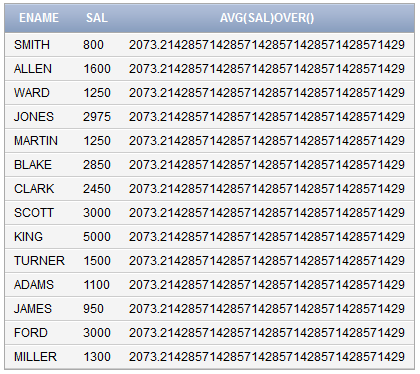
|  |
| --- |
| Лабораторная работа № 15 Аналитические функции в Oracle (Oracle Database 11g Express Edition)  **Цели введения аналитических функций в Oracle**  Техническая цель введения аналитических функций - дать лаконичную формулировку и увеличить скорость выполнения "аналитических запросов" к БД, 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. |
| |  |  |  | | --- | --- | --- | | 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 |   **Аналитический SQL**демонстрирует исключительную производительность, запросы с использованием Аналитического SQL потребляют меньше ресурсов и выполняются быстрее.  Применяется в основном для отчетности , для следующих типов задач 1. Запросы рейтингов, первых N 2. Запросы с накопительным итогом 3. Запросы с конструкцией окна 4. Может применятся и для оптимизации запросов  Какие бывают функции в аналитическом, разберём основные **ROW\_NUMBER() – номер строки в группе LAG(f, n,m) –f имя поля, n предыдущее значение в группе, m – знач по умолчанию LEAD(f, n,m) -f имя поля, n последующее значение в группе, m – знач по умолчанию FIRST\_VALUE(f) – f имя поля, первое значение в группе , LAST\_VALUE(f) –f имя поля, последнее значение в группе STD\_DEV(f) – f имя поля, значение стандартного распределения в группе SUM(f) – f имя поля, накопительная сумма по группе AVG (f)– f имя поля, среднее по группе заданной групп RANK(f) – f имя поля, относительный ранг записи в группе**  Синтаксис  SELECT аналитическая функция OVER([PARTITION партицирование…]  ORDER BY (упорядочивание выражение 2 [,…] [{ASC/DESC}] [{NULLS FIRST/NULLS LAST}])  Где Партишен - это некоторое количество записей с общими ключами на которую будет распространятся действие Аналитической функции , сортировка внутри партишена осуществляется с помощью **order by**  Проще всего разобраться с аналитическими функциями на примерах. Подготовим необходимые данные для демонстрации  Ход работы (предварительные действия):  - запустите sqldeveloper (отчет по лабораторной работе – файл <ФИО\_гр.sql>)  - подсоединитесь под SYSTEM ([Tools] [SQL WorkSheet] [Connection]  - посмотрите пользователей (Run Statement - Ctrl + Enter)  select username from sys.dba\_users;  - просмотрите «ненужных» пользователей (не забывайте удалять их!)  - просмотрите «ненужные» таблицы (не забывайте удалять их!)  Ex.  select table\_name from sys.all\_tables where owner ='USER1';  desc user1.tab\_user1;  select table\_name from sys.all\_tables where owner ='USER%' and table\_name like 'TAB%';  drop user USER1 cascade;  select username from sys.dba\_users;  - просмотрите наличие таблицы с именем EMP  select owner, table\_name from sys.all\_tables where table\_name like 'EM%';  - cоздайте отдельного пользователя для выполнения данной работы, предоставьте ему определенные права  create user user1 identified by user1  default tablespace system  quota unlimited on system;  grant connect to user1;  grant create any table to user1;  **Задание 1. Создать рабочую таблицу**  SQL> create table emp (ename varchar2(20), deptno number, job varchar2(20), sal number, sum\_sal number);  **Задание 2. Заполнить рабочую таблицу**  SQL> insert into emp (ename, deptno, job, sal, sum\_sal) values ('Miller', 10, 'clerk', 1000, 0);   |  |  |  |  |  | | --- | --- | --- | --- | --- | | MILLER | **10** | **CLERK** | 1000 |  | | CLARK | **10** | **MANAGER** | 2000 |  | | KING | **10** | **PRESIDENT** | 5000 |  | | SCOTT | **20** | **ANALYST** | 6000 |  | | FORD | **20** | **ANALYST** | 6000 |  | | SMITH | **20** | **CLERK** | 7000 |  | | ADAMS | **20** | **CLERK** | 7000 |  | | JONES | **20** | **MANAGER** | 3000 |  | | JAMES | **30** | **CLERK** | 3000 |  | | BLAKE | **30** | **MANAGER** | 2000 |  | | ALLEN | **30** | **SALESMAN** | 5000 |  | | MARTIN | **30** | **SALESMAN** | 5000 |  | | TURNER | **30** | **SALESMAN** | 6000 |  | | WARD | **30** | **SALESMAN** | 6000 |  |   **Задание 3. 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** |  | <**- -** одна группа | | **10** | **MANAGER** |  | <**- -** одна группа | | **10** | **PRESIDENT** |  | < одна группа | | **20** | **CLERK** |  | <**- -** одна группа | | **20** | **MANAGER** |  |  | | **20** | **PRESIDENT** |  |  | | **30** | **CLERK** |  |  | | **30** | **MANAGER** |  |  | | **30** | **PRESIDENT** |  |  |   Результат второго запроса:  ENAME                    **DEPTNO           JOB** SUM\_SAL ----------                        ----------             ---------                 ----------   |  |  |  |  |  | | --- | --- | --- | --- | --- | | MILLER | **10** | **CLERK** |  | <- - одна группа | | CLARK | **10** | **MANAGER** |  | <- - еще одна группа | | KING | **10** | **PRESIDENT** |  | <- - еще одна группа | | SCOTT | **20** | **ANALYST** |  | <- - еще одна группа | | FORD | **20** | **ANALYST** |  |  | | SMITH | **20** | **CLERK** |  | <- - еще одна группа | | ADAMS | **20** | **CLERK** |  |  | | JONES | **20** | **MANAGER** |  | <- - еще одна группа | | JAMES | **30** | **CLERK** |  | <- - еще одна группа | | BLAKE | **30** | **MANAGER** |  | <- - еще одна группа | | ALLEN | **30** | **SALESMAN** |  | <- - еще одна группа | | MARTIN | **30** | **SALESMAN** |  |  | | TURNER | **30** | **SALESMAN** |  |  | | WARD | **30** | **SALESMAN** |  |  |   **Задание 4. Особенности обработки**  Построим в SQL 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).  **Задание 5. 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 |  | <- -  единственная группа, | | ALLEN | 30 | SALESMAN |  | и сумма на всех одна | | WARD | 30 | SALESMAN |  |  | | JONES | 20 | MANAGER |  |  | | MARTIN | 30 | SALESMAN |  |  | | BLAKE | 30 | MANAGER |  |  | | CLARK | 10 | MANAGER |  |  | | SCOTT | 20 | ANALYST |  |  | | KING | 10 | PRESIDENT |  |  | | TURNER | 30 | SALESMAN |  |  | | ADAMS | 20 | CLERK |  |  | | JAMES | 30 | CLERK |  |  | | FORD | 20 | ANALYST |  |  | | MILLER | 10 | CLERK |  |  |   .  **Задание 6 Упорядочение в границах отдельной группы**  В Задании появляется поле hiredate – необходимо внести изменения в таблицу EMP (например)  alter table emp add hiredate date;  desc emp;  update emp set hiredate= current\_date;  update emp set hiredate= current\_date-30 where deptno=10;  update emp set hiredate= current\_date-60 where deptno=20;  select \* from emp;  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 |  |  | | CLARK | 10 | MANAGER |  |  | | KING | 10 | PRESIDENT |  |  | | FORD | 20 | ANALYST |  | <- - порядок и сумма изменились | | SCOTT | 20 | ANALYST |  |  | | SMITH | 20 | CLERK |  | <- - порядок и сумма изменились | | ADAMS | 20 | CLERK |  |  | | JONES | 20 | MANAGER |  |  | | JAMES | 30 | CLERK |  |  | | BLAKE | 30 | MANAGER |  |  | | ALLEN | 30 | SALESMAN |  | <- - порядок и сумма изменились | | WARD | 30 | SALESMAN |  |  | | TURNER | 30 | SALESMAN |  |  | | MARTIN | 30 | SALESMAN |  |  |   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.  **Задание 7 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 |  |  | | CLARK | 10 | MANAGER |  |  | | KING | 10 | PRESIDENT |  |  | | FORD | 20 | ANALYST |  | <- - зарплата FORD'а | | SCOTT | 20 | ANALYST |  | <- - сумма FORD'а и SCOTT'а | | SMITH | 20 | CLERK |  | <- - зарплата SMITH'а | | ADAMS | 20 | CLERK |  | <- - сумма SMITH'а и ADAMS'а | | JONES | 20 | MANAGER |  |  | | JAMES | 30 | CLERK |  |  | | BLAKE | 30 | MANAGER |  |  | | ALLEN | 30 | SALESMAN |  | <- - зарплата ALLEN'а | | WARD | 30 | SALESMAN |  | <- - сумма ALLEN'а и WARD'а | | TURNER | 30 | SALESMAN |  | <- - ALLEN+WARD+TURNER | | MARTIN | 30 | SALESMAN |  | <- - ALLEN+WARD+TURNER+MARTIN |   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.  **Задание 8 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 |  |  |  |  | | ALLEN |  |  |  |  | | WARD |  |  |  |  | | JONES |  |  |  |  | | BLAKE |  |  |  |  | | CLARK |  |  |  |  | | TURNER |  |  |  |  | | MARTIN |  |  |  |  | | KING |  |  |  |  | | JAMES |  |  |  |  | | FORD |  |  |  |  | | MILLER |  |  |  |  | | SCOTT |  |  |  |  | | ADAMS |  |  |  |  |   .  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.  **Задание 10 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 |  |  |  |  |  |  | | ALLEN |  |  |  |  |  |  | | WARD |  |  |  |  |  |  | | JONES |  |  |  |  |  |  | | BLAKE |  |  |  |  |  |  | | CLARK |  |  |  |  |  |  | | TURNER |  |  |  |  |  |  | | MARTIN |  |  |  |  |  |  | | KING |  |  |  |  |  |  | | JAMES |  |  |  |  |  |  | | FORD |  |  |  |  |  |  | | MILLER |  |  |  |  |  |  | | SCOTT |  |  |  |  |  |  | | ADAMS |  |  |  |  |  |  |   **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 '2' MONTH PRECEDING**AND CURRENT ROW) avg\_sal FROM emp;  ENAME          HIREDATE             SAL            AVG\_SAL ----------                   ---------               ----------            ----------  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(2, '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;  Create table PersonA(Tbn number primary key, name varchar2(20), otd number, sal number);  *--Табельный номер , имя, отдел , зарплата*  Insert into PersonA(Tbn,name,otd,sal) values(1, 'Аня',10,9000);  Insert into PersonA(Tbn,name,otd,sal) values(2, 'Саша',10,5500);  Insert into PersonA(Tbn,name,otd,sal) values(3, 'Таня',10,7000);  Insert into PersonA(Tbn,name,otd,sal) values(4, 'Ваня',20,2300);  Insert into PersonA(Tbn,name,otd,sal) values(5, 'Олег',20,4300);  Insert into PersonA(Tbn,name,otd,sal) values(6, 'Коля',20,3900);  Insert into PersonA(Tbn,name,otd,sal) values(7, 'Таня',30,7000);  Insert into PersonA(Tbn,name,otd,sal) values(8, 'Макс',30,9000);  Insert into PersonA(Tbn,name,otd,sal) values(9, 'Таня',30,8500);  Insert into PersonA(Tbn,name,otd,sal) values(10,'Макс',30,9900);  Insert into PersonA(Tbn,name,otd,sal) values(11,'Олег',30,9900);  Insert into PersonA(Tbn,name,otd,sal) values(12,'Макс',30,9900);  Insert into PersonA(Tbn,name,otd,sal) values(13,'Макс',30,9900);  Insert into PersonA(Tbn,name,otd,sal) values(14,'Макс',30,9900);  Insert into PersonA(Tbn,name,otd,sal) values(15,'Макс',30,9900);  Insert into PersonA(Tbn,name,otd,sal) values(16,'Макс',30,7000);  Insert into PersonA(Tbn,name,otd,sal) values(17,'Таня',30,3500);  Запросы списка лидеров Первые три сотрудника с самой высокой зарплатой по отделам (партишен по отделу) select \* from ( select name , otd , sal , row\_number() over (partition by otd order by sal desc)  as num from personA) where num<4  Более корректно  select \* from ( select name , otd , sal  , rank() over (partition by otd order by sal desc) as num from personA) where num<4  **Внимание ! Пример демонстрирует отличие rank() от row\_number()**  По наименованию (партишен по отделу) сортировка по name  select \* from ( select name , otd , sal , row\_number() over (partition by otd order by name) as num  from personA) where num<4  Накопительный итог по зарплате   select name , otd , sal , sum(sal) over (partition by otd order by sal) as num from personA  Среднее по зарплате в рамках отдела  select name , otd , sal , avg(sal) over (partition by otd order by sal) as num from personA  Демонстрация работы **lag, leed** - сотрудник , отдел , зарплата , сотрудник с более большей заплатой (maxsl), , сотрудник с менее меньшей заплатой чем данный(minsl) в рамках отдела  select name , otd , sal , lead(name,1) over (partition by otd order by sal) as maxsl  , lag(name,1) over (partition by otd order by sal) as minsl  from personA  запрос демонстрирует конструкцию окна в рамках отдела , среднее по зарплате, вычисляется, не только в рамках отдела , но так же и в рамках окна из 3х строк  select name , otd , sal  , avg(sal) over (partition by otd order by sal ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) as num from personA  в запросе используется конструкция окна ROWS BETWEEN N PRECEDING AND CURRENT ROW то есть, для вычисления среднего avg(sal), считаются 3 предыдущие строки перед текущей строкой   select name , otd , sal  , first\_value(name) over (partition by otd) as maxsl, last\_value(name) over (partition by otd) as minsl  from personA  этот запрос демонстрирует применение аналитических функций **first\_value last\_value** |

