**Общие положения**

**Общая информация**

В версии СУБД Oracle 8.1.6 появился новый класс из 26 функций, названных аналитическими, и получившим дальнейшее развитие в [версии 9](http://www.interface.ru/oracle/or9i_db.htm). Их описания были созданы совместными усилиями фирм IBM, Informix, Oracle и Compaq путем разработки так называемых "улучшений" некоторых конструкций, имеющихся в стандарте SQL1999.

В отличие от обычных скалярных функций аналитические функции берут аргументом SQL-таблицу, представляющую логический промежуточный результат обработки SQL-оператора, где использовано обращение к такой функции, и возвращают в качестве своего результата обычно тоже SQL-таблицу.

**Цели введения аналитических функций в Oracle**

Техническая цель введения аналитических функций - дать лаконичную формулировку и увеличить скорость выполнения "аналитических запросов" к БД, то есть запросов, имеющих смыслом выявление внутренних соотношений и зависимостей в данных. Более точно, пользование аналитическими функциями может дать следующие выгоды перед обычными SQL-операторами:

* **Лаконичную и простую формулировку**. Многие аналитические запросы к БД традиционными средствами сложно формулируются, а потому с трудом осмысливаются и плохо отлаживаются.
* **Снижение нагрузки на сеть**. То, что раньше могло формулироваться только серией запросов, сворачивается в один запрос. По сети только отправляется запрос и получается окончательный результат.
* **Перенос вычислений на сервер**. С использованием аналитических функций нет нужды организовывать расчеты на клиенте; они полностью проводятся на сервере, ресурсы которого могут быть более подходящи для быстрой обработки больших объемов данных.
* **Лучшую эффективность обработки запросов**. Аналитические функции имеют алгоритмы вычисления, неразрывно связанные со специальными планами обработки запросов, оптимизированными для большей скорости получения результата.

Стратегическая цель введения в Oracle аналитических функций - дать базовое средство для построения ИС типа "складов данных" (data warehouse, DW), ИС "аналитического характера" (business intelligence systems, BI) или OLAP-систем. По представлениям разработчиков, набор таких базовых средств помимо аналитических функций формируют еще и прочие средства Oracle, такие как

* конструкции ROLLUP, CUBE и связанные с ними в предложениях с GROUP BY
* материализованные выводимые таблицы (materialized views)

**Классификация видов аналитических функций в Oracle**

Согласно классификации из документации по Oracle, аналитические функции могут быть следующих видов:

**(a)** функции ранжирования **(b)** статистические функции для плавающего интервала **(c)** функции подсчета долей
**(d)** статистические функции LAG/LEAD с запаздывающим/опережающим аргументом
**(e)** статистические функции (линейная регрессия и т. д.)

**Основные технические особенности**

**Место указания аналитических функций в SQL-предложении**

Аналитические функции принимают в качестве аргумента столбец промежуточного результата вычисления SQL-предложения и возвращают тоже столбец. Поэтому местом их использования в SQL-предложении могут быть только фразы ORDER BY и SELECT, выполняющие завершающую обработку логического промежуточного результата.

**Сравнение с обычными функциями агрегирования**

Многие аналитические функции действуют подобно обычным скалярным функциям агрегирования SUM, MAX и прочим, примененным к группам строк, сформированным с помощью GROUP BY. Однако обычные функции агрегирования уменьшают степень детализации, а аналитические функции нет. Поясняющий сравнительный пример:

SELECT deptno, job, **SUM(sal)** sum\_sal
FROM emp **GROUP BY deptno, job;**

SELECT ename, deptno, job,
**SUM(sal) OVER (PARTITION BY deptno, job) sum\_sal**
FROM emp;

Результат первого запроса:

**DEPTNO             JOB**                     SUM\_SAL
----------                  ---------                         ----------

|  |  |  |  |
| --- | --- | --- | --- |
| **10** | **CLERK**  | 1300  |  <**- -** одна группа |
| **10** | **MANAGER**  | 2450  |  <**- -** одна группа |
| **10** | **PRESIDENT**  | 5000 |  < одна группа |
| **20** | **CLERK**  | 6000 |  <**- -** одна группа |
| **20** | **MANAGER**  | 1900 |  |
| **20** | **PRESIDENT**  | 2975 |  |
| **30** | **CLERK**  | 950 |  |
| **30** | **MANAGER**  | 2850 |  |
| **30** | **PRESIDENT**  | 5600 |  |

