoke(adfELContext, new Object[] { });
```

«binding» указывает выражение EL, из которого вызывается метод

Чтобы вызвать функцию javaScript

```
AdfmfContainerUtilities.invokeContainerJavaScriptFunction(AdfmfJavaUtilities.getFeatureId(),
<function>, new Object[] {
                                                        });
```

«function» - это желаемая функция js, которая должна быть вызвана

Прочитайте Oracle MAF онлайн: https://riptutorial.com/ru/oracle/topic/6352/oracle-maf

глава 5: Автономные транзакции

замечания

Типичные варианты использования автономной транзакции.

1. Для построения любой структуры ведения журнала, такой как структура регистрации ошибок, описанная в приведенном выше примере.
2. Для аудита операций DML в триггерах на таблицах, независимо от конечного состояния транзакции (COMMIT или ROLLBACK).

Examples

Использование автономной транзакции для протоколирования ошибок

Следующая процедура является общей, которая будет использоваться для регистрации всех ошибок в приложении в общей таблице журнала ошибок.

```
CREATE OR REPLACE PROCEDURE log_errors
(
  p_calling_program IN VARCHAR2,
  p_error_code IN INTEGER,
  p_error_description IN VARCHAR2
)
IS
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  INSERT INTO error_log
  VALUES
  (
  p_calling_program,
  p_error_code,
  p_error_description,
  SYSDATE,
  USER
  );
  COMMIT;
END log_errors;
```

Следующий анонимный блок PLSQL показывает, как вызвать процедуру log_errors.

```
BEGIN
   DELETE FROM dept WHERE deptno = 10;
EXCEPTION
   WHEN OTHERS THEN
      log_errors('Delete dept',sqlcode, sqlerrm);
      RAISE;
END;

SELECT * FROM error_log;
```

```
CALLING_PROGRAM    ERROR_CODE    ERROR_DESCRIPTION
ERROR_DATETIME         DB_USER
Delete dept        -2292         ORA-02292: integrity constraint violated - child record found
08/09/2016             APEX_PUBLIC_USER
```

Прочитайте Автономные транзакции онлайн: https://riptutorial.com/ru/oracle/topic/6103/автономные-транзакции

глава 6: Анонимный блок PL / SQL

замечания

Поскольку они неназванные, анонимные блоки не могут ссылаться на другие программные единицы.

Examples

Пример анонимного блока

```
DECLARE
    -- declare a variable
    message varchar2(20);
BEGIN
  -- assign value to variable
  message := 'HELLO WORLD';

  -- print message to screen
  DBMS_OUTPUT.PUT_LINE(message);
END;
/
```

Прочитайте Анонимный блок PL / SQL онлайн: https://riptutorial.com/ru/oracle/topic/6451/анонимный-блок-pl---sql

глава 7: Даты

Examples

Создание дат без компонента времени

Все `DATE` имеют временной компонент; однако обычно принято хранить даты, которые не обязательно должны включать информацию о времени с часами / минутами / секундами, установленными на ноль (т.е. полночь).

Используйте литерал ANSI `DATE` (с использованием формата даты ISO 8601):

```
SELECT DATE '2000-01-01' FROM DUAL;
```

Преобразуйте его из строкового литерала с помощью `TO_DATE()` :

```
SELECT TO_DATE( '2001-01-01', 'YYYY-MM-DD' ) FROM DUAL;
```

(Более подробную информацию о моделях формата даты можно найти в документации Oracle.)

или же:

```
SELECT TO_DATE(
         'January 1, 2000, 00:00 A.M.',
         'Month dd, YYYY, HH12:MI A.M.',
         'NLS_DATE_LANGUAGE = American'
       )
FROM   DUAL;
```

(Если вы конвертируете специфические для языка термины, такие как имена месяцев, то хорошей практикой является включение `nlsparam` параметра `nlsparam` в функцию `TO_DATE()` и указание языка, который следует ожидать.)

Создание дат с помощью компонента времени

Преобразуйте его из строкового литерала с помощью `TO_DATE()` :

```
SELECT TO_DATE( '2000-01-01 12:00:00', 'YYYY-MM-DD HH24:MI:SS' ) FROM DUAL;
```

Или используйте литерал `TIMESTAMP` :

```
CREATE TABLE date_table(
  date_value DATE
);
```

```
INSERT INTO date_table ( date_value ) VALUES ( TIMESTAMP '2000-01-01 12:00:00' );
```

Oracle будет неявным образом использовать `TIMESTAMP` для `DATE` при хранении в столбце `DATE` таблицы; однако вы можете явно `CAST()` значение `CAST()` для `DATE` :

```
SELECT CAST( TIMESTAMP '2000-01-01 12:00:00' AS DATE ) FROM DUAL;
```

Формат даты

В Oracle тип данных `DATE` не имеет формата; когда Oracle отправляет `DATE` в клиентскую программу (SQL / Plus, SQL / Developer, Toad, Java, Python и т. д.), он отправит 7- или 8-байты, которые представляют дату.

`DATE` которая не хранится в таблице (т.е. сгенерирована `SYSDATE` и имеет тип 13 при использовании команды `DUMP()`) имеет 8 байтов и имеет структуру (числа справа являются внутренним представлением `2012-11-26 16:41:09`):

```
BYTE VALUE                          EXAMPLE
---- ------------------------------ --------------------------------------
1    Year modulo 256                220
2    Year multiples of 256          7   (7 * 256 + 220 = 2012)
3    Month                          11
4    Day                            26
5    Hours                          16
6    Minutes                        41
7    Seconds                        9
8    Unused                         0
```

`DATE` который хранится в таблице («тип 12» при использовании команды `DUMP()`) имеет 7 байтов и имеет структуру (числа справа являются внутренним представлением `2012-11-26 16:41:09`):

```
BYTE VALUE                          EXAMPLE
---- ------------------------------ --------------------------------------
1    ( Year multiples of 100 ) + 100 120
2    ( Year modulo 100 ) + 100      112 ((120-100)*100 + (112-100) = 2012)
3    Month                          11
4    Day                            26
5    Hours + 1                      17
6    Minutes + 1                    42
7    Seconds + 1                    10
```

Если вы хотите, чтобы дата имела определенный формат, вам нужно будет преобразовать ее в то, что имеет формат (т. Е. Строку). Клиент SQL может косвенно выполнять это или вы можете явно преобразовать значение в строку с использованием `TO_CHAR( date, format_model, nls_params )` .

Преобразование дат в строку

Используйте `TO_CHAR( date [, format_model [, nls_params]] )` :

(Примечание: если модель формата не `NLS_DATE_FORMAT` параметр сеанса `NLS_DATE_FORMAT` будет использоваться в качестве модели формата по умолчанию , это может быть различным для каждого сеанса, поэтому не следует полагаться. Хорошая практика всегда указывать модель формата.)

```
CREATE TABLE table_name (
  date_value DATE
);

INSERT INTO table_name ( date_value ) VALUES ( DATE '2000-01-01' );
INSERT INTO table_name ( date_value ) VALUES ( TIMESTAMP '2016-07-21 08:00:00' );
INSERT INTO table_name ( date_value ) VALUES ( SYSDATE );
```

Затем:

```
SELECT TO_CHAR( date_value, 'YYYY-MM-DD' ) AS formatted_date FROM table_name;
```

Выходы:

```
FORMATTED_DATE
--------------
2000-01-01
2016-07-21
2016-07-21
```

А также:

```
SELECT TO_CHAR(
         date_value,
         'FMMonth d yyyy, hh12:mi:ss AM',
         'NLS_DATE_LANGUAGE = French'
       ) AS formatted_date
FROM   table_name;
```

Выходы:

```
FORMATTED_DATE
-----------------------------
Janvier   01 2000, 12:00:00 AM
Juillet   21 2016, 08:00:00 AM
Juillet   21 2016, 19:08:31 PM
```

Установка модели формата даты по умолчанию

Когда Oracle неявно преобразует из `DATE` в строку или наоборот (или когда `TO_CHAR()` или `TO_DATE()` явно вызываются без модели формата), параметр сеанса `NLS_DATE_FORMAT` будет использоваться в качестве модели формата при преобразовании. Если литерал не соответствует модели формата, будет создано исключение.

Вы можете просмотреть этот параметр, используя:

```
SELECT VALUE FROM NLS_SESSION_PARAMETERS WHERE PARAMETER = 'NLS_DATE_FORMAT';
```

Вы можете установить это значение в текущем сеансе, используя:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD HH24:MI:SS';
```

(Примечание: это не изменяет значение для других пользователей.)

Если вы полагаетесь на `NLS_DATE_FORMAT` для предоставления маски формата в `TO_DATE()` или `TO_CHAR()` вы не должны удивляться, когда ваши запросы ломаются, если это значение когда-либо изменяется.

Изменение того, как SQL / Plus или SQL Developer отображают даты

Когда SQL / Plus или SQL Developer отображают даты, они будут выполнять неявное преобразование в строку с использованием модели формата даты по умолчанию (см. Пример « Установка образца модели даты по умолчанию»).

Вы можете изменить способ отображения даты, изменив параметр `NLS_DATE_FORMAT` .

Арифметика даты - разница между датами в днях, часах, минутах и / или секундах

В оракуле разница (в днях и / или их фракциях) между двумя `DATE` s может быть найдена с помощью вычитания:

```
SELECT DATE '2016-03-23' - DATE '2015-12-25' AS difference FROM DUAL;
```

Выводит количество дней между двумя датами:

```
DIFFERENCE
----------
        89
```

А также:

```
SELECT TO_DATE( '2016-01-02 01:01:12', 'YYYY-MM-DD HH24:MI:SS' )
         - TO_DATE( '2016-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS' )
         AS difference
FROM   DUAL
```

Выводит долю дней между двумя датами:

```
DIFFERENCE
----------
```

```
    1.0425
```

Разницу в часах, минутах или секундах можно найти, умножив это число на `24` , `24*60` или `24*60*60` соответственно.

Предыдущий пример можно изменить, чтобы получить дни, часы, минуты и секунды между двумя датами, используя:

```
SELECT TRUNC( difference                       ) AS days,
       TRUNC( MOD( difference * 24,       24 ) ) AS hours,
       TRUNC( MOD( difference * 24*60,    60 ) ) AS minutes,
       TRUNC( MOD( difference * 24*60*60, 60 ) ) AS seconds
FROM   (
  SELECT TO_DATE( '2016-01-02 01:01:12', 'YYYY-MM-DD HH24:MI:SS' )
         - TO_DATE( '2016-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS' )
         AS difference
  FROM   DUAL
);
```

(Примечание: `TRUNC()` используется вместо `FLOOR()` чтобы правильно обрабатывать отрицательные отличия.)

Выходы:

```
DAYS HOURS MINUTES SECONDS
---- ----- ------- -------
   1     1       1      12
```

Предыдущий пример также может быть решен путем преобразования числовой разности в интервал, используя `NUMTODSINTERVAL()` :

```
SELECT EXTRACT( DAY    FROM difference ) AS days,
       EXTRACT( HOUR   FROM difference ) AS hours,
       EXTRACT( MINUTE FROM difference ) AS minutes,
       EXTRACT( SECOND FROM difference ) AS seconds
FROM   (
  SELECT NUMTODSINTERVAL(
           TO_DATE( '2016-01-02 01:01:12', 'YYYY-MM-DD HH24:MI:SS' )
             - TO_DATE( '2016-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS' ),
           'DAY'
         ) AS difference
  FROM   DUAL
);
```

Арифметика даты - разница между датами в месяцах или годах

Разницу в месяцах между двумя датами можно найти с помощью `MONTHS_BETWEEN( date1, date2 )` :

```
SELECT MONTHS_BETWEEN( DATE '2016-03-10', DATE '2015-03-10' ) AS difference FROM DUAL;
```

Выходы:

```
DIFFERENCE
----------
        12
```

Если разница включает частичные месяцы, тогда она вернет часть месяца на основании **31** дня в каждом месяце:

```
SELECT MONTHS_BETWEEN( DATE '2015-02-15', DATE '2015-01-01' ) AS difference FROM DUAL;
```

Выходы:

```
DIFFERENCE
----------
 1.4516129
```

Из-за `MONTHS_BETWEEN` предполагающего 31 день в месяц, когда может быть меньше дней в месяц, это может привести к разным значениям различий, охватывающих границы между месяцами.