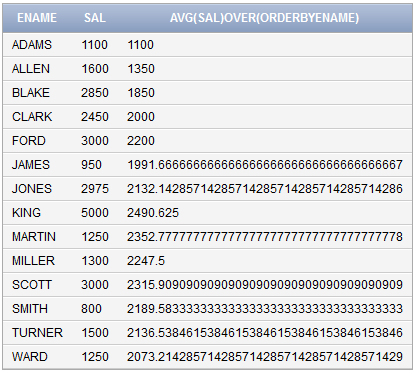
**ОЗНАКОМИТЬСЯ !!!**

**Oracle. Аналитические функции.**

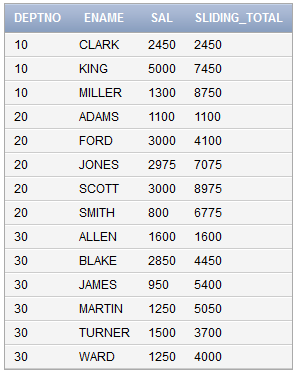
Общий синтаксис для использования аналитических функций следующий  
  
**имя\_функции(<аргумент>,< аргумент >, . . .** **)  
over (<конструкция\_фрагментации><конструкция\_упорядочения><конструкция\_окна>)**  
  
Рассмотрим основные части данного синтаксиса.  
  