**9 rows** selected.

Результат второго запроса:

ENAME                    **DEPTNO           JOB** SUM\_SAL
----------                        ----------             ---------                 ----------

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MILLER | **10** | **CLERK** | 1300 | <- - одна группа |
| CLARK  | **10** | **MANAGER** | 2450 | <- - еще одна группа |
| KING | **10** | **PRESIDENT**  | 5000 | <- - еще одна группа |
| SCOTT | **20** | **ANALYST** | 6000  | <- - еще одна группа |
| FORD  | **20** | **ANALYST** | 6000  |   |
| SMITH  | **20** | **CLERK** | 1900 | <- - еще одна группа |
| ADAMS  | **20** | **CLERK** | 1900 |   |
| JONES  | **20** | **MANAGER** | 2975  | <- - еще одна группа |
| JAMES  | **30** | **CLERK** | 950 | <- - еще одна группа |
| BLAKE  | **30** | **MANAGER** | 2850  | <- - еще одна группа |
| ALLEN  | **30** | **SALESMAN**  | 5600 | <- - еще одна группа |
| MARTIN | **30** | **SALESMAN**  | 5600 |   |
| TURNER | **30** | **SALESMAN**  | 5600 |   |
| WARD  | **30** | **SALESMAN**  | 5600 |   |

**14 rows** selected.

**Особенности обработки**

Построим в SQL\*Plus планы для двух запросов выше:

SET AUTOTRACE TRACEONLY EXPLAIN

SELECT deptno, job, SUM(sal) sum\_sal
FROM emp
GROUP BY deptno, job;

SELECT empno, deptno, job,
SUM(sal) OVER (PARTITION BY deptno, job) sum\_sal
FROM emp;

SET AUTOTRACE OFF

Обратим внимание на однопроходность и специальный шаг плана второго запроса (шаг WINDOW).

**Разбиение данных на группы для вычислений**

Аналитические функции агрегируют данные порциями (partitions; группами), количество и размер которых можно регулировать специальной синтаксической конструкцией. Ниже она указана на примере агрегирующей функции SUM:

SUM(выражение 1) OVER([PARTITION BY выражение 2 [, выражение 3 [, …]]])

Пример использования такой конструкции см. выше.

Если PARTITION BY не указано, то в качестве единственной группы для вычислений будет взят полный набор строк:

SELECT ename, deptno, job,
SUM(sal) OVER () sum\_sal
FROM emp;

Результат последнего запроса:

ENAME DEPTNO JOB SUM\_SAL
---------- ---------- --------- ----------

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SMITH | 20 | CLERK  | 29025  | <- -  единственная группа, |
| ALLEN | 30  | SALESMAN  | 29025  |      и сумма на всех одна |
| WARD  | 30  | SALESMAN | 29025  |   |
| JONES  | 20  | MANAGER | 29025  |   |
| MARTIN  | 30  | SALESMAN | 29025  |   |
| BLAKE  | 30 | MANAGER | 29025  |   |
| CLARK  | 10 | MANAGER | 29025  |   |
| SCOTT  | 20 | ANALYST  | 29025  |   |
| KING  | 10  | PRESIDENT  | 29025  |   |
| TURNER  | 30  | SALESMAN | 29025  |   |
| ADAMS  | 20  | CLERK | 29025  |   |
| JAMES  | 30 | CLERK | 29025  |   |
| FORD  | 20 | ANALYST  | 29025  |   |
| MILLER  | 10 | CLERK  | 29025  |   |

14 rows selected.

**Упорядочение в границах отдельной группы**

С помощью синтаксической конструкции ORDER BY строки в группах вычислений можно упорядочивать. Синтаксис иллюстрируется на примере агрегирующей функции SUM:

SUM(выражение 1) OVER([PARTITION …]
ORDER BY выражение 2 [,…] [{ASC|DESC}] [{NULLS FIRST|NULLS LAST}])

Правила работы ORDER BY - как в обычных SQL-операторах. Пример:

SELECT ename, deptno, job,
SUM(sal) OVER (PARTITION BY deptno, job **ORDER BY hiredate**) sum\_sal
FROM emp;

ENAME             DEPTNO           JOB                            SUM\_SAL
----------                    ----------               ---------                    ----------