Пример:

```
SELECT MONTHS_BETWEEN( DATE'2016-02-01', DATE'2016-02-01' - INTERVAL '1' DAY ) AS "JAN-FEB",
       MONTHS_BETWEEN( DATE'2016-03-01', DATE'2016-03-01' - INTERVAL '1' DAY ) AS "FEB-MAR",
       MONTHS_BETWEEN( DATE'2016-04-01', DATE'2016-04-01' - INTERVAL '1' DAY ) AS "MAR-APR",
       MONTHS_BETWEEN( DATE'2016-05-01', DATE'2016-05-01' - INTERVAL '1' DAY ) AS "APR-MAY"
FROM   DUAL;
```

Выход:

```
JAN-FEB FEB-MAR MAR-APR APR-MAY
------- ------- ------- -------
0.03226 0.09677 0.03226 0.06452
```

Разницу в годах можно найти, разделив разницу в месяц на 12.

Извлечение года, месяца, дня, часа, минут или вторых компонентов даты

Компоненты года, месяца или дня типа данных `DATE` можно найти, используя `EXTRACT( [ YEAR | MONTH | DAY ] FROM datevalue )`

```
SELECT EXTRACT (YEAR  FROM DATE '2016-07-25') AS YEAR,
       EXTRACT (MONTH FROM DATE '2016-07-25') AS MONTH,
       EXTRACT (DAY   FROM DATE '2016-07-25') AS DAY
FROM DUAL;
```

Выходы:

```
YEAR MONTH DAY
---- ----- ---
2016     7  25
```

Компоненты времени (час, минута или секунда) могут быть найдены либо:

- Использование `CAST( datevalue AS TIMESTAMP )` для преобразования `DATE` в `TIMESTAMP` а затем с помощью `EXTRACT( [ HOUR | MINUTE | SECOND ] FROM timestampvalue )` ; или же
- Использование `TO_CHAR( datevalue, format_model )` для получения значения в виде строки.

Например:

```
SELECT EXTRACT( HOUR   FROM CAST( datetime AS TIMESTAMP ) ) AS Hours,
       EXTRACT( MINUTE FROM CAST( datetime AS TIMESTAMP ) ) AS Minutes,
       EXTRACT( SECOND FROM CAST( datetime AS TIMESTAMP ) ) AS Seconds
FROM   (
  SELECT TO_DATE( '2016-01-01 09:42:01', 'YYYY-MM-DD HH24:MI:SS' ) AS datetime FROM DUAL
);
```

Выходы:

```
HOURS MINUTES SECONDS
----- ------- -------
    9      42       1
```

Временные зоны и летнее время

Тип данных `DATE` не обрабатывает часовые пояса или изменения в летнее время.

Или:

- используйте тип данных `TIMESTAMP WITH TIME ZONE` ; или же
- обрабатывать изменения в логике вашего приложения.

`DATE` может быть сохранен как скоординированное универсальное время (UTC) и преобразован в текущий часовой пояс сеанса следующим образом:

```
SELECT FROM_TZ(
         CAST(
           TO_DATE( '2016-01-01 12:00:00', 'YYYY-MM-DD HH24:MI:SS' )
           AS TIMESTAMP
         ),
         'UTC'
       )
       AT LOCAL AS time
FROM   DUAL;
```

Если вы запустите `ALTER SESSION SET TIME_ZONE = '+01:00';` то выход:

```
TIME
-----------------------------------
2016-01-01 13:00:00.000000000 +01:00
```

и `ALTER SESSION SET TIME_ZONE = 'PST';` то выход:

```
TIME
-----------------------------------
2016-01-01 04:00:00.000000000 PST
```

Секундомер

Oracle не обрабатывает прыжки секунд . Дополнительную информацию см. В примечаниях к поддержке `2019397.2` и `730795.1` .

Получение дня недели

Вы можете использовать `TO_CHAR( date_value, 'D' )` чтобы получить день недели.

Однако это зависит от параметра сеанса `NLS_TERRITORY` :

```
ALTER SESSION SET NLS_TERRITORY = 'AMERICA';        -- First day of week is Sunday
SELECT TO_CHAR( DATE '1970-01-01', 'D' ) FROM DUAL;
```

Выходы `5`

```
ALTER SESSION SET NLS_TERRITORY = 'UNITED KINGDOM'; -- First day of week is Monday
SELECT TO_CHAR( DATE '1970-01-01', 'D' ) FROM DUAL;
```

Выходы `4`

Чтобы сделать это независимо от настроек `NLS` , вы можете усечь дату до полуночи текущего дня (чтобы удалить любые доли дней) и вычесть дату, усеченную до начала текущей недели iso-week (которая всегда начинается в понедельник):

```
SELECT TRUNC( date_value ) - TRUNC( date_value, 'IW' ) + 1 FROM DUAL
```

Прочитайте Даты онлайн: https://riptutorial.com/ru/oracle/topic/2087/даты

глава 8: Динамический SQL

Вступление

Dynamic SQL позволяет собирать код SQL-запроса во время выполнения. Этот метод имеет некоторые недостатки и должен использоваться очень осторожно. В то же время он позволяет реализовать более сложную логику. PL / SQL требует, чтобы все объекты, используемые в коде, должны существовать и быть действительными во время компиляции. Вот почему вы не можете напрямую выполнять инструкции DDL в PL / SQL, но динамический SQL позволяет это сделать.

замечания

Некоторые важные замечания:

1. Никогда не используйте конкатенацию строк для добавления значений к запросу, вместо этого используйте параметры. Это не верно:

   ```
   execute immediate 'select value from my_table where id = ' ||
        id_valiable into result_variable;
   ```

   И это правильно:

   ```
   execute immediate 'select value from my_table where id = :P '
       using id_valiable into result_variable;
   ```

   Для этого есть две причины. Первая - это безопасность. Конкатенация строк позволяет сделать SQL-инъекцию. В запросе ниже, если переменная будет содержать значение `1 or 1 = 1` , оператор `UPDATE` обновит все строки в таблице:

   ```
   execute immediate 'update my_table set value = ''I have bad news for you'' where id = '
   || id;
   ```

   Вторая причина - производительность. Oracle будет анализировать запрос без параметров каждый раз, когда он выполняется, тогда как запрос с параметром будет анализироваться только один раз в сеансе.

2. Обратите внимание, что когда механизм базы данных выполняет оператор DDL, он выполняет неявное совершение ранее.

Examples

Выберите значение с динамическим SQL

Предположим, пользователь хочет выбрать данные из разных таблиц. Пользователь задает таблицу.

```
 function get_value(p_table_name varchar2, p_id number) return varchar2 is
    value varchar2(100);
  begin
    execute immediate 'select column_value from ' || p_table_name ||
                      ' where id = :P' into value using p_id;
    return value;
  end;
```

Вызовите эту функцию как обычно:

```
declare
  table_name varchar2(30) := 'my_table';
  id number := 1;
begin
  dbms_output.put_line(get_value(table_name, id));
end;
```

Таблица для тестирования:

```
create table my_table (id number, column_value varchar2(100));
insert into my_table values (1, 'Hello, world!');
```

Вставка значений в динамический SQL

Пример ниже вставляет значение в таблицу из предыдущего примера:

```
declare
  query_text varchar2(1000) := 'insert into my_table(id, column_value) values (:P_ID,
:P_VAL)';
  id number := 2;
  value varchar2(100) := 'Bonjour!';
begin
  execute immediate query_text using id, value;
end;
/
```

Обновить значения в динамическом SQL

Давайте обновим таблицу из первого примера:

```
declare
  query_text varchar2(1000) := 'update my_table set column_value = :P_VAL where id = :P_ID';
  id number := 2;
  value varchar2(100) := 'Bonjour le monde!';
begin
  execute immediate query_text using value, id;
end;
/
```

Выполнить инструкцию DDL

Этот код создает таблицу:

```
begin
  execute immediate 'create table my_table (id number, column_value varchar2(100))';
end;
/
```

Выполнять анонимный блок

Вы можете выполнить анонимный блок. В этом примере также показано, как вернуть значение из динамического SQL:

```
declare
  query_text varchar2(1000) := 'begin :P_OUT := cos(:P_IN); end;';
  in_value number := 0;
  out_value number;
begin
  execute immediate query_text using out out_value, in in_value;
  dbms_output.put_line('Result of anonymous block: ' || to_char(out_value));
end;
/
```

Прочитайте Динамический SQL онлайн: https://riptutorial.com/ru/oracle/topic/10905/динамический-sql

глава 9: Иерархический поиск с помощью Oracle Database 12C

Вступление

Вы можете использовать иерархические запросы для извлечения данных на основе естественной иерархической связи между строками в таблице

Examples

Использование CONNECT BY Caluse

```
SELECT E.EMPLOYEE_ID,E.LAST_NAME,E.MANAGER_ID FROM HR.EMPLOYEES E
CONNECT BY PRIOR E.EMPLOYEE_ID = E.MANAGER_ID;
```

Предложение `CONNECT BY` определяет отношения между сотрудниками и менеджерами.

Указание направления запроса сверху вниз

```
SELECT E.LAST_NAME|| ' reports to ' ||
PRIOR E.LAST_NAME "Walk Top Down"
FROM HR.EMPLOYEES E
START WITH E.MANAGER_ID IS NULL
CONNECT BY PRIOR E.EMPLOYEE_ID = E.MANAGER_ID;
```

Прочитайте Иерархический поиск с помощью Oracle Database 12C онлайн: https://riptutorial.com/ru/oracle/topic/8777/иерархический-поиск-с-помощью-oracle-database-12c

глава 10: Индексы

Вступление

Здесь я объясню другой пример, используя пример, как повысить производительность запросов индекса, как индекс уменьшает производительность DML и т. Д.

Examples

индекс b-дерева

```
CREATE INDEX ord_customer_ix ON orders (customer_id);
```

По умолчанию, если мы ничего не упоминаем, oracle создает индекс в качестве индекса b-дерева. Но мы должны знать, когда его использовать. Индекс B-дерева хранит данные в виде двоичного дерева. Как мы знаем, индекс - это объект схемы, который хранит какую-либо запись для каждого значения для индексированного столбца. Таким образом, всякий раз, когда в этих столбцах происходит поиск, он проверяет в индексе точное местоположение этой записи для быстрого доступа. Несколько вопросов об индексировании:

- Для поиска записи в индексе используется какой-то алгоритм бинарного поиска.
- Когда **мощность данных высока,** индекс **b-дерева** идеально подходит для использования.
- Индекс делает DML медленным, так как для каждой записи должна быть одна запись в индексе для индексированного столбца.
- Поэтому, если это не необходимо, нам следует избегать создания индекса.

Растровый индекс

```
CREATE BITMAP INDEX
emp_bitmap_idx
ON index_demo (gender);
```

- Индекс битмапа используется, когда **мощность данных низкая.**
- Здесь **гендер** имеет ценность с низкой мощностью. Ценностями могут быть мужчины, женщины и другие.
- Таким образом, если мы создадим двоичное дерево для этих 3 значений во время поиска, у него будет ненужный траверс.
- В растровых структурах создается двумерный массив с одним столбцом для каждой строки в индексируемой таблице. Каждый столбец представляет собой отдельное значение в растровом индексе. Этот двумерный массив представляет каждое

значение в индексе, умноженное на количество строк в таблице.
- При времени поиска строки Oracle распаковывает растровое изображение в буферы данных RAM, поэтому его можно быстро сканировать для сопоставления значений. Эти совпадающие значения доставляются в Oracle в виде списка Row-ID, и эти значения Row-ID могут напрямую обращаться к необходимой информации.