***1. <конструкция\_фрагментации>***  
  
Синтаксис для задания конструкции фрагментации выглядит следующим образом  
  
**partition by** **выражение [, выражение] [, выражение]**  
  
Данная конструкция логически разбивает результирующее множество на N групп  
по критериям, задаваемым выражениями фрагментации. Аналитические функции применяются к каждой группе независимо, - для каждой новой группы они сбрасываются. Если не указать конструкцию фрагментации, все результирующее множество считается  
одной группой.  
  
***2. <конструкция\_упорядочения>***  
  
Конструкция упорядочения имеет следующий синтаксис  
  
**order by выражение [asc | desc] [nulls first | nulls last]**  
  
Конструкция **order by** задает критерий сортировки данных в каждой группе (в каждом фрагменте). Это, несомненно, влияет на результат выполнения любой аналитической функции. При наличии (или отсутствии) конструкции **order by** аналитические функции вычисляются по-другому. Например.  
- без конструкции order by  
  
**select ename, sal, avg(sal) over ()  
from emp**

[](http://4.bp.blogspot.com/-8x6KZw3aSzs/Ux7lcqpzYXI/AAAAAAAAA0o/fKT3qZqUgOA/s1600/s1.png)

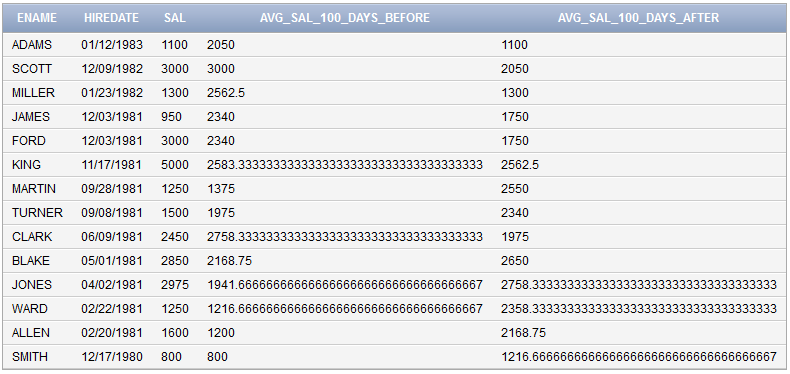
- с конструкцией order by  
  
**select ename, sal, avg(sal) over (order by ename)  
from emp**

[](http://4.bp.blogspot.com/-M0JPmJC-ukE/Ux7lzyMN-RI/AAAAAAAAA0w/_eCMbVrEWbE/s1600/s2.png)

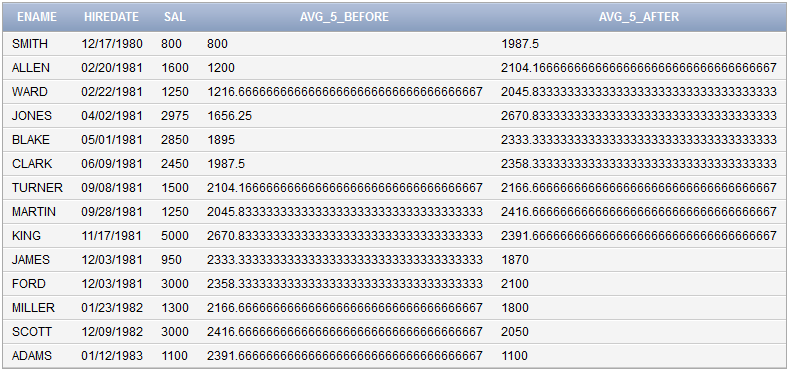
Здесь стоит отметить следующее, на самом деле наличие конструкции **order by** в вызове аналитической функции добавляет стандартную конструкцию окна — **RANGE UNBOUNDED PRECEDING**. Это означает, что для вычисления используется набор из всех предыдущих и текущей строки в текущем фрагменте. При отсутствии **order by** стандартным окном является весь фрагмент. То есть по-сути предыдущий запрос будет выглядеть следующим образом  
  
**select ename, sal, avg(sal) over (order by ename RANGE UNBOUNDED PRECEDING)  
from emp**  
  
***3. <конструкция\_окна>***  
  
Конструкция окна позволяет задать перемещающееся или жестко привязанное окно (набор)  
данных в пределах группы, с которым будет работать аналитическая функция.  
Например, для создания отчета, показывающего сумму зарплат текущего и двух предыдущих сотрудников отдела, можно выполнить следующий запрос  
  
**select deptno, ename, sal,  
sum(sal) over (partition by deptno order by ename rows 2 preceding) sliding\_total  
from emp  
order by deptno, ename**

[](http://4.bp.blogspot.com/-mCurYLm8n2s/Ux7mT_ZX2uI/AAAAAAAAA04/R8AGwSsm5fY/s1600/s3.png)