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MILLER  | 10 | CLERK | 1300 |   |
| CLARK  | 10 | MANAGER | 2450 |   |
| KING  | 10 | PRESIDENT  | 5000 |   |
| FORD  | 20 | ANALYST  | 3000 | <- - порядок и сумма изменились |
| SCOTT  | 20 | ANALYST  | 6000 |   |
| SMITH  | 20 | CLERK | 800  | <- - порядок и сумма изменились |
| ADAMS  | 20 | CLERK | 1900 |   |
| JONES  | 20 | MANAGER | 2975 |   |
| JAMES | 30 | CLERK  | 950 |   |
| BLAKE | 30 | MANAGER | 2850 |   |
| ALLEN  | 30 | SALESMAN  | 1600 | <- - порядок и сумма изменились |
| WARD  | 30 | SALESMAN  | 2850 |   |
| TURNER  | 30 | SALESMAN  | 4350 |   |
| MARTIN  | 30 | SALESMAN  | 5600 |   |

14 rows selected.

В группах из более одной строки появился заданный порядок. Природа изменения поля SUM\_SAL в пределах групп из нескольких строк станет ясна из следующего раздела.

**Выполнение вычислений для строк в группе по плавающему окну (интервалу)**

Для некоторых аналитических функций, например, агрегирующих, можно дополнительно указать объем строк, участвующих в вычислении, выполняемом для каждой строки в группе. Этот объем, своего рода контекст строки, называется "окном", а границы окна могут задаваться различными способами.

{ROWS | RANGE} {{UNBOUNDED | выражение} PRECEDING | CURRENT ROW }

{ROWS | RANGE}
BETWEEN
{{UNBOUNDED PRECEDING | CURRENT ROW |
{UNBOUNDED | выражение 1}{PRECEDING | FOLLOWING}}
AND
{{UNBOUNDED FOLLOWING | CURRENT ROW |
{UNBOUNDED | выражение 2}{PRECEDING | FOLLOWING}}

Фразы PRECEDING и FOLLOWING задают верхнюю и нижнюю границы агрегирования (то есть интервал строк, "окно" для агрегирования).

Вот поясняющий пример, воспроизводящий результат из предыдущего раздела:

SELECT ename, deptno, job,
SUM(sal)
OVER (PARTITION BY deptno, job ORDER BY hiredate
**ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) sum\_sal**
FROM emp;

ENAME DEPTNO JOB SUM\_SAL
---------- ---------- --------- ----------

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MILLER  | 10 | CLERK  | 1300 |   |
| CLARK | 10 | MANAGER  | 2450 |   |
| KING  | 10 | PRESIDENT  | 5000 |   |
| FORD  | 20 | ANALYST | **3000** | <- - зарплата FORD'а |
| SCOTT  | 20 | ANALYST | **6000**  | <- - сумма FORD'а и SCOTT'а |
| SMITH | 20 | CLERK  | **800** | <- - зарплата SMITH'а |
| ADAMS  | 20 | CLERK  | **1900** | <- - сумма SMITH'а и ADAMS'а |
| JONES  | 20 | MANAGER  | 2975 |   |
| JAMES  | 30 | CLERK  | 950 |   |
| BLAKE | 30 | MANAGER  | 2850 |   |
| ALLEN | 30 | SALESMAN  | **1600**  | <- - зарплата ALLEN'а |
| WARD  | 30 | SALESMAN  | **2850**  | <- - сумма ALLEN'а и WARD'а |
| TURNER  | 30 | SALESMAN  | **4350** | <- - ALLEN+WARD+TURNER |
| MARTIN  | 30 | SALESMAN  | **5600** | <- - ALLEN+WARD+TURNER+MARTIN |

14 rows selected.

Здесь в пределах каждой группы (использована фраза PARTITION BY) сотрудники упорядочиваются по времени найма на работу (фраза ORDER BY) и для каждого в группе вычисляется сумма зарплат: его и всех его предшественников (фраза ROWS BETWEEN формулирует "окошко суммирования" от первого в группе до текущего рассматриваемого).

Выделенная в последнем запросе жирным цветом фраза подразумевается по умолчанию, если она попросту отсутствует (ср. с запросом из предыдущего раздела).