Функциональный индекс

```
CREATE INDEX first_name_idx ON user_data (UPPER(first_name));


SELECT *
FROM   user_data
WHERE  UPPER(first_name) = 'JOHN2';
```

- Функциональный индекс означает, создавая индекс, основанный на функции.
- Если в поиске (где предложение) часто используется какая-либо функция, лучше создать индекс, основанный на этой функции.
- Здесь, в примере, для поиска используется функция **Upper ()** . Таким образом, лучше создать индекс, используя верхнюю функцию.

Прочитайте Индексы онлайн: https://riptutorial.com/ru/oracle/topic/9978/индексы

глава 11: Манипуляция строк

Examples

Конкатенация: оператор || или concat ()

Oracle SQL и PL / SQL `||` оператор позволяет объединить две или более строки вместе.

Пример:

Предполагая следующую таблицу `customers` :

```
 id  firstname    lastname
---  -----------  ----------
  1  Thomas       Woody
```

Запрос:

```
 SELECT firstname || ' ' || lastname || ' is in my database.' as "My Sentence"
   FROM customers;
```

Выход:

```
My Sentence
--------------------------------
Thomas Woody is in my database.
```

Oracle также поддерживает стандартную функцию SQL `CONCAT(str1, str2)` :

Пример:

Запрос:

```
 SELECT CONCAT(firstname, ' is in my database.') from customers;
```

Выход:

```
Expr1
--------------------------------
Thomas is in my database.
```

ВЕРХНИЙ

Функция UPPER позволяет вам преобразовывать все строчные буквы в строке в верхний регистр.

```
SELECT UPPER('My text 123!') AS result FROM dual;
```

Выход:

```
RESULT
------------
MY TEXT 123!
```

INITCAP

Функция `INITCAP` преобразует случай строки, так что каждое слово начинается с заглавной буквы, а все последующие буквы имеют строчный регистр.

```
SELECT INITCAP('HELLO mr macdonald!') AS NEW FROM dual;
```

Выход

```
NEW
--------------------
Hello Mr Macdonald!
```

НИЖНИЙ

LOWER преобразует все прописные буквы в строку в нижний регистр.

```
SELECT LOWER('HELLO World123!') text FROM dual;
```

Выходы:

текст
привет мир123!

Регулярное выражение

Предположим, мы хотим заменить только цифры двумя цифрами: регулярное выражение найдет их с `(\d\d)`

```
SELECT REGEXP_REPLACE ('2, 5, and 10 are numbers in this example', '(\d\d)', '#')
FROM dual;
```

Результаты в:

```
'2, 5, and # are numbers in this example'
```

Если я хочу поменять части текста, я использую `\1` , `\2` , `\3` чтобы вызывать

соответствующие строки:

```
SELECT REGEXP_REPLACE ('swap around 10 in that one ', '(.*)(\d\d )(.*)', '\3\2\1\3')
FROM dual;
```

SUBSTR

`SUBSTR` извлекает часть строки, указывая начальную позицию и количество символов для извлечения

```
SELECT SUBSTR('abcdefg',2,3) FROM DUAL;
```

возвращает:

```
bcd
```

Для подсчета с конца строки `SUBSTR` принимает отрицательное число в качестве второго параметра, например

```
SELECT SUBSTR('abcdefg',-4,2) FROM DUAL;
```

возвращает:

```
de
```

Чтобы получить последний символ в строке: `SUBSTR(mystring,-1,1)`

LTRIM / RTRIM

`LTRIM` и `RTRIM` удаляют символы из начала или конца строки (соответственно). Может быть поставлен набор из одного или нескольких символов (по умолчанию это пробел) для удаления.

Например,

```
select LTRIM('<===>HELLO<===>', '=<>')
      ,RTRIM('<===>HELLO<===>', '=<>')
from dual;
```

Возвращает:

```
HELLO<===>
<===>HELLO
```

Прочитайте Манипуляция строк онлайн: https://riptutorial.com/ru/oracle/topic/1518/манипуляция-строк

глава 12: Насос данных

Вступление

Ниже приведены шаги по созданию импорта / экспорта данных:

Examples

Мониторинг рабочих мест Datapump

Задачи Datapump можно отслеживать, используя

1. просмотр словаря данных:

```
   select * from dba_datapump_jobs;
   SELECT * FROM DBA_DATAPUMP_SESSIONS;
   select username,opname,target_desc,sofar,totalwork,message from V$SESSION_LONGOPS where
username = 'bkpadmin';
```

2. Статус Datapump:

- Запишите имя задания из журналов импорта / экспорта или имени словаря данных и
- Запустите команду **attach** :
- статус типа в приглашении импорта / экспорта

```
impdp <bkpadmin>/<bkp123> attach=<SYS_IMPORT_SCHEMA_01>
Import> status
```

Нажмите **CTRL + C,** чтобы выйти из приглашения импорта / экспорта

Шаг 3/6: Создать каталог

```
create or replace directory DATAPUMP_REMOTE_DIR as '/oracle/scripts/expimp';
```

Шаг 7: Команды экспорта

Команды:

```
expdp <bkpadmin>/<bkp123>  parfile=<exp.par>
```

* Пожалуйста, замените данные в <> соответствующими значениями в соответствии с вашей средой. Вы можете добавлять / изменять параметры в соответствии с вашими требованиями. В приведенном выше примере все остальные параметры добавляются в файлы параметров, как указано ниже: *

- Тип экспорта: **экспорт пользователя**
- Экспорт всей схемы
- Сведения о файле параметров [say exp.par]:

```
schemas=<schema>
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>.dmp
logfile=exp_<dbname>_<schema>.log
```

- Тип экспорта: **Пользователь Экспорт для большой схемы**
- Экспорт всей схемы для больших наборов данных: здесь файлы дампа экспорта будут разбиты и сжаты. Параллелизм используется здесь *(Примечание. Добавление параллелизма увеличит загрузку ЦП на сервере)*
- Сведения о файле параметров [say exp.par]:

```
schemas=<schema>
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>_%U.dmp
logfile=exp_<dbname>_<schema>.log
compression = all
parallel=5
```

- Тип экспорта: **Экспорт таблицы** [Экспорт набора таблиц]
- Сведения о файле параметров [say exp.par]:

```
tables= tname1, tname2, tname3
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>.dmp
logfile=exp_<dbname>_<schema>.log
```

Шаг 9: Импорт команд

Необходимое условие:

- Перед импортом пользователя рекомендуется удалить импортированную схему или таблицу.

Команды:

```
impdp <bkpadmin>/<bkp123>  parfile=<imp.par>
```

* Пожалуйста, замените данные в <> соответствующими значениями в соответствии с вашей средой. Вы можете добавлять / изменять параметры в соответствии с вашими требованиями. В приведенном выше примере все остальные параметры добавляются в файлы параметров, как указано ниже: *

- Тип импорта: **импорт пользователя**
- Импортировать всю схему
- Сведения о файле параметров [say imp.par]:

```
schemas=<schema>
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>.dmp
logfile=imp_<dbname>_<schema>.log
```

- Тип импорта: **импорт пользователя для большой схемы**
- Импортировать всю схему для больших наборов данных: здесь используется параллелизм *(Примечание: добавление параллелизма увеличит загрузку ЦП на сервере)*
- Сведения о файле параметров [say imp.par]:

```
schemas=<schema>
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>_%U.dmp
logfile=imp_<dbname>_<schema>.log
parallel=5
```

- Тип импорта: **импорт таблицы** [Import set of tables]
- Сведения о файле параметров [say imp.par]:

```
tables= tname1, tname2, tname3
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>.dmp
logfile=exp_<dbname>_<schema>.log
TABLE_EXISTS_ACTION= <APPEND /SKIP /TRUNCATE /REPLACE>
```

---

1. Шаги Datapump

Исходный сервер [Экспорт данных]	Целевой сервер [Импорт данных]
1. Создайте папку datapump, которая будет содержать файлы дампа экспорта	4. Создайте папку datapump, которая будет содержать файлы дампа импорта
2. Войдите в схему базы данных, которая будет выполнять экспорт.	5. Войдите в схему базы данных, которая будет выполнять импорт.
3. Создайте каталог, указывающий на шаг 1.	6. Создайте каталог, указывающий на шаг 4.
7. Запустите Export Statement.	

Исходный сервер [Экспорт данных]	Целевой сервер [Импорт данных]
8. Скопируйте / SCP файлы дампа на целевой сервер.	
	9. Запустить операции импорта
	10. проверять данные, компилировать недопустимые объекты и предоставлять соответствующие гранты

Копирование таблиц между различными схемами и табличными пространствами

```
 expdp <bkpadmin>/<bkp123> directory=DATAPUMP_REMOTE_DIR dumpfile=<customer.dmp>


 impdp <bkpadmin>/<bkp123> directory=DATAPUMP_REMOTE_DIR dumpfile=<customer.dmp>
remap_schema=<source schema>:<target schema> remap_tablespace=<source tablespace>:<target
tablespace>
```

Прочитайте Насос данных онлайн: https://riptutorial.com/ru/oracle/topic/9391/насос-данных

глава 13: Обновление с помощью объединений

Вступление

Вопреки распространенному недоразумению (в том числе и на SO), Oracle позволяет получать обновления через соединения. Однако есть некоторые (довольно логичные) требования. Мы проиллюстрируем, что не работает, а что - на простом примере. Другим способом достижения этого является утверждение MERGE.

Examples

Примеры: что работает, а что нет

```
create table tgt ( id, val ) as
  select 1, 'a' from dual union all
  select 2, 'b' from dual
;

Table TGT created.

create table src ( id, val ) as
  select 1, 'x' from dual union all
  select 2, 'y' from dual
;

Table SRC created.

update
  ( select t.val as t_val, s.val as s_val
    from   tgt t inner join src s on t.id = s.id
  )
set t_val = s_val
;

SQL Error: ORA-01779: cannot modify a column which maps to a non key-preserved table
01779. 00000 -  "cannot modify a column which maps to a non key-preserved table"
*Cause:    An attempt was made to insert or update columns of a join view which
           map to a non-key-preserved table.
*Action:   Modify the underlying base tables directly.
```

Представьте, что произойдет, если бы мы имели значение `1` в столбце `src.id` более одного раза, с разными значениями для `src.val` . Очевидно, что обновление не имеет смысла (в ЛЮБОЙ базе данных - это логическая проблема). Теперь **мы** знаем, что дубликатов в `src.id` , но механизм Oracle не знает этого, поэтому он жалуется. Возможно, именно поэтому так много практикующих полагают, что у Oracle «нет UPDATE с объединениями»?

Oracle ожидает, что `src.id` должен быть уникальным и что он, Oracle, будет знать это

заранее. Легко фиксируется! Обратите внимание, что то же самое работает с составными ключами (более одного столбца), если для соответствия для обновления требуется использовать более одного столбца. На практике `src.id` может быть PK, а `tgt.id` может быть FK, указывающим на этот PK, но это не относится к обновлениям с соединением; значение имеет то ограничение уникальности.

```
alter table src add constraint src_uc unique (id);

Table SRC altered.

update
  ( select t.val as t_val, s.val as s_val
    from   tgt t inner join src s on t.id = s.id
  )
set t_val = s_val
;

2 rows updated.

select * from tgt;

ID  VAL
--  ---
 1  x
 2  y
```

Тот же результат может быть достигнут с помощью оператора MERGE (который заслуживает своей собственной статьи Документации), и я лично предпочитаю MERGE в этих случаях, но причина не в том, что «Oracle не делает обновлений с объединениями». Как показывает этот пример, Oracle *делает* обновления с объединениями.