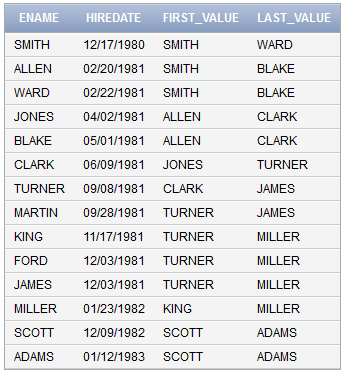
Можно создавать окна по двум критериям: *по диапазону (****RANGE****) значений данных* или *по  
смещению (****ROWS****) относительно текущей строки*. Использование конструкции **range**как было сказано ранее в  некоторых случаях используется неявно, **RANGE UNBOUNDED PRECEDING** например. Она требует брать все строки вплоть до текущей, в соответствии с порядком, задаваемым конструкцией **order by**. Следует помнить, что для использования окон  
необходимо задавать конструкцию **order by**.  
  
Окно определяется диапазоном строк, объединяемых в соответствии с заданным порядком.  
Применять конструкцию **range**можно либо с числовыми выражениями (**NUMBER**), либо с выражениями, значением которого является дата (**DATE**). Еще одно ограничение для таких окон состоит в том, что в конструкции **order by** может быть только один столбец — диапазоны по природе своей одномерны. Нельзя задать диапазон в N-мерном пространстве. Пример.  
Пусть необходимо выбрать зарплату каждого сотрудника и среднюю зарплату всех принятых на работу в течение 100 предыдущих дней, а также среднюю зарплату всех принятых на работу в течение 100 следующих дней. Соответствующий запрос будет выглядеть так:  
  
**select ename, hiredate, sal,**  
**avg(sal) over (order by hiredate asc range 100 preceding) avg\_sal\_100\_days\_before,**  
**avg(sal) over (order by hiredate desc range 100 preceding) avg\_sal\_100\_days\_after**  
**from emp**  
**order by hiredate desc**

[](http://3.bp.blogspot.com/-FdV7iNqN34Q/Ux7munuFVtI/AAAAAAAAA1A/2zH0PBQQULI/s1600/s4.png)

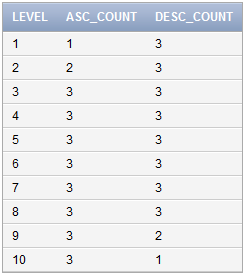
Помимо определения окна по диапазону (**RANGE**), также окна определяются и по количеству строк (**ROWS**). Для окон по строкам нет ограничений, присущих окнам по диапазону; данные могут быть любого типа и упорядочивать можно по любому количеству столбцов.  
Например, пусть нужно вычислить среднюю зарплату для сотрудника и пяти принятых на работу до него и после него. Запрос можно записать следующим образом  
  
**select ename, hiredate, sal,**  
**avg(sal) over (order by hiredate asc rows 5 preceding) avg\_5\_before,**  
**avg(sal) over (order by hiredate desc rows 5 preceding) avg\_5\_after**  
**from emp**  
**order by hiredate**

[](http://2.bp.blogspot.com/-o3MFQbfXbxg/Ux7neYA3M8I/AAAAAAAAA1I/xDide_MJQks/s1600/s5.png)

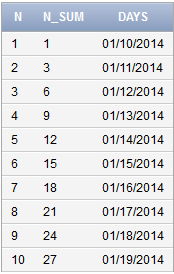
Зная как определяются окна (по диапазону или по количеству строк), рассмотрим как окончательно задаются окна. В простейшем случае, окно задается с помощью одной из трех следующих взаимоисключающих конструкций.  
  
***- UNBOUNDED PRECEDING.***  
Окно начинается с первой строки текущей группы и заканчивается текущей обрабатываемой строкой.  
  
***- CURRENT ROW.***  
Окно начинается (и заканчивается) текущей строкой.  
  
***- Числовое\_выражение PRECEDING.***  
Окно начинается со строки за *числовое\_выражение* строк до текущей, если оно задается по строкам, или со строки, меньшей по значению столбца, упомянутого в конструкции order by, не более чем на числовое выражение, если оно задается по диапазону.  
  
***- Числовое\_выражение FOLLOWING.***  
Окно заканчивается (или начинается) со строки, через *числовое\_выражение* строк после текущей, если оно задается по строкам, или со строки, большей  
по значению столбца, упомянутого в конструкции order by, не более чем на числовое\_выражение, если оно задается по диапазону.  
  
Стоит отметить, что окно **CURRENT ROW** в простейшем виде, вероятно, никогда не используется, поскольку ограничивает применение аналитической функции одной текущей строкой, а для этого аналитические функции не нужны. В более сложном случае для окна задается также конструкция **BETWEEN**. В ней **CURRENT ROW** можно указывать в качестве начальной или конечной строки окна. Начальную и конечную строку окна в конструкции **BETWEEN**можно задавать с использованием любой из перечисленных выше конструкций. Например, можно  задать  окно  так,  что  обрабатываемая  строка  не  будет  
последней, а окажется где-то в середине окна  
  
**select ename, hiredate,**  
**first\_value(ename) over (order by hiredate asc range between 100 preceding and 100 following),**  
**last\_value(ename) over (order by hiredate asc range between 100 preceding and 100 following)**  
**from emp**  
**order by hiredate asc**