Обратите внимание, что плавающий интервал задается в терминах упорядоченных строк (ROWS) или значений (RANGE), для чего фраза ORDER BY в определении группы обязана присутствовать.

**Формирование интервалов агрегирования "по строкам" и "по значениям"**

Разницу между ROWS и RANGE (определяющими, как говорится в документации, "физические" и "логические" интервалы-окна) удобно продемонстрировать следующим примером:

SELECT ename, **hiredate**, sal,
SUM(sal)
OVER (ORDER BY **hiredate**
**ROWS** BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) **rows\_sal,**
SUM(sal)
OVER (ORDER BY **hiredate**
**RANGE** BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) **range\_sal**
FROM emp;

ENAME HIREDATE SAL ROWS\_SAL RANGE\_SAL
---------- --------- ---------- ---------- ----------

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SMITH  | 17-DEC-80  | 800 | 800 | 800 |
| ALLEN | 20-FEB-81 | 1600  | 2400 | 2400 |
| WARD  | 22-FEB-81  | 1250 | 3650  | 3650  |
| JONES  | 02-APR-81  | 2975  | 6625 | 6625 |
| BLAKE  | 01-MAY-81  | 2850  | 9475 | 9475 |
| CLARK  | 09-JUN-81  | 2450  | 11925  | 11925  |
| TURNER  | 08-SEP-81  | 1500 | 13425  | 13425  |
| MARTIN  | 28-SEP-81  | 1250 | 14675 | 14675 |
| KING  | 17-NOV-81  | 5000 | 19675 | 19675 |
| JAMES | 03-DEC-81  | 950 | **20625** | **23625** |
| FORD | 03-DEC-81  | 3000  | **23625** | **23625** |
| MILLER  | 23-JAN-82  | 1300  | 24925 | 24925 |
| SCOTT  | 19-APR-87  | 3000  | 27925  | 27925  |
| ADAMS  | 23-MAY-87  | 1100  | 29025  | 29025  |

14 rows selected.

JAMES и FORD поступили на работу одновременно, и с точки зрения интервала суммирования неразличимы. Поэтому суммирование "по значению" присвоило им один и тот же общий для "мини-группы", образованной этой парой, результат - максимальную сумму, которая при всех возможных порядках перечисления сотрудников внутри этой пары будет всегда одинакова. Суммирование "по строкам" (ROWS) поступило иначе: оно упорядочило сотрудников в "мини-группе", образованной равными датами (на самом деле чисто произвольно) и подсчитало суммы, как будто бы у этих сотрудников был задан порядок следования.

**Функции FIRST\_VALUE и LAST\_VALUE для интервалов агрегирования**

Эти функции позволяют для каждой строки выдать первое значение ее окна и последнее. Пример:

SELECT ename, hiredate, sal,
**FIRST\_VALUE(sal)**
OVER (ORDER BY hiredate
ROWS **BETWEEN 2 PRECEDING AND CURRENT ROW) first\_rows,
LAST\_VALUE(sal)**
OVER (ORDER BY hiredate
ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) **last\_rows,
FIRST\_VALUE(sal)**
OVER (ORDER BY hiredate
RANGE BETWEEN 2 PRECEDING AND CURRENT ROW) **first\_range,
LAST\_VALUE(sal)**
OVER (ORDER BY hiredate
RANGE BETWEEN 2 PRECEDING AND CURRENT ROW) **last\_range**
FROM emp;

ENAME HIREDATE SAL FIRST\_ROWS LAST\_ROWS FIRST\_RANGE LAST\_RANGE
--------                      ---------             ------            ----------      ---------     -----------     ----------

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| SMITH  | 17-DEC-80  | 800 | **800**  | **800**  | **800**  | **800**  |
| ALLEN | 20-FEB-81 | 1600  | **800**  | **1600**  | **1600**  | **1600**  |
| WARD  | 22-FEB-81  | 1250 | **800**  | **1250** | **1600**  | **1250** |
| JONES  | 02-APR-81  | 2975  | **1600**  | **2975**  | **2975**  | **2975**  |
| BLAKE  | 01-MAY-81  | 2850  | 1250  | **2850**  | **2850**  | **2850**  |
| CLARK  | 09-JUN-81  | 2450  | **2975** | **2450** | **2450** | **2450** |
| TURNER  | 08-SEP-81  | 1500 | **2850**  | **1500**  | **1500**  | **1500**  |
| MARTIN  | 28-SEP-81  | 1250 | **2450** | **1250**  | **1250**  | **1250**  |
| KING  | 17-NOV-81  | 5000 | **1500** | **5000**  | **5000**  | **5000**  |
| JAMES | 03-DEC-81  | 950 | **1250**  | **950**  | **950**  | **3000** |
| FORD | 03-DEC-81  | 3000  | **5000** | **3000** | **950**  | **3000** |
| MILLER  | 23-JAN-82  | 1300  | **950**  | **1300** | **1300** | **1300** |
| SCOTT  | 19-APR-87  | 3000  | **3000**  | **3000**  | **3000**  | **3000**  |
| ADAMS  | 23-MAY-87  | 1100  | **1300** | **1100** | **1100** | **1100** |