Прочитайте Обновление с помощью объединений онлайн:
https://riptutorial.com/ru/oracle/topic/8061/обновление-с-помощью-объединений

глава 14: Обработка значений NULL

Вступление

Столбец имеет значение NULL, если он не имеет значения, независимо от типа данных этого столбца. Столбец никогда не должен сравниваться с NULL, используя этот синтаксис `a = NULL` как результат будет UNKNOWN. Вместо этого используйте условия `a IS NULL` или `a IS NOT NULL` . NULL не равен NULL. Чтобы сравнить два выражения, где может быть null, используйте одну из функций, описанных ниже. Все операторы, кроме конкатенации, возвращают NULL, если один из их операндов равен NULL. Например, результат `3 * NULL + 5` равен null.

замечания

NULL не может отображаться в столбцах, ограниченных ограничением PRIMARY KEY или NOT NULL. (Исключение - новое ограничение с предложением NOVALIDATE)

Examples

Столбцы любого типа данных могут содержать NULL

```
SELECT 1 NUM_COLUMN, 'foo' VARCHAR2_COLUMN from DUAL
UNION ALL
SELECT NULL, NULL from DUAL;
```

NUM_COLUMN	VARCHAR2_COLUMN
1	Foo
(ноль)	(ноль)

Пустые строки: NULL

```
SELECT 1 a, '' b from DUAL;
```

	B
1	(ноль)

Операции, содержащие NULL, являются NULL, за исключением конкатенации

```
SELECT 3 * NULL + 5, 'Hello ' || NULL || 'world'   from DUAL;
```

3 * NULL + 5	'ПРИВЕТ' || NULL || 'МИР'
(ноль)	Привет, мир

NVL заменит нулевое значение

```
SELECT a column_with_null, NVL(a, 'N/A') column_without_null FROM
  (SELECT NULL a FROM DUAL);
```

COLUMN_WITH_NULL	COLUMN_WITHOUT_NULL
(ноль)	N / A

NVL полезно сравнить два значения, которые могут содержать NULL:

```
SELECT
    CASE WHEN a = b THEN 1 WHEN a <> b THEN 0 else -1 END comparison_without_nvl,
    CASE WHEN NVL(a, -1) = NVL(b, -1) THEN 1 WHEN NVL(a, -1) <> NVL(b, -1) THEN 0 else -1 END
comparison_with_nvl
  FROM
    (select null a, 3 b FROM DUAL
     UNION ALL
     SELECT NULL, NULL FROM DUAL);
```

COMPARISON_WITHOUT_NVL	COMPARISON_WITH_NVL
-1	0
-1	1

NVL2 получить другой результат, если значение равно null или нет

Если первый параметр NOT NULL, NVL2 вернет второй параметр. В противном случае он вернет третий.

```
SELECT NVL2(null, 'Foo', 'Bar'), NVL2(5, 'Foo', 'Bar') FROM DUAL;
```

NVL2 (NULL, 'Foo', 'BAR')	NVL2 (5, 'Foo', 'BAR')
Бар	Foo

COALESCE, чтобы вернуть первое значение, отличное от NULL

```
SELECT COALESCE(a, b, c, d, 5) FROM
```

```
    (SELECT NULL A, NULL b, NULL c, 4 d FROM DUAL);
```

COALESCE (A, B, C, D, 5)
4

В некоторых случаях использование COALESCE с двумя параметрами может быть быстрее, чем использование NVL, когда второй параметр не является константой. NVL всегда будет оценивать оба параметра. COALESCE остановится при первом не-NULL-значении, с которым он сталкивается. Это означает, что если первое значение не является NULL, COALESCE будет быстрее.

Прочитайте Обработка значений NULL онлайн: https://riptutorial.com/ru/oracle/topic/8183/обработка-значений-null

глава 15: Ограничение строк, возвращаемых запросом (Pagination)

Examples

Получить первые N строк с предложением ограничения строки

Предложение `FETCH` было введено в Oracle 12c R1:

```
SELECT   val
FROM     mytable
ORDER BY val DESC
FETCH FIRST 5 ROWS ONLY;
```

Пример без FETCH, который также работает в более ранних версиях:

```
SELECT * FROM (
   SELECT   val
   FROM     mytable
   ORDER BY val DESC
) WHERE ROWNUM <= 5;
```

Разбиение на страницы в SQL

```
SELECT val
FROM   (SELECT val, rownum AS rnum
        FROM   (SELECT val
                FROM   rownum_order_test
                ORDER BY val)
        WHERE rownum <= :upper_limit)
WHERE  rnum >= :lower_limit ;
```

таким образом, мы можем разбивать на таблицы данные таблицы, как и на веб-странице

Получить N номеров записей из таблицы

Мы можем ограничить количество строк из результата, используя предложение rownum

```
select * from
(
  select val from  mytable
) where rownum<=5
```

Если нам нужна первая или последняя запись, то мы хотим, чтобы во внутреннем запросе выполнялся оператор order by, который даст результат по заказу.

Последние пять записей:

```
select * from
(
    select val from  mytable order by val desc
) where rownum<=5
```

Первые пять записей

```
select * from
(
    select val from  mytable order by val
) where rownum<=5
```

Получить строку N через M из многих строк (до Oracle 12c)

Используйте аналитическую функцию row_number ():

```
with t as (
  select col1
  , col2
  , row_number() over (order by col1, col2) rn
  from table
)
select col1
, col2
from t
where rn between N and M; -- N and M are both inclusive
```

Oracle 12c легче справляется с `OFFSET` и `FETCH` .

Пропуск некоторых строк, а затем некоторые

В Oracle 12g +

```
SELECT Id, Col1
FROM TableName
ORDER BY Id
OFFSET 20 ROWS FETCH NEXT 20 ROWS ONLY;
```

В предыдущих версиях

```
SELECT Id,
   Col1
 FROM (SELECT Id,
           Col1,
           row_number() over (order by Id) RowNumber
      FROM TableName)
WHERE RowNumber BETWEEN 21 AND 40
```

Пропуск ряда строк из результата

В Oracle 12g +

```
SELECT Id, Col1
FROM TableName
ORDER BY Id
OFFSET 5 ROWS;
```

В предыдущих версиях

```
SELECT Id,
   Col1
 FROM (SELECT Id,
           Col1,
           row_number() over (order by Id) RowNumber
      FROM TableName)
WHERE RowNumber > 20
```

Прочитайте Ограничение строк, возвращаемых запросом (Pagination) онлайн: https://riptutorial.com/ru/oracle/topic/4300/ограничение-строк--возвращаемых-запросом--pagination-

глава 16: ограничения

Examples

Обновление внешних ключей с новым значением в Oracle

Предположим, что у вас есть таблица, и вы хотите изменить один из этих первичных идентификаторов этой таблицы. вы можете использовать следующий scrpit. первичный идентификатор здесь - «PK_S»

```
begin
  for i in (select a.table_name, c.column_name
              from user_constraints a, user_cons_columns c
             where a.CONSTRAINT_TYPE = 'R'
               and a.R_CONSTRAINT_NAME = 'PK_S'
               and c.constraint_name = a.constraint_name) loop


         execute immediate 'update ' || i.table_name || ' set ' || i.column_name ||
                      '=to_number(''1000'' || ' ||  i.column_name || ') ';



  end loop;

end;
```

Отключить все связанные внешние ключи в oracle

Предположим, что у вас есть таблица T1, и она имеет отношение ко многим таблицам, а ее имя ограничения основного ключа - «pk_t1», которое вы хотите отключить с помощью этих внешних ключей, которые вы можете использовать:

```
Begin
    For I in (select table_name, constraint_name from user_constraint t where
r_constraint_name='pk_t1') loop

Execute immediate ' alter table ' || I.table_name || ' disable constraint ' ||
i.constraint_name;

   End loop;
End;
```

Прочитайте ограничения онлайн: https://riptutorial.com/ru/oracle/topic/6040/ограничения

глава 17: Последовательности

Синтаксис

- CREATE SEQUENCE SCHEMA.SEQUENCE {INCREMENT by INTEGER | НАЧАЛО С INTEGER | MAXVALUE INTEGER | NOMAXVALUE INTEGER | MINVALUE INTEGER | NOMINVALUE INTEGER | CYCLE INTEGER | NOCYCLE INTEGER | CACHE | NOCACHE | ЗАКАЗАТЬ | NOODER}

параметры

параметр	подробности
схема	название схемы
приращение	интервал между числами
начните с	необходимо первое число
MAXVALUE	Максимальное значение для последовательности
NOMAXVALUE	Максимальное значение по умолчанию
MinValue	минимальное значение для последовательности
nominvalue	минимальное значение по умолчанию
цикл	Сброс до начала после достижения этого значения
NOCYCLE	По умолчанию
кэш	Предел превалирования
NoCache	По умолчанию
порядок	Гарантировать порядок номеров
Noorder	дефолт

Examples

Создание последовательности: пример

Цель

Используйте инструкцию CREATE SEQUENCE для создания последовательности, которая является объектом базы данных, из которой несколько пользователей могут генерировать уникальные целые числа. Вы можете использовать последовательности, чтобы автоматически генерировать значения первичного ключа.

Когда генерируется порядковый номер, последовательность увеличивается, независимо от транзакции, совершающей или откат. Если два пользователя одновременно увеличивают одну и ту же последовательность, то порядковые номера, которые каждый пользователь получает, могут иметь пробелы, так как порядковые номера генерируются другим пользователем. Один пользователь никогда не сможет получить порядковый номер, сгенерированный другим пользователем. После того как значение последовательности генерируется одним пользователем, этот пользователь может продолжить доступ к этому значению независимо от того, увеличивается ли эта последовательность другим пользователем.

Номера последовательностей генерируются независимо от таблиц, поэтому одна и та же последовательность может использоваться для одного или нескольких таблиц. Возможно, что отдельные порядковые номера будут пропущены, потому что они были сгенерированы и использованы в транзакции, которая в конечном счете откат. Кроме того, один пользователь может не понимать, что другие пользователи рисуют из одной и той же последовательности.

После создания последовательности вы можете получить доступ к своим значениям в операторах SQL с псевдоколонкой CURRVAL, которая возвращает текущее значение последовательности или псевдоколонку NEXTVAL, которая увеличивает последовательность и возвращает новое значение.

Предпосылки

Чтобы создать последовательность в вашей собственной схеме, вы должны иметь системную привилегию CREATE SEQUENCE.

Чтобы создать последовательность в схеме другого пользователя, вы должны иметь системную привилегию CREATE ANY SEQUENCE.

Создание последовательности: пример Следующий оператор создает последовательность customers_seq в схеме экземпляра oe. Эта последовательность может использоваться для предоставления идентификационных номеров клиентов, когда строки добавляются в таблицу клиентов.

```
CREATE SEQUENCE customers_seq
START WITH     1000
INCREMENT BY   1
NOCACHE
NOCYCLE;
```

Первая ссылка на customer_seq.nextval возвращает 1000. Вторая возвращает 1001. Каждая последующая ссылка вернет значение 1 больше, чем предыдущая ссылка.