[](http://2.bp.blogspot.com/-9FdHDv6Bvro/Ux7lALjusII/AAAAAAAAA0g/DxrS6g12Wqc/s1600/c3.png)

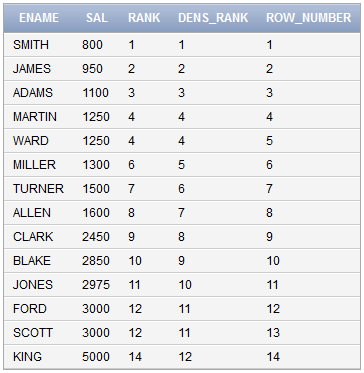
В данном запросе дополнительно использованы функции **first\_value()** и **last\_value()**, которые возвращают первое значений текущего окна и последнее значение также текущего окна, соответственно, в то время как диапазон окна ограничен слева текущая скользящая дата - 100 дней и справа к текущей скользящей дате + 100 дней, в этом и состоит смысл выражения **- between 100 preceding and 100 following**.  
  
**select  
  level,  
  count(\*) over (order by level asc rows 2 preceding) asc\_count,  
  count(\*) over (order by level desc rows 2 preceding) desc\_count  
from dual  
connect by level <= 10  
order by level**  
  
Окно rows 2 preceding, как видно из результата запроса, содержит от 1 до 3 строк (это определяется тем, как далеко текущая строка находится от начала группы). Для первой строки группы имеем значение 1 (предыдущих строк нет). Для следующей строки в группе таких строк 2. Наконец, для третьей и далее строк значение count(\*) остается постоянным, поскольку мы считаем только текущую строку и две предыдущие.

[](http://2.bp.blogspot.com/-O_sBv0IImaE/Ux7p_DxjKoI/AAAAAAAAA1c/5bd01ZKChqg/s1600/e2.png)

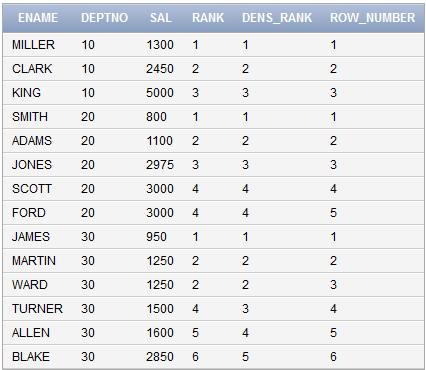
**select  
  n,  
  sum(n) over (order by days range 2 preceding) n\_sum,  
  days  
from (  
  select  
     level n,  
     ( to\_date('10.01.2014', 'dd.mm.yyyy') + (level - 1) ) days  
  from dual  
  connect by level <= 10  
)  
order by days**  
  
В данном случае идет суммирование в пределах окна, диапазон которого составляет скользящий день и 2 предыдущих дня

[](http://3.bp.blogspot.com/-9tulYDz6lfk/Ux7pqPY0rkI/AAAAAAAAA1U/8vcJlbokeLo/s1600/e1.png)

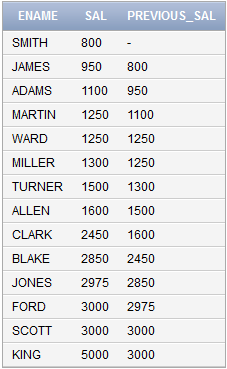
**select  
  ename,  
  sal,  
  rank() over (order by sal) rank,  
  dense\_rank() over (order by sal) dens\_rank,  
  row\_number() over (order by sal) row\_number  
from emp  
order by sal**  
  
Данный запрос демонстрирует работу ранжирующих функций rank(), dense\_rank() и row\_number() по окладам работников. Обратите внимание на поведение данных функций в строках с одинаковыми значениями окладов.

[](http://4.bp.blogspot.com/-3KFhSQZkoOE/Ux7u5losZHI/AAAAAAAAA1s/nhmPW_gthfA/s1600/e3.png)

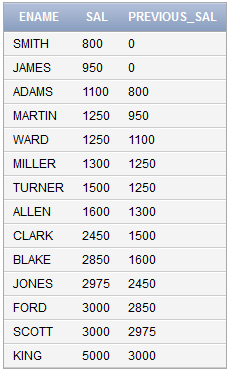
**select  
  ename,  
  deptno,  
  sal,  
  rank() over (partition by deptno order by sal) rank,  
  dense\_rank() over (partition by deptno order by sal) dens\_rank,  
  row\_number() over (partition by deptno order by sal) row\_number  
from emp  
order by deptno**  
  
Можно предыдущий запрос фрагментировать по отделам, то есть разбить на группы по отделам, в пределах которых будут работать аналитические функции, которые при выходе за пределы групп будут сбрасывать результаты. Результат представлен ниже

[](http://2.bp.blogspot.com/-uEfyspR1aD8/Ux7yXWAE6vI/AAAAAAAAA14/SO2d-6y0b7Q/s1600/e4.png)

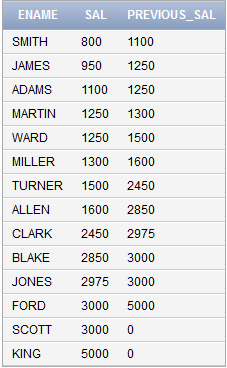
Рассмотрим пример применения функций lag() и lead(). Для того, чтобы можно было обращаться в текущей строке к предыдущим строкам, необходимо использовать функцию - lag(). Синтаксис ее таков  
  