14 rows selected.

**Интервалы времени**

Для интервалов (окон), упорядоченных внутри по значению ("логическом", RANGE) в случае, если это значение имеет тип "дата", границы интервала можно указывать выражением над датой, а не конкретными значениями из строк. Примеры таких выражений:

INTERVAL число {YEAR | MONTH | DAY | HOUR | MINUTE | SECOND}

NUMTODSINTERVAL(число, '{DAY | HOUR | MINUTE | SECOND}')

NUMTOYMINTERVAL(число, '{YEAR | MONTH}')

Пример выдачи зарплат сотрудников и средних зарплат за последние полгода на момент приема нового сотрудника:

SELECT ename, hiredate, sal,
AVG(sal)
OVER (ORDER BY hiredate
**RANGE BETWEEN INTERVAL '6' MONTH PRECEDING** AND CURRENT ROW) avg\_sal
FROM emp;

ENAME          HIREDATE             SAL            AVG\_SAL
----------                   ---------               ----------            ----------

|  |  |  |  |
| --- | --- | --- | --- |
| SMITH  | 17-DEC-80  | 800 | 800 |
| ALLEN | 20-FEB-81 | 1600  | 1200 |
| WARD  | 22-FEB-81  | 1250 | 1216.66667 |
| JONES  | 02-APR-81  | 2975  | 1656.25 |
| BLAKE  | 01-MAY-81  | 2850  | 1895 |
| CLARK  | 09-JUN-81  | 2450  | 1987.5 |
| TURNER  | 08-SEP-81  | 1500 | 2443.75 |
| MARTIN  | 28-SEP-81  | 1250 | 2205 |
| KING  | 17-NOV-81  | 5000 | 2550 |
| JAMES | 03-DEC-81  | 950 | **2358.33333** |
| FORD | 03-DEC-81  | 3000  | **2358.33333** |
| MILLER  | 23-JAN-82  | 1300  | 2166.66667 |
| SCOTT  | 19-APR-87  | **3000**  | **3000**  |
| ADAMS  | 23-MAY-87  | 1100  | 2050 |

14 rows selected.

Вот другая запись для того же запроса, но позволяющая использовать для числа месяцев обычное числовое выражение:

SELECT ename, hiredate, sal,
AVG(sal)
OVER (ORDER BY hiredate
RANGE BETWEEN **NUMTOYMINTERVAL(6, 'MONTH')** PRECEDING
AND CURRENT ROW) avg\_sal
FROM emp;

**Виды аналических функций**

В качестве базовой в аналитической функции могут быть указаны традиционные для Oracle статистические (агрегатные, то есть обобщающие) функции COUNT, MIN, MAX, SUM, AVG и другие ("стандартные агрегатные функции" по документации). Примеры приводились выше. Можно обратить внимание на то, что аналитические функции со статистическими агрегатами разумно обрабатывают NULL:

SELECT ename, hiredate, sal,
**AVG(sal)**
OVER (ORDER BY hiredate
RANGE BETWEEN UNBOUNDED PRECEDING AND **INTERVAL '1' SECOND PRECEDING**) avg\_sal
FROM emp;

Ниже приводится полный перечень аналитических функций в версии СУБД 9.2:

|  |  |  |
| --- | --- | --- |
| AVG \* CORR \* COVAR\_POP \* COVAR\_SAMP \* COUNT \* CUME\_DIST DENSE\_RANK FIRST FIRST\_VALUE \* LAG LAST | LAST\_VALUE \* LEAD MAX \* MIN \* NTILE PERCENT\_RANK PERCENTILE\_CONT PERCENTILE\_DISC RANK RATIO\_TO\_REPORT | REGR\_ (вид\_функции\_линейной\_регрессии) \* ROW\_NUMBER STDDEV \* STDDEV\_POP \* STDDEV\_SAMP \* SUM \* VAR\_POP \* VAR\_SAMP \* VARIANCE  |

Звездочкой помечены функции, допускающие использование плавающего интервала расчета.