Прочитайте Последовательности онлайн: https://riptutorial.com/ru/oracle/topic/3709/последовательности

глава 18: Работа с датами

Examples

Арифметика даты

Oracle поддерживает `DATE` (включая время до ближайшей секунды) и `TIMESTAMP` (включая время до долей секунды) типов данных, которые позволяют арифметически (сложение и вычитание) изначально. Например:

Чтобы получить следующий день:

```
select to_char(sysdate + 1, 'YYYY-MM-DD') as tomorrow from dual;
```

Чтобы получить предыдущий день:

```
select to_char(sysdate - 1, 'YYYY-MM-DD') as yesterday from dual;
```

Чтобы добавить 5 дней к текущей дате:

```
select to_char(sysdate + 5, 'YYYY-MM-DD') as five_days_from_now from dual;
```

Чтобы добавить 5 часов к текущей дате:

```
select to_char(sysdate + (5/24), 'YYYY-MM-DD HH24:MI:SS') as five_hours_from_now from dual;
```

Чтобы добавить 10 минут к текущей дате:

```
select to_char(sysdate + (10/1440), 'YYYY-MM-DD HH24:MI:SS') as ten_mintues_from_now from
dual;
```

Чтобы добавить 7 секунд к текущей дате:

```
select to_char(sysdate + (7/86400), 'YYYY-MM-DD HH24:MI:SS') as seven_seconds_from_now from
dual;
```

Чтобы выбрать строки, где `hire_date` - 30 дней назад или более:

```
select * from emp where hire_date < sysdate - 30;
```

Чтобы выбрать строки, где `last_updated` столбец находится в последний час:

```
select * from logfile where last_updated >= sysdate - (1/24);
```

Oracle также предоставляет встроенный тип данных `INTERVAL` который представляет собой продолжительность времени (например, 1,5 дня, 36 часов, 2 месяца и т. Д.). Они также могут использоваться с арифметикой с выражениями `DATE` и `TIMESTAMP` . Например:

```
select * from logfile where last_updated >= sysdate - interval '1' hour;
```

Функция Add_months

Синтаксис: `add_months(p_date, integer) return date;`

Функция Add_months добавляет несколько месяцев к дате p_date.

```
SELECT add_months(date'2015-01-12', 2) m FROM dual;
```

M
2015-03-12

Вы также можете вычитать месяцы, используя отрицательный `amt`

```
SELECT add_months(date'2015-01-12', -2) m FROM dual;
```

M
2014-11-12

Когда расчетный месяц имеет меньше дней в качестве заданной даты, будет возвращен последний день расчетного месяца.

```
SELECT to_char( add_months(date'2015-01-31', 1),'YYYY-MM-DD') m FROM dual;
```

M
2015-02-28

Прочитайте Работа с датами онлайн: https://riptutorial.com/ru/oracle/topic/768/работа-с-датами

глава 19: Разделение разделительных строк

Examples

Разделение строк с использованием пункта рекурсивного подзапроса факторинга

Пример данных :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Запрос :

```
WITH bounds ( id, list, start_pos, end_pos, lvl ) AS (
  SELECT id, list, 1, INSTR( list, ',' ), 1 FROM table_name
UNION ALL
  SELECT id,
         list,
         end_pos + 1,
         INSTR( list, ',', end_pos + 1 ),
         lvl + 1
  FROM   bounds
  WHERE  end_pos > 0
)
SELECT id,
       SUBSTR(
         list,
         start_pos,
         CASE end_pos
           WHEN 0
           THEN LENGTH( list ) + 1
           ELSE end_pos
         END - start_pos
       ) AS item,
      lvl
FROM   bounds
ORDER BY id, lvl;
```

Выход :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                 1
         1 b                 2
         1 c                 3
```

```
         1 d               4
         2 e               1
         3 (NULL)          1
         4 f               1
         4 (NULL)          2
         4 g               3
```

Разделение строк с использованием функции PL / SQL

PL / SQL Функция :

```
CREATE OR REPLACE FUNCTION split_String(
  i_str    IN  VARCHAR2,
  i_delim  IN  VARCHAR2 DEFAULT ','
) RETURN SYS.ODCIVARCHAR2LIST DETERMINISTIC
AS
  p_result       SYS.ODCIVARCHAR2LIST := SYS.ODCIVARCHAR2LIST();
  p_start        NUMBER(5) := 1;
  p_end          NUMBER(5);
  c_len CONSTANT NUMBER(5) := LENGTH( i_str );
  c_ld  CONSTANT NUMBER(5) := LENGTH( i_delim );
BEGIN
  IF c_len > 0 THEN
    p_end := INSTR( i_str, i_delim, p_start );
    WHILE p_end > 0 LOOP
      p_result.EXTEND;
      p_result( p_result.COUNT ) := SUBSTR( i_str, p_start, p_end - p_start );
      p_start := p_end + c_ld;
      p_end := INSTR( i_str, i_delim, p_start );
    END LOOP;
    IF p_start <= c_len + 1 THEN
      p_result.EXTEND;
      p_result( p_result.COUNT ) := SUBSTR( i_str, p_start, c_len - p_start + 1 );
    END IF;
  END IF;
  RETURN p_result;
END;
/
```

Пример данных :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Запрос :

```
SELECT t.id,
       v.column_value AS value,
       ROW_NUMBER() OVER ( PARTITION BY id ORDER BY ROWNUM ) AS lvl
FROM   table_name t,
       TABLE( split_String( t.list ) ) (+) v
```

Выход :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
         4 (NULL)           2
         4 g                3
```

Разделение строк с использованием коррелированного выражения таблицы

Пример данных :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Запрос :

```
SELECT t.id,
       v.COLUMN_VALUE AS value,
       ROW_NUMBER() OVER ( PARTITION BY id ORDER BY ROWNUM ) AS lvl
FROM   table_name t,
       TABLE(
         CAST(
           MULTISET(
             SELECT REGEXP_SUBSTR( t.list, '([^,]*)(,|$)', 1, LEVEL, NULL, 1 )
             FROM   DUAL
             CONNECT BY LEVEL < REGEXP_COUNT( t.list, '[^,]*(,|$)' )
           )
           AS SYS.ODCIVARCHAR2LIST
         )
       ) v;
```

Выход :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
         4 (NULL)           2
         4 g                3
```

Разбиение строк с использованием иерархического запроса

Пример данных :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Запрос :

```
SELECT t.id,
       REGEXP_SUBSTR( list, '([^,]*)(,|$)', 1, LEVEL, NULL, 1 ) AS value,
       LEVEL AS lvl
FROM   table_name t
CONNECT BY
       id = PRIOR id
AND    PRIOR SYS_GUID() IS NOT NULL
AND    LEVEL < REGEXP_COUNT( list, '([^,]*)(,|$)' )
```

Выход :

```
        ID ITEM            LVL
---------- ------- ----------
         1 a                 1
         1 b                 2
         1 c                 3
         1 d                 4
         2 e                 1
         3 (NULL)            1
         4 f                 1
         4 (NULL)            2
         4 g                 3
```

Разбиение строк с использованием выражений XMLTable и FLWOR

Это решение использует функцию `ora:tokenize` XQuery , доступную из Oracle 11.

Пример данных :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Запрос :

```
SELECT t.id,
       x.item,
       x.lvl
```

```
FROM   table_name t,
       XMLTABLE(
         'let $list := ora:tokenize(.,","),
              $cnt := count($list)
          for $val at $r in $list
          where $r < $cnt
          return $val'
       PASSING list||','
       COLUMNS
         item VARCHAR2(100) PATH '.',
         lvl FOR ORDINALITY
       ) (+) x;
```

Выход :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                 1
         1 b                 2
         1 c                 3
         1 d                 4
         2 e                 1
         3 (NULL)       (NULL)
         4 f                 1
         4 (NULL)            2
         4 g                 3
```

Разделение строк с использованием CROSS APPLY (Oracle 12c)

Пример данных :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Запрос :

```
SELECT t.id,
       REGEXP_SUBSTR( t.list, '([^,]*)($|,)', 1, l.lvl, NULL, 1 ) AS item,
       l.lvl
FROM   table_name t
       CROSS APPLY
       (
         SELECT LEVEL AS lvl
         FROM   DUAL
         CONNECT BY LEVEL <= REGEXP_COUNT( t.list, ',' ) + 1
       ) l;
```

Выход :

```
        ID ITEM           LVL
---------- ------- ----------
```

```
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
         4 (NULL)           2
         4 g                3
```

Разделение разделительных строк с использованием XMLTable

Пример данных :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Запрос :

```
SELECT t.id,
       SUBSTR( x.item.getStringVal(), 2 ) AS item,
       x.lvl
FROM   table_name t
       CROSS JOIN
       XMLTABLE(
         ( '"#' || REPLACE( t.list, ',', '","#' ) || '"' )
         COLUMNS item XMLTYPE PATH '.',
                 lvl  FOR ORDINALITY
       ) x;
```

(Примечание: символ `#` добавляется для облегчения извлечения значений `NULL` , а затем удаляется с помощью `SUBSTR( item, 2 )` . Если значения `NULL` не требуются, вы можете упростить запрос и опустить это.)

Выход :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
         4 (NULL)           2
         4 g                3
```

Прочитайте Разделение разделительных строк онлайн:
https://riptutorial.com/ru/oracle/topic/1968/разделение-разделительных-строк

глава 20: Разделение таблиц

Вступление

Разделение - это функция разделения таблиц и индексов на более мелкие части. Он используется для повышения производительности и управления меньшими частями по отдельности. Ключ раздела - это столбец или набор столбцов, определяющий, в каком разделе будет храниться каждая строка. Обзор разделов в официальной документации Oracle

замечания

Разделение является дополнительным вариантом и доступно только для Enterprise Edition.

Examples

Разделение хэшей

Это создает таблицу, разбитую на хэш, в этом примере на идентификаторе хранилища.

```
CREATE TABLE orders (
    order_nr NUMBER(15),
    user_id VARCHAR2(2),
    order_value NUMBER(15),
    store_id NUMBER(5)
)
PARTITION BY HASH(store_id) PARTITIONS 8;
```

Вы должны использовать мощность 2 для количества хэш-разделов, чтобы получить равномерное распределение в размере раздела.

Разбиение диапазона

Это создает таблицу, разбитую по диапазонам, в этом примере на значения порядка.

```
CREATE TABLE orders (
    order_nr NUMBER(15),
    user_id VARCHAR2(2),
    order_value NUMBER(15),
    store_id NUMBER(5)
)
PARTITION BY RANGE(order_value) (
    PARTITION p1 VALUES LESS THAN(10),
    PARTITION p2 VALUES LESS THAN(40),
    PARTITION p3 VALUES LESS THAN(100),
    PARTITION p4 VALUES LESS THAN(MAXVALUE)
);
```

Выберите существующие разделы

Проверка существующих разделов на схеме

```
SELECT * FROM user_tab_partitions;
```

Перечисление списков

Это создает таблицу, разбитую по спискам, в этом примере на идентификатор магазина.

```
CREATE TABLE orders (
    order_nr NUMBER(15),
    user_id VARCHAR2(2),
    order_value NUMBER(15),
    store_id NUMBER(5)
)
PARTITION BY LIST(store_id) (
    PARTITION p1 VALUES (1,2,3),
    PARTITION p2 VALUES(4,5,6),
    PARTITION p3 VALUES(7,8,9),
    PARTITION p4 VALUES(10,11)
);
```

Падение раздела

```
ALTER TABLE table_name DROP PARTITION partition_name;
```

Выбор данных из раздела

Выбор данных из раздела

```
SELECT * FROM orders PARTITION(partition_name);
```

Усечение раздела

```
ALTER TABLE table_name TRUNCATE PARTITION partition_name;
```

Переименовать раздел

```
ALTER TABLE table_name RENAME PARTITION p3 TO p6;
```

Переместить раздел в другое табличное пространство

```
ALTER TABLE table_name
MOVE PARTITION partition_name TABLESPACE tablespace_name;
```

Добавить новый раздел

```
ALTER TABLE table_name
ADD PARTITION new_partition VALUES LESS THAN(400);
```

Разделить раздел

Разделяет некоторый раздел на два раздела с другой высокой границей.