**lag(*поле\_для\_обращения, смещение, значение\_для\_замещения\_null*)**  
**over (partition by *выражение* order by *выражение*)**  
  
поле\_для\_обращения - поле, по которому нужно просматривать значения;  
смещение - смещенная строка, с которой просматривается поле, по-умолчанию равно 1, если проставить 0, тогда будет просматриваться текущее поле;  
значение\_для\_замещения\_null - по-умолчанию равно null, в случае отсутствия значения в просматриваемом поле, возвращает данное значение. Здесь стоит отметить, что подставляемое значение должно быть того же типа, что и просматриваемое поле;  
для данной функции обязательно использование **order by**  
  
**with  
  main as (  
    select empno, ename, job, mgr, hiredate, sal, comm, deptno, rownum  
    from emp  
    order by sal  
  ),  
  numerated\_main as (  
    select empno, ename, job, mgr, hiredate, sal, comm, deptno, rownum  
    from main  
  )  
select  
  ename,  
  sal,  
  lag(sal) over (order by rownum) previous\_sal  
from numerated\_main**

[](http://4.bp.blogspot.com/-4ycUuUqD6IQ/Ux8JLw7Fo2I/AAAAAAAAA2I/IZBmrmGAw2U/s1600/e5.png)

Или вот так  
  
**with  
  main as (  
    select empno, ename, job, mgr, hiredate, sal, comm, deptno, rownum  
    from emp  
    order by sal  
  ),  
  numerated\_main as (  
    select empno, ename, job, mgr, hiredate, sal, comm, deptno, rownum  
    from main  
  )  
select  
  ename,  
  sal,  
  lag(sal, 2, 0) over (order by rownum) previous\_sal  
from numerated\_main**

[](http://3.bp.blogspot.com/-BcuRZFojjA4/Ux8MLUk7urI/AAAAAAAAA2U/DS05CcgpyVE/s1600/e6.png)

Логика и синтаксис работы функции lead() аналогичен предыдущей функции с одной лишь разницей: просмотр идет не назад, а вперед  
  
with  
  main as (  
    select empno, ename, job, mgr, hiredate, sal, comm, deptno, rownum  
    from emp  
    order by sal  
  ),  
  numerated\_main as (  
    select empno, ename, job, mgr, hiredate, sal, comm, deptno, rownum  
    from main  
  )  
select  
  ename,  
  sal,  
  lead(sal, 2, 0) over (order by rownum) previous\_sal  
from numerated\_main

[](http://2.bp.blogspot.com/-l2ijTsLgEYM/Ux8POIIT2cI/AAAAAAAAA2w/UAOridIdIHI/s1600/e7.png)

Ниже для ознакомления приведен неполный список аналитических функций. Для более полной информации обращайтесь к [документации](http://docs.oracle.com/cd/E11882_01/server.112/e26088/functions004.htm#SQLRF06174). Всем успехов.