Некоторые из этих функций рассматриваются ниже.

**Функции ранжирования**

Функции ранжирования позволяют "раздать" строкам "места" в зависимости от имеющихся в них значениях. Некоторые примеры:

SELECT ename, sal,
**ROW\_NUMBER () OVER (ORDER BY sal DESC)** AS salbacknumber,
**ROW\_NUMBER () OVER (ORDER BY sal)** AS salnumber,
**RANK() OVER (ORDER BY sal)** AS salrank,
**DENSE\_RANK() OVER (ORDER BY sal)** AS saldenserank
FROM emp;

(раздать сотрудникам места в порядке убывания/возрастания зарплат)

**Функции подсчета долей**

Функции подсчета долей позволяют одной SQL-операцией получить для каждой строки ее "вес" в таблице в соответствии с ее значениями. Некоторые примеры:

SELECT ename, sal, **RATIO\_TO\_REPORT(sal) OVER () AS salshare FROM emp;**

(доли сотрудников в общей сумме зарплат)

Пример выдачи доли сотрудников с меньшей или равной зарплатой, чем у "текущего":

SELECT job, ename, sal,
**CUME\_DIST() OVER (PARTITION BY job ORDER BY sal**) AS cume\_dist
FROM emp;

JOB                       ENAME                      SAL                            CUME\_DIST
--------- -                     ---------                              ----------                     ----------

|  |  |  |  |
| --- | --- | --- | --- |
| ANALYST | SCOTT | **3000** | **1** |
| ANALYST | FORD  | **3000** | **1** |
| CLERK  | SMITH  | **800** | **.25** |
| CLERK  | JAMES | **950**  | **.5** |
| CLERK  | ADAMS | **1100**  | **.75** |
| CLERK  | MILLER | **1300** | **1** |
| MANAGER | CLARK  | **2450** | **.333333333** |
| MANAGER | BLAKE  | **2850**  | **.666666667** |
| MANAGER | JONES  | **2975** | **1** |
| PRESIDENT | KING  | **5000**  | **1** |
| SALESMAN  | WARD  | **1250**  | **.5** |
| SALESMAN  | MARTIN  | **1250**  | **.5** |
| SALESMAN  | TURNER  | **1500** | **.75** |
| SALESMAN  | ALLEN  | **1600** | **1** |

14 rows selected.

(видно, что три четверти клерков имеют зарплату, меньше чем ADAMS).

Проранжировать эту выдачу по доле сотрудников в группе можно функцией PERCENT\_RANK:

SELECT job, ename, sal,
**CUME\_DIST()** OVER (PARTITION BY job ORDER BY sal) AS **cume\_dist,**
**PERCENT\_RANK()** OVER (PARTITION BY job ORDER BY sal) AS **pct\_rank**
FROM emp;

JOB ENAME SAL CUME\_DIST **PCT\_RANK**
--------- ---------- ---------- ---------- ----------

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ANALYST | SCOTT | 3000 | 1 | **0** |
| ANALYST | FORD  | 3000 | 1 | **0** |
| CLERK  | SMITH  | 800 | .25 | **0** |
| CLERK  | JAMES | 950  | .5 | **.333333333** |
| CLERK  | ADAMS | 1100  | .75 | **.666666667** |
| CLERK  | MILLER | 1300 | 1 | **1** |
| MANAGER | CLARK  | 2450 | .333333333 | **0** |
| MANAGER | BLAKE  | 2850  | .666666667 | **.5** |
| MANAGER | JONES  | 2975 | 1 | **1** |
| PRESIDENT | KING  | 5000  | 1 | **0** |
| SALESMAN  | WARD  | 1250  | .5 | **0** |
| SALESMAN  | MARTIN  | 1250  | .5 | **0** |
| SALESMAN  | TURNER  | 1500 | .75 | **.666666667** |
| SALESMAN  | ALLEN  | 1600 | 1 | **1** |

14 rows selected.