```
ALTER TABLE table_name SPLIT PARTITION old_partition
    AT (new_high_bound) INTO (PARTITION new_partition TABLESPACE new_tablespace,
    PARTITION old_partition)
```

Объединить разделы

Объединить два раздела в один

```
ALTER TABLE table_name
  MERGE PARTITIONS first_partition, second_partition
  INTO  PARTITION  splitted_partition TABLESPACE new_tablespace
```

Обмен разделами

Exchange / конвертировать раздел в несегментированную таблицу и наоборот. Это облегчает быстрое «перемещение» данных между сегментами данных (в отличие от «insert ... select» или «create table ... as select»), поскольку операция DDL (операция обмена разделами является данными обновление словаря без перемещения фактических данных), а не DML (большие отладки / повторные издержки).

Большинство основных примеров:

1. Преобразовать несегментированную таблицу (таблицу «B») в раздел (таблицы «A»):

Таблица «A» не содержит данных в разделе «OLD_VALUES», а таблица «B» содержит данные

```
ALTER TABLE "A" EXCHANGE PARTITION "OLD_VALUES" WITH TABLE "B";
```

Результат: данные «перемещаются» из таблицы «B» (не содержат данных после операции) для разделения «OLD_VALUES»

2. Преобразовать раздел в несегментированную таблицу:

Таблица «A» содержит данные в разделе «OLD_VALUES», а таблица «B» не содержит данных

```
ALTER TABLE "A" EXCHANGE PARTITION "OLD_VALUES" WITH TABLE "B";
```

Результат: данные «перемещаются» из раздела «OLD_VALUES» (не содержит данных после операции) в таблицу «B»

Примечание. Существует немало дополнительных опций, функций и ограничений для этой операции.

Дополнительную информацию можно найти по этой ссылке ---> « https://docs.oracle.com/cd/E11882_01/server.112/e25523/part_admin002.htm#i1107555 » (раздел «Обмен разделами»)

Прочитайте Разделение таблиц онлайн: https://riptutorial.com/ru/oracle/topic/3955/разделение-таблиц

глава 21: Различные способы обновления записей

Синтаксис

- UPDATE table-Name [[AS] corre-Name] SET column-Name = Value [, column-Name = Value}] * [Предложение WHERE]
- UPDATE table-Name SET column-Name = Value [, column-Name = Value] * WHERE CURRENT OF

Examples

Обновить синтаксис с примером

Нормальное обновление

```
UPDATE
      TESTTABLE
SET
      TEST_COLUMN= 'Testvalue',TEST_COLUMN2= 123
WHERE
      EXISTS
          (SELECT MASTERTABLE.TESTTABLE_ID
           FROM MASTERTABLE
           WHERE ID_NUMBER=11);
```

Обновление с помощью встроенного представления

Использование Inline View (если оно считается обновляемым Oracle)

Примечание . Если вы столкнулись с нефиксированной сохраненной ошибкой строки, добавьте индекс, чтобы решить эту проблему, чтобы сделать его обновляемым

```
UPDATE
      (SELECT
            TESTTABLE.TEST_COLUMN AS OLD,
            'Testvalue' AS NEW
       FROM
                TESTTABLE
            INNER JOIN
                MASTERTABLE
            ON TESTTABLE.TESTTABLE_ID = MASTERTABLE.TESTTABLE_ID
       WHERE ID_NUMBER=11) T
SET
      T.OLD        = T.NEW;
```

Обновление с использованием Merge

Использование слияния

```
MERGE INTO
      TESTTABLE
USING
      (SELECT
            T1.ROWID AS RID,
            T2.TESTTABLE_ID
       FROM
                TESTTABLE T1
            INNER JOIN
                MASTERTABLE T2
            ON TESTTABLE.TESTTABLE_ID = MASTERTABLE.TESTTABLE_ID
       WHERE ID_NUMBER=11)
ON
      ( ROWID = RID )
WHEN MATCHED
THEN
    UPDATE SET TEST_COLUMN= 'Testvalue';
```

Слияние с данными образца

```
drop table table01;
drop table table02;

create table table01 (
       code int,
       name varchar(50),
       old int
);

create table table02 (
       code int,
       name varchar(50),
       old int
);

truncate table table01;
insert into table01 values (1, 'A', 10);
insert into table01 values (9, 'B', 12);
insert into table01 values (3, 'C', 14);
insert into table01 values (4, 'D', 16);
insert into table01 values (5, 'E', 18);

truncate table table02;
insert into table02 values (1, 'AA', null);
insert into table02 values (2, 'BB', 123);
insert into table02 values (3, 'CC', null);
insert into table02 values (4, 'DD', null);
insert into table02 values (5, 'EE', null);

select * from table01 a order by 2;
select * from table02 a order by 2;

--
```

```
merge into table02 a using (
      select b.code, b.old from table01 b
) c on (
  a.code = c.code
)
when matched then update set a.old = c.old
;

--

select a.*, b.* from table01 a
inner join table02 b on a.code = b.codetable01;

select * from table01 a
where
       exists
       (
         select 'x' from table02 b where a.code = b.codetable01
       );

select * from table01 a where a.code in (select b.codetable01 from table02 b);

--

select * from table01 a
where
       not exists
       (
         select 'x' from table02 b where a.code = b.codetable01
       );

select * from table01 a where a.code not in (select b.codetable01 from table02 b);
```

Прочитайте Различные способы обновления записей онлайн:
https://riptutorial.com/ru/oracle/topic/4193/различные-способы-обновления-записей

глава 22: Разметка ключевых слов или специальных символов

Examples

Отметьте имя таблицы или столбца специальными символами

Выберите * from firm_s_address;

Выберите * из "firm_s_address";

Разграничение имени таблицы или столбца, которое также является зарезервированным словом

Скажем, у вас есть таблица с именем table или вы хотите создать таблицу с именем, которое также является ключевым словом. Вы должны включить таблицу имен в пару двойных кавычек «таблица»,

Выберите * из таблицы; Выше запроса будет сбой с синтаксической ошибкой, где, как показано ниже, запрос будет работать нормально.

Выберите * из «таблицы»;

Прочитайте Разметка ключевых слов или специальных символов онлайн: https://riptutorial.com/ru/oracle/topic/6553/разметка-ключевых-слов-или-специальных-символов

глава 23: Реальная безопасность приложений

Вступление

Oracle Real Application Security была внедрена в Oracle 12c. В нем суммируются многие темы безопасности, такие как User-Role-Model, Access Control, Application vs. Database, End-User-Security или Row- и Column Level Security

Examples

заявка

Чтобы связать приложение с чем-то в базе данных, есть три основные части:

Привилегия приложения: Привилегия приложения описывает привилегии типа `SELECT` , `INSERT` , `UPDATE` , `DELETE` , ... Привилегии приложений можно суммировать как сводную привилегию.

```
XS$PRIVILEGE(
    name=>'privilege_name'
    [, implied_priv_list=>XS$NAME_LIST('"SELECT"', '"INSERT"', '"UPDATE"', '"DELETE"')]
)

XS$PRIVILEGE_LIST(
    XS$PRIVILEGE(...),
    XS$PRIVILEGE(...),
    ...
);
```

Пользователь приложения:

Простой пользователь приложения:

```
BEGIN
    SYS.XS_PRINCIPAL.CREATE_USER('user_name');
END;
```

Пользователь прямого входа:

```
BEGIN
    SYS.XS_PRINCIPAL.CREATE_USER(name => 'user_name', schema => 'schema_name');
END;

BEGIN
    SYS.XS_PRINCIPAL.SET_PASSWORD('user_name', 'password');
```

```
END;
CREATE PROFILE prof LIMIT
    PASSWORD_REUSE_TIME 1/4440
    PASSWORD_REUSE_MAX 3
    PASSWORD_VERIFY_FUNCTION Verify_Pass;

BEGIN
    SYS.XS_PRINCIPAL.SET_PROFILE('user_name', 'prof');
END;

BEGIN
    SYS.XS_PRINCIPAL.GRANT_ROLES('user_name', 'XSONNCENT');
END;
```

(необязательный:)

```
BEGIN
    SYS.XS_PRINCIPAL.SET_VERIFIER('user_name', '6DFF060084ECE67F', XS_PRINCIPAL.XS_SHA512");
END;
```

Роль приложения:

Регулярная роль приложения:

```
DECLARE
    st_date TIMESTAMP WITH TIME ZONE;
    ed_date TIMESTAMP WITH TIME ZONE;
BEGIN
    st_date := SYSTIMESTAMP;
    ed_date := TO_TIMESTAMP_TZ('2013-06-18 11:00:00 -5:00','YYYY-MM-DD HH:MI:SS');
    SYS.XS_PRINCIPAL.CREATE_ROLE
        (name => 'app_regular_role',
        enabled => TRUE,
        start_date => st_date,
        end_date => ed_date);
END;
```

Динамическая роль приложения: (активируется динамически на основе состояния аутентификации)

```
BEGIN
    SYS.XS_PRINCIPAL.CREATE_DYNAMIC_ROLE
        (name => 'app_dynamic_role',
        duration => 40,
        scope => XS_PRINCIPAL.SESSION_SCOPE);
END;
```

Предопределенные роли приложения:

Regular:

- `XSPUBLIC`
- `XSBYPASS`
- `XSSESSIONADMIN`
- `XSNAMESPACEADMIN`

- `XSPROVISIONER`
- `XSCACHEADMIN`
- `XSDISPATCHER`

Динамический: (зависит от состояния аутентификации пользователя приложения)

- `DBMS_AUTH` : (`DBMS_AUTH` прямого входа или другого метода проверки подлинности базы данных)
- `EXTERNAL_DBMS_AUTH` : (метод прямой регистрации или другой проверки подлинности базы данных и внешний пользователь)
- `DBMS_PASSWD` : (прямой вход с паролем)
- `MIDTIER_AUTH` : (аутентификация через приложение среднего уровня)
- `XSAUTHENTICATED` : (приложение прямого или среднего уровня)
- `XSSWITCH` : (пользователь переключился с прокси-пользователя на пользователя приложения)

Прочитайте Реальная безопасность приложений онлайн:
https://riptutorial.com/ru/oracle/topic/10864/реальная-безопасность-приложений

глава 24: Регистрация ошибок

Examples

Регистрация ошибок при записи в базу данных

Создать таблицу журналов ошибок Oracle ERR $ _EXAMPLE для существующей таблицы EXAMPLE:

```
EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG('EXAMPLE', NULL, NULL, NULL, TRUE);
```

Сделайте операцию записи с SQL:

```
insert into EXAMPLE (COL1) values ('example')
LOG ERRORS INTO ERR$_EXAMPLE reject limit unlimited;
```

Прочитайте Регистрация ошибок онлайн: https://riptutorial.com/ru/oracle/topic/3505/регистрация-ошибок

глава 25: Рекурсивный факторинг подзапросов с использованием предложения WITH (общие выражения таблицы AKA)

замечания

Рекурсивный факторинг подзапросов доступен в Oracle 11g R2.

Examples

Простой целочисленный генератор

Запрос :

```
WITH generator ( value ) AS (
  SELECT 1 FROM DUAL
UNION ALL
  SELECT value + 1
  FROM   generator
  WHERE  value < 10
)
SELECT value
FROM   generator;
```

Выход :

```
VALUE
-----
    1
    2
    3
    4
    5
    6
    7
    8
    9
   10
```

Разделение разделительной строки

Пример данных :

```
CREATE TABLE table_name ( value VARCHAR2(50) );

INSERT INTO table_name ( value ) VALUES ( 'A,B,C,D,E' );
```

Запрос :

```
WITH items ( list, item, lvl ) AS (
  SELECT value,
         REGEXP_SUBSTR( value, '[^,]+', 1, 1 ),
         1
  FROM   table_name
UNION ALL
  SELECT value,
         REGEXP_SUBSTR( value, '[^,]+', 1, lvl + 1 ),
         lvl + 1
  FROM   items
  WHERE  lvl < REGEXP_COUNT( value, '[^,]+' )
)
SELECT * FROM items;
```

Выход :

```
LIST      ITEM LVL
--------- ---- ---
A,B,C,D,E    A   1
A,B,C,D,E    B   2
A,B,C,D,E    C   3
A,B,C,D,E    D   4
A,B,C,D,E    E   5
```

Прочитайте Рекурсивный факторинг подзапросов с использованием предложения WITH (общие выражения таблицы AKA) онлайн: https://riptutorial.com/ru/oracle/topic/3506/рекурсивный-факторинг-подзапросов-с-использованием-предложения-with--общие-выражения-таблицы-aka-

глава 26: Словарь данных

замечания

Представления словаря данных, также известные как представления каталога, позволяют отслеживать состояние базы данных в режиме реального времени:

Представления, предваряемые `USER_` , `ALL_` и `DBA_` , отображают информацию об объектах схемы, которые принадлежат вам (`USER_`), доступные вам (`ALL_`) или доступны пользователю с привилегией SYSDBA (`DBA_`). Например, в представлении `ALL_TABLES` отображаются все таблицы, на которые у вас есть привилегии.

Представления `V$` показывают информацию, связанную с производительностью.

Представления `_PRIVS` показывают информацию о привилегиях для разных комбинаций пользователей, ролей и объектов.

Документация Oracle: Представления каталога / Словарь данных

Examples

Текстовый источник сохраненных объектов

`USER_SOURCE` описывает текстовый источник хранимых объектов, принадлежащих текущему пользователю. В этом представлении не отображается столбец `OWNER` .

```
select * from user_source where type='TRIGGER' and lower(text) like '%order%'
```

`ALL_SOURCE` описывает текстовый источник хранимых объектов, доступных для текущего пользователя.

```
select * from all_source where owner=:owner
```

`DBA_SOURCE` описывает текстовый источник всех сохраненных объектов в базе данных.

```
select * from dba_source
```

Получить список всех таблиц в Oracle

```
select owner, table_name
from all_tables
```

`ALL_TAB_COLUMNS` описывает столбцы таблиц, представлений и кластеров, доступных для

текущего пользователя. `COLS` является синонимом `USER_TAB_COLUMNS` .

```
select *
from all_tab_columns
where table_name = :tname
```

Информация о привилегиях

Все роли, предоставленные пользователю.

```
select *
from dba_role_privs
where grantee= :username
```

Привилегии, предоставленные пользователю:

1. системные привилегии

```
select *
from dba_sys_privs
where grantee = :username
```

2. объектные гранты

```
select *
from dba_tab_privs
where grantee = :username
```

Разрешения на роли.

Роли, предоставленные другим ролям.

```
select *
from role_role_privs
where role in (select granted_role from dba_role_privs where grantee= :username)
```

1. системные привилегии

```
select *
from role_sys_privs
where role in (select granted_role from dba_role_privs where grantee= :username)
```

2. объектные гранты

```
select *
from role_tab_privs
where role in (select granted_role from dba_role_privs where grantee= :username)
```

Версия Oracle

```
select *
from v$version
```

Описывает все объекты в базе данных.

```
select *
from dba_objects
```

Чтобы просмотреть все виды словаря данных, к которым у вас есть доступ

```
select * from dict
```

Прочитайте Словарь данных онлайн: https://riptutorial.com/ru/oracle/topic/7347/словарь-данных

глава 27: Советы

параметры

параметры	подробности
Степень параллелизма (DOP)	Это количество параллельных подключений / процессов, которые вы хотите, чтобы ваш запрос открывался. Обычно это 2, 4, 8, 16 и так далее.
Название таблицы	Имя таблицы, на которой будет применяться параллельная подсказка.

Examples

Параллельный намек

Параллельные подсказки на уровне инструкций являются самыми легкими:

```
SELECT /*+ PARALLEL(8) */ first_name, last_name FROM employee emp;
```

Параллельные подсказки на уровне объекта дают больше контроля, но более подвержены ошибкам; разработчики часто забывают использовать псевдоним вместо имени объекта или забывают включать некоторые объекты.

```
SELECT /*+ PARALLEL(emp,8) */ first_name, last_name FROM employee emp;
```

```
SELECT /*+ PARALLEL(table_alias,Degree of Parallelism) */ FROM table_name table_alias;
```

Предположим, что для выполнения запроса требуется 100 секунд без использования параллельного намека. Если мы изменим DOP на 2 для одного запроса, тогда в *идеале один и* тот же запрос с параллельным намеком займет 50 секунд. Аналогично, используя DOP как 4, потребуется 25 секунд.

На практике параллельное выполнение зависит от многих других факторов и не масштабируется линейно. Это особенно справедливо для небольших периодов времени, когда параллельные накладные расходы могут быть больше, чем выгоды от работы на нескольких параллельных серверах.

USE_NL

Используйте вложенные петли.

Использование: `use_nl(AB)`

Этот подсказку попросит движок использовать метод вложенных циклов, чтобы присоединиться к таблицам A и B. Это сравнение по ряду строк. Подсказка не форсирует порядок соединения, просто запрашивает NL.

```
SELECT /*+use_nl(e d)*/ *
FROM Employees E
JOIN Departments D on E.DepartmentID = D.ID
```

ПРИЛОЖЕНИЕ СОВЕТ

Msgstr "Использовать метод DIRECT PATH для вставки новых строк".

В подсказке `APPEND` указывается, что двигатель использует нагрузку прямого пути . Это означает, что двигатель не будет использовать обычную вставку с использованием структур памяти и стандартных блокировок, но будет записывать непосредственно в табличное пространство данные. Всегда создает новые блоки, которые добавляются к сегменту таблицы. Это будет быстрее, но есть некоторые ограничения:

- Вы не можете прочитать из таблицы, которую вы добавили в тот же сеанс, пока вы не объявите или не отмените транзакцию.
- Если на таблице определены триггеры, Oracle не будет использовать прямой путь (это другая история для загрузок sqlldr).
- другие

Пример.

```
INSERT /*+append*/ INTO Employees
SELECT *
FROM Employees;
```

USE_HASH

Поручает движку использовать хэш-метод для объединения таблиц в аргумент.

Использование: `use_hash(TableA [TableB] ... [TableN])`

Как объяснялось во многих местах , «в HASH-соединении Oracle обращается к одной таблице (обычно к меньшим из объединенных результатов) и создает хэш-таблицу в ключе соединения в памяти. Затем она сканирует другую таблицу в соединении (обычно больше один) и проверяет хэш-таблицу для совпадений с ней ».

Это предпочтительнее, чем метод вложенных циклов, когда таблицы большие, индексов нет и т. Д.

Примечание : подсказка не форсирует порядок соединения, просто запрашивает метод

HASH JOIN.

Пример использования:

```
SELECT /*+use_hash(e d)*/ *
FROM Employees E
JOIN Departments D on E.DepartmentID = D.ID
```

ПОЛНЫЙ

FULL hint сообщает Oracle о выполнении полного сканирования таблицы в указанной таблице, независимо от того, может ли использоваться индекс.

```
create table fullTable(id) as select level from dual connect by level < 100000;
create index idx on fullTable(id);
```

Без подсказок индекс используется:

```
select count(1) from fullTable f where id between 10 and 100;
-----------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    13 |     3   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE   |      |     1 |    13 |            |          |
|*  2 |   INDEX RANGE SCAN| IDX  |     2 |    26 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

FULL hint заставляет полностью сканировать:

```
select /*+ full(f) */ count(1) from fullTable f where id between 10 and 100;
-------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |     1 |    13 |    47   (3)| 00:00:01 |
|   1 |  SORT AGGREGATE    |           |     1 |    13 |            |          |
|*  2 |   TABLE ACCESS FULL| FULLTABLE |     2 |    26 |    47   (3)| 00:00:01 |
-------------------------------------------------------------------------------
```

Кэш результатов

Oracle (*11g и выше*) позволяет SQL-запросам кэшироваться в SGA и повторно использовать для повышения производительности. Он запрашивает данные из кеша, а не из базы данных. Последующее выполнение одного и того же запроса происходит быстрее, поскольку теперь данные извлекаются из кеша.

```
SELECT /*+ result_cache */ number FROM main_table;
```

Выход -

```
Number
------
   1
   2
   3
   4
   5
   6
   7
   8
   9
   10

Elapsed: 00:00:02.20
```

Если я снова запустил тот же запрос, время выполнения будет уменьшаться, поскольку данные теперь извлекаются из кеша, который был установлен во время первого выполнения.

Выход -

```
Number
------
   1
   2
   3
   4
   5
   6
   7
   8
   9
   10

Elapsed: 00:00:00.10
```

Обратите внимание, как прошедшее время сократилось с **2.20 секунд** до **0,10 секунды** .

> Кэш результатов хранит кеш, пока данные в базе данных не будут обновлены / изменены / удалены. Любое изменение освободит кеш.

Прочитайте Советы онлайн: https://riptutorial.com/ru/oracle/topic/1490/советы

глава 28: Создание контекста

Синтаксис

- СОЗДАТЬ [ИЛИ ЗАМЕНИТЬ] пространство имен CONTEXT ИСПОЛЬЗОВАНИЕ [схема];
- СОЗДАТЬ [ИЛИ ЗАМЕНИТЬ] пространство имен CONTEXT ИСПОЛЬЗОВАНИЕ [схема] пакет INITIALIZED EXTERNALLY;
- СОЗДАТЬ [ИЛИ ЗАМЕНИТЬ] пространство имен CONTEXT ИСПОЛЬЗОВАНИЕ [схема] пакет INITIALIZED GLOBALY;
- СОЗДАТЬ [ИЛИ ЗАМЕНИТЬ] пространство имен CONTEXT ИСПОЛЬЗОВАНИЕ [схема] пакет ДОСТУП К ГЛОБАЛЬНО;

параметры

параметр	подробности
`OR REPLACE`	Переопределить существующее пространство имен контекста
Пространство имен	Имя контекста - это пространство имен для вызовов `SYS_CONTEXT`
схема	Владелец пакета
пакет	Пакет базы данных, который устанавливает или сбрасывает атрибуты контекста. Примечание. Пакет базы данных не должен существовать для создания контекста.
`INITIALIZED`	Укажите объект, отличный от базы данных Oracle, который может установить контекст.
`EXTERNALLY`	Разрешить интерфейсу OCI инициализировать контекст.
`GLOBALLY`	Разрешить каталогу LDAP инициализировать контекст при создании сеанса.
`ACCESSED GLOBALLY`	Разрешить контекст быть доступным во всем экземпляре - несколько сеансов могут совместно использовать значения атрибутов, если они имеют одинаковый идентификатор клиента.

замечания

Документация Oracle (12cR1):
http://docs.oracle.com/database/121/SQLRF/statements_5003.htm

Examples

Создать контекст

```
CREATE CONTEXT my_ctx USING my_pkg;
```

Это создает контекст, который может быть установлен только подпрограммами в пакете базы данных `my_pkg` , например:

```
CREATE PACKAGE my_pkg AS
  PROCEDURE set_ctx;
END my_pkg;

CREATE PACKAGE BODY my_pkg AS
  PROCEDURE set_ctx IS
  BEGIN
    DBMS_SESSION.set_context('MY_CTX','THE KEY','Value');
    DBMS_SESSION.set_context('MY_CTX','ANOTHER','Bla');
  END set_ctx;
END my_pkg;
```

Теперь, если сеанс делает это:

```
my_pkg.set_ctx;
```

Теперь он может получить значение для ключа таким образом:

```
SELECT SYS_CONTEXT('MY_CTX','THE KEY') FROM dual;

Value
```

Прочитайте Создание контекста онлайн: https://riptutorial.com/ru/oracle/topic/2088/создание-контекста

глава 29: Ссылки на базы данных

Examples

Создание ссылки на базу данных

```
CREATE DATABASE LINK dblink_name
CONNECT TO remote_username
IDENTIFIED BY remote_password
USING 'tns_service_name';
```

Затем удаленная БД будет доступна следующим образом:

```
SELECT * FROM MY_TABLE@dblink_name;
```

Чтобы протестировать соединение с базой данных без необходимости знать какое-либо из имен объектов в связанной базе данных, используйте следующий запрос:

```
SELECT * FROM DUAL@dblink_name;
```

Чтобы явно указать домен для службы связанных баз данных, доменное имя добавляется в оператор `USING` . Например:

```
USING 'tns_service_name.WORLD'
```

Если имя домена не указано явно, Oracle использует домен базы данных, в которой создается ссылка.