Процентный ранг отсчитывается от 0 и изменяется до 1.

**Некоторые жизненные примеры аналитических запросов**

**Для типов сегментов, более других расходующих дисковое пространство, выдать главных пользователей, ответственных за такой расход**

Построить такой запрос на основе таблицы SYS.DBA\_SEGMENTS, можно пошагово.

**Шаг 1.** Выдать типы сегментов в БД, общий объем памяти на диске для каждого типа и долю числа типов с равным или меньшим общим объемом памяти:

SELECT segment\_type,
      SUM(bytes) bytes,
      CUME\_DIST() OVER (ORDER BY SUM(bytes)) bytes\_percentile
      FROM sys.dba\_segments
      GROUP BY segment\_type;

**Шаг 2.** Отобрать 40% "наиболее расточительных" по дисковой памяти типов:

**SELECT \*
FROM**
(SELECT segment\_type,
SUM(bytes) bytes,
CUME\_DIST() OVER (ORDER BY SUM(bytes)) bytes\_percentile
FROM sys.dba\_segments
GROUP BY segment\_type)
**WHERE bytes\_percentile >= 0.5;**

**Шаг 3.** Отобрать пользователей, занимающих первые пять мест по расходованию памяти среди "наиболее расточительных" типов сегментов:

**SELECT \*
FROM
(
SELECT owner,
        SUM(bytes) bytes,
        RANK() OVER(ORDER BY SUM(bytes) DESC) bytes\_rank
FROM sys.dba\_segments
WHERE segment\_type IN**
      (SELECT segment\_type
        FROM
           (SELECT segment\_type,
                SUM(bytes) bytes,
                CUME\_DIST() OVER (ORDER BY SUM(bytes)) bytes\_percentile
                FROM sys.dba\_segments
                GROUP BY segment\_type)
        WHERE bytes\_percentile >= 0.5)
**GROUP BY owner
)
WHERE bytes\_rank <=5**
/

**Выдать список периодов наиболее активного переключения журнальных файлов БД**

Список переключений журнальных файлов хранится в динамической таблице v$loghist. Ниже приводится один из вариантов запроса.

var treshold number
exec :treshold := 30
alter session set nls\_date\_format='MON-DD HH24:MI:SS';

SELECT
start\_time,
end\_time,
ROUND((end\_time - start\_time)\*24\*60, 2) delta\_min,
switches,
switches / ((end\_time - start\_time)\*24\*60) per\_minute
FROM
(
SELECT
MIN(time\_stamp) start\_time,
MAX(time\_stamp) end\_time,
count (\*) switches
FROM
(
SELECT time\_stamp, freq10, more,
SUM(ABS(indicator)) OVER (ORDER BY time\_stamp) part
FROM
(
SELECT time\_stamp, freq10,
SIGN(freq10 - :treshold - 0.5) more,
SIGN(freq10 - :treshold - 0.5) - LAG(SIGN(freq10 - :treshold - 0.5), 1)
OVER (ORDER BY time\_stamp) indicator
FROM
(
SELECT first\_time time\_stamp,
GREATEST(
COUNT(\*)
OVER (ORDER BY first\_time
RANGE BETWEEN CURRENT ROW AND INTERVAL '10' MINUTE FOLLOWING)
,
COUNT(\*)
OVER (ORDER BY first\_time
RANGE BETWEEN INTERVAL '10' MINUTE PRECEDING AND CURRENT ROW)
) freq10
FROM v$loghist
) /\* frequency table \*/
) /\* frequency treshold overcome table \*/
) /\* transient partitioned table \*/
WHERE more > 0
GROUP BY part
)
WHERE (end\_time - start\_time)\*24\*60 > 0
/

**Пояснения.**

* Фактически проверяется не частота переключений журнальных файлов, а частота фиксации первого изменения в журнальных файлах. Это не совсем одно и то же, но, похоже, сильно коррелирующие события.
* Результат получается в несколько проходов. Сначала для каждой записи проверяется средняя активность переключений в 10-минутные предшествующий и последующий интервалы. Затем выбираются записи, для которых средняя активность превышает порог :treshold = 30 в минуту. Затем размечаются точки перехода через порог, которые далее служат границами групп "повышенной" и "пониженной" активности. Потом интервалы с повышенной активностью выдаются на экран.