Документация Oracle для создания ссылок на базы данных:

- 10g: https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_5005.htm
- 11g: https://docs.oracle.com/cd/B28359_01/server.111/b28310/ds_concepts002.htm
- 12g: https://docs.oracle.com/database/121/SQLRF/statements_5006.htm#SQLRF01205

Создать ссылку на базу данных

Предположим, что у нас есть две базы данных «ORA1» и «ORA2». Мы можем получить доступ к объектам «ORA2» из базы данных «ORA1», используя ссылку базы данных.

Предварительные требования: для создания частной ссылки базы данных вам нужна привилегия `CREATE DATABASE LINK` . Для создания частной ссылки базы данных вам нужна привилегия `CREATE PUBLIC DATABASE LINK` .

* Oracle Net должен присутствовать на обоих экземплярах.

Как создать ссылку на базу данных:

Из ORA1:

```
SQL> create <public> database link ora2 connect to user1 identified by pass1 using <tns name
of ora2>;
```

Создана ссылка на базу данных.

Теперь, когда мы установили связь с БД, мы можем доказать, что, выполнив следующее из ORA1:

```
SQL> Select name from V$DATABASE@ORA2; -- should return ORA2
```

Вы также можете получить доступ к объектам БД «ORA2» из «ORA1», учитывая, что пользователь `user1` имеет привилегию `SELECT` для этих объектов на ORA2 (например, ТАБЛИЦА 1 ниже):

```
  SELECT COUNT(*) FROM TABLE1@ORA2;
```

Предварительно requistes:

- Обе базы данных должны быть запущены и открыты (открыты).
- Оба слушателя базы данных должны быть запущены и запущены.
- TNS должен быть настроен правильно.
- Пользователь user1 должен присутствовать в базе данных ORA2, пароль должен быть проверен и проверен.
- Пользователь user1 должен иметь хотя бы привилегию `SELECT` или любую другую, необходимую для доступа к объектам на ORA2.

Прочитайте Ссылки на базы данных онлайн: https://riptutorial.com/ru/oracle/topic/3859/ссылки-на-базы-данных

глава 30: Статистические функции

Examples

Вычисление медианы набора значений

Функция MEDIAN, поскольку Oracle 10g является простой в использовании функцией агрегации:

```
SELECT MEDIAN(SAL)
FROM EMP
```

Он возвращает медиану значений

Работает и с значениями `DATETIME` .

> Результат MEDIAN вычисляется путем первого упорядочения строк. Используя N как количество строк в группе, Oracle вычисляет интересующий номер строки (RN) с помощью формулы RN = (1 + (0,5 * (N-1)). Окончательный результат агрегатной функции вычисляется линейным интерполяция между значениями из строк в номерах строк CRN = CEILING (RN) и FRN = FLOOR (RN).

Начиная с Oracle 9i вы можете использовать PERCENTILE_CONT, который работает так же, как и функция MEDIAN, при этом значение процентиля по умолчанию равно 0,5

```
SELECT PERCENTILE_CONT(.5) WITHIN GROUP(order by SAL)
FROM EMP
```

VARIANCE

Отклонение измеряет, насколько далеко распределены множество чисел из его значения. С практической точки зрения это квадрат расстояния от его среднего (в центре) - чем больше число, тем дальше.

Следующий пример возвращает отклонение значений зарплаты

```
SELECT name, salary, VARIANCE(salary) "Variance"
FROM employees
```

STDDEV

STDDEV возвращает стандартное стандартное отклонение выражения expr, набор чисел. Вы можете использовать его как совокупную, так и аналитическую функцию. Он отличается от STDDEV_SAMP тем, что STDDEV возвращает ноль, когда он имеет только

одну строку входных данных, тогда как STDDEV_SAMP возвращает значение null.

Oracle Database вычисляет стандартное отклонение как квадратный корень из дисперсии, определенной для агрегатной функции VARIANCE.

Эта функция принимает в качестве аргумента любой числовой тип данных или какой-либо нечетный тип данных, который может быть неявно преобразован в числовой тип данных. Функция возвращает тот же тип данных, что и числовой тип данных аргумента.

Если вы укажете DISTINCT, вы можете указать только query_partition_clause аналитического_класса. Нельзя разрешить order_by_clause и windowing_clause.

В следующем примере возвращается стандартное отклонение заработной платы в таблице **hr.employees** :

Где hr - схема, а сотрудники - это имя таблицы.

```
SELECT STDDEV(salary) "Deviation"
FROM employees;

Deviation
----------
3909.36575
```

Запрос в следующем примере возвращает кумулятивное стандартное отклонение заработной платы в Департаменте 80 в таблице образцов hr.employees, упорядоченное по методу rent_date:

```
SELECT last_name, salary,
STDDEV(salary) OVER (ORDER BY hire_date) "StdDev"
FROM employees
WHERE department_id = 30;

LAST_NAME                     SALARY     StdDev
------------------------- ---------- ----------
Raphaely                       11000          0
Khoo                            3100 5586.14357
Tobias                          2800  4650.0896
```

Прочитайте Статистические функции онлайн: https://riptutorial.com/ru/oracle/topic/2283/статистические-функции

глава 31: Таблица DUAL

замечания

Таблица `DUAL` имеет один столбец `DUMMY` , определенный как `VARCHAR2(1)` и только одна строка со значением `x` .

Таблица `DUAL` автоматически создается в схеме `SYS` при создании базы данных. Вы можете получить к нему доступ из любой схемы.

Вы не можете изменить таблицу `DUAL` .

Вы можете использовать таблицу `DUAL` для вызова любой функции из инструкции SQL. Это полезно, потому что у него есть только одна строка, и оптимизатор оракула знает все об этом.

Examples

Следующий пример возвращает текущую дату и время операционной системы

```
select sysdate from dual
```

В следующем примере генерируются числа между start_value и end_value

```
select :start_value + level -1 n
from dual
connect by level <= :end_value  - :start_value + 1
```

Прочитайте Таблица DUAL онлайн: https://riptutorial.com/ru/oracle/topic/7328/таблица-dual

глава 32: уровень запроса

замечания

условие уровня отвечает за генерирование N число фиктивных записей на основе некоторых конкретных условий.

Examples

Создать N Количество записей

```
SELECT ROWNUM NO FROM DUAL CONNECT BY LEVEL <= 10
```

Несколько способов запроса уровня

/ * Это простой запрос, который может генерировать последовательность чисел. В следующем примере генерируется последовательность чисел от 1..100 * /

```
select level from dual connect by level <= 100;
```

/ * Вышеприведенный запрос полезен в различных сценариях, таких как генерация последовательности дат с определенной даты. Следующий запрос генерирует 10 последовательных дат * /

```
select to_date('01-01-2017','mm-dd-yyyy')+level-1 as  dates from dual connect by level <= 10;
```

01-JAN-17
02-JAN-17
03-JAN-17
04-JAN-17
05-JAN-17
06-JAN-17
07-JAN-17
08-JAN-17
09-JAN-17
10-JAN-17

Прочитайте уровень запроса онлайн: https://riptutorial.com/ru/oracle/topic/6548/уровень-запроса

глава 33: Функции окна

Синтаксис

- Ratio_To_Report (expr) OVER (query_partition_clause)

Examples

Ratio_To_Report

Соотношение текущего значения строк ко всем значениям в окне.

```
--Data
CREATE TABLE Employees (Name Varchar2(30), Salary Number(10));
INSERT INTO Employees Values ('Bob',2500);
INSERT INTO Employees Values ('Alice',3500);
INSERT INTO Employees Values ('Tom',2700);
INSERT INTO Employees Values ('Sue',2000);
--Query
SELECT Name, Salary, Ratio_To_Report(Salary) OVER () As Ratio
FROM Employees
ORDER BY Salary, Name, Ratio;
--Output
NAME                           SALARY      RATIO
------------------------------ ---------- ----------
Sue                                  2000 .186915888
Bob                                  2500  .23364486
Tom                                  2700 .252336449
Alice                                3500 .327102804
```

Прочитайте Функции окна онлайн: https://riptutorial.com/ru/oracle/topic/6669/функции-окна

кредиты

S. No	Главы	Contributors
1	Начало работы с Oracle Database	Community, J. Chomel, Jeffrey Kemp, Jon Ericson, Kevin Montrose, Mark Stewart, Sanjay Radadiya, Steven Feuerstein, tonirush
2	JOINS	Aleksej, B Samedi, Bakhtiar Hasan, Daniel Langemann, Erkan Haspulat, Pranav Shah, Robin James, SriniV, Sumner Evans
3	Oracle Advanced Queuing (AQ)	Jon Theriault
4	Oracle MAF	Anand Raj
5	Автономные транзакции	phonetic_man
6	Анонимный блок PL / SQL	Jon Heller, Skynet, Zohar Elkayam
7	Даты	carlosb, MT0, Roman, tonirush
8	Динамический SQL	Dmitry
9	Иерархический поиск с помощью Oracle Database 12C	Muntasir, Vahid
10	Индексы	smshafiqulislam
11	Манипуляция строк	carlosb, Eric B., Florin Ghita, Francesco Serra, J. Chomel, J.Hudler, Jeffrey Kemp, Mark Stewart, SriniV, Thunder, walen, zhliu03
12	Насос данных	Vidya Thotangare
13	Обновление с помощью объединений	mathguy
14	Обработка значений NULL	Dalex, JeromeFr
15	Ограничение строк,	Ahmed Mohamed, Martin Schapendonk, Matas Vaitkevicius,

	возвращаемых запросом (Pagination)	Sanjay Radadiya, tonirush, trincot
16	ограничения	SSD
17	Последовательности	Pranav Shah, SriniV
18	Работа с датами	David Aldridge, Florin Ghita, Jeffrey Kemp, Mark Stewart, tonirush, zygimantus
19	Разделение разделительных строк	Arkadiusz Łukasiewicz, MT0
20	Разделение таблиц	BobC, carlosb, ivanzg, JeromeFr, Kamil Islamov, Stephen Leppik, tonirush
21	Различные способы обновления записей	nimour pristou, Nogueira Jr, SriniV
22	Разметка ключевых слов или специальных символов	dev
23	Реальная безопасность приложений	Ben H
24	Регистрация ошибок	zygimantus
25	Рекурсивный факторинг подзапросов с использованием предложения WITH (общие выражения таблицы AKA)	B Samedi, MT0
26	Словарь данных	Mark Stewart, Pancho, Slava Babin
27	Советы	Aleksej, Florin Ghita, Jon Heller, Mark Stewart, Pirate X
28	Создание контекста	Jeffrey Kemp
29	Ссылки на базы	carlosb, Daniel Langemann, g00dy, kasi

	данных	
30	Статистические функции	Evgeniy K., Matas Vaitkevicius, ppeterka, Pranav Shah
31	Таблица DUAL	Slava Babin
32	уровень запроса	Sanjay Radadiya, TechEnthusiast
33	Функции окна	Leigh Riffel