|  |  |
| --- | --- |
| **Аналитическая функция** | **Описание** |
| **AVG([DISTINCT | ALL] выражение)** | Используется  для  вычисления  среднего значения выражения  в  пределах  группы  и окна.  Для  поиска среднего  после  удаления  дублирующихся  значений  можно указывать ключевое слово DISTINCT |
| **CORR(выражение, выражение)** | Выдает  коэффициент  корреляции  для  пары  выражений, возвращающих  числовые  значения. В  статистическом  смысле,  корреляция  —  это  степень связи  между  переменными.  Связь  между переменными означает,  что  значение  одной  переменной  можно  в определенной степени предсказать по значению другой. Коэффициент  корреляции  представляет степень корреляции  в  виде  числа  в  диапазоне  от  -1  (высокая обратная корреляция) до 1 (высокая корреляция). Значение 0 соответствует отсутствию корреляции |
| **COUNT([DISTINCT][\*] [выражение])** | Эта  функция  считает  строки  в  группах.  Если  указать  \* или  любую  константу,  кроме  NULL,  функция  COUNT  будет считать  все  строки.  Если  указать  выражение,  функция COUNT  будет  считать  строки,  для  которых  выражение имеет  значение  не  NULL.  Можно  задавать  модификатор DISTINCT,  чтобы  считать  строки  в  группах  после удаления дублирующихся строк |
| **COVAR\_POP(выражение, выражение)** | Возвращает  ковариацию  генеральной  совокупности (population  covariance)  пары  выражений  с  числовыми значениями |
| **COVAR\_SAMP(выражение, выражение)** | Возвращает  выборочную  ковариацию  (sample  covariance) пары выражений с числовыми значениями |
| **CUME\_DIST** | Вычисляет относительную позицию строки в группе. Функция CUME\_DIST всегда возвращает число большее  0 и меньше или равное  1. Это число представляет "позицию" строки  в  группе  из  N  арок.  В  группе  из  трех  строк, например,  возвращаются  следующие  значения кумулятивного распределения: 1/3, 2/3 и 3/3 |
| **DENSE\_RANK** | Эта  функция  вычисляет  относительный  ранг  каждой возвращаемой  запросом  строки  по  отношению  к  другим строкам,  основываясь  на  значениях  выражений  в конструкции  ORDER  BY.  Данные  в  группе  сортируются  в соответствии  с  конструкцией  ORDER  BY,  а  затем  каждой строке  поочередно  присваивается  числовой  ранг, начиная с 1. Ранг увеличивается при каждом изменении значений выражений,  входящих  в  конструкцию  ORDER  BY.  Строки  с одинаковыми  значениями  получают  один  и  тот  же  ранг (при  этом  сравнении  значения  NULL  считаются одинаковыми).  Возвращаемый  этой  функцией  "плотный" ранг дает ранговые значения без промежутков. Сравните с представленной далее функцией RANK |
| **FIRST\_VALUE** | Возвращает первое значение в группе |
| **LAG(выражение,<смещение>,** **<стандартное** **значение>)** | Функция  LAG  дает  доступ  к  другим  строкам результирующего  множества,  избавляя  от  необходимости выполнять  самосоединения.  Она  позволяет  работать  с курсором  как  с  массивом.  Можно  ссылаться  на  строки, предшествующие  текущей  строке  в  группе.  О  том,  как обращаться  к  следующим  строкам  в  группе,  см.  в описании функции LEAD. Смещение  -  это  положительное  целое  число  со стандартным значением 1 (предыдущая строка). Стандартное  значение  возвращается,  если  индекс выходит  за  пределы  окна  (для  первой  строки  группы будет возвращено стандартное значение) |
| **LAST\_VALUE** | Возвращает последнее значение в группе |
| **LEAD(выpaжeниe,<смещение>,<стандартное** **значение>)** | Функция  LEAD  противоположна функции  LAG. Если функция LAG  дает  доступ  к  предшествующим  строкам  группы,  то функция  LEAD  позволяет  обращаться  к  строкам, следующим за текущей. Смещение  —  это  положительное  целое  число  со стандартным  значением  1  (следующая  строка). Стандартное  значение  возвращается,  если  индекс выходит  за  пределы  окна  (для  последней  строки  группы будет возвращено стандартное значение) |
| **МАХ(выражение)** | Находит  максимальное  значение  выражения  в  пределах окна в группе |
| **МIN(выражение)** | Находит  минимальное  значение  выражения  в  пределах окна в группе |
| **NTILE(выражение)** | Делит группу на фрагменты по значению выражения. Например,  если  выражение  =  4,  то  каждой  строке  в группе присваивается число от 1 до 4 в соответствии с фрагментом,  в  которую  она  попадает.  Если  в  группе  20 строк,  первые  5  получат  значение  1,  следующие  5  — значение  2  и  т.д.  Если  количество  строк  в  группе  не делится  на  значение  выражения  без  остатка,  строки распределяются  так,  что  ни  в  одном  фрагменте количество  строк  не  превосходит  минимальное количество в других фрагментах более чем на 1, причем дополнительные  строки  будут  в  группах  с  меньшими номера фрагмента. Например,  если  снова  выражение  =  4,  а  количество строк  =  21,  в  первом  фрагменте  будет  6  строк,  во втором и последующих - 5 |
| **PERCENT RANK** | Аналогична  функции  CUME\_DIST  (кумулятивное распределение).  Вычисляет  ранг  строки  в  группе  минус 1,  деленный  на  количество  обрабатываемых  строк  минус 1. Эта функция всегда возвращает значения в диапазоне от 0 до 1 включительно |
| **RANK** | Эта  функция  вычисляет  относительный  ранг  каждой строки,  возвращаемой  запросом,  на  основе  значений выражений,  входящих  в  конструкцию  ORDER  BY.  Данные  в группе  сортируются  в  соответствии  с  конструкцией ORDER  BY,  а  затем  каждой  строке  поочередно присваивается  числовой  ранг,  начиная  с  1.  Строки  с одинаковыми  значениями  выражений,  входящих  в конструкцию  ORDER  BY,  получают  одинаковый  ранг,  но если  две  строки  получат  одинаковый  ранг,  следующее значение ранга пропускается. Если две строки получили ранг  1,  строки с рангом  2  не будет;  следующая строка в  группе  получит  ранг  3.  В  этом  отличие  от  функции DENSE\_RANK, которая не пропускает значений |
| **RATIO\_TO\_REPORT(выражение)** | Эта  функция  вычисляет  значение  выражение  / (SUM(выражение)) по строкам группы. Это дает процент,  который составляет значение текущей строки по отношению к SUM(выражение) |
| **REGR\_xxxxxxx(выражение,выражение)** | Эти  функции  линейной  регрессии  применяют  стандартную линейную  регрессию  по  методу  наименьших  квадратов  к паре  выражений.  Предлагается  9  различных  функций регрессии |
| **ROW\_NUMBER** | Возвращает  смещение  строки  по  отношению  к  началу упорядоченной  группы.  Может  использоваться  для последовательной  нумерации  строк,  упорядоченных  по определенным критериям |
| **STDDEV(выражение)** | Вычисляет  стандартное  (среднеквадратичное)  отклонение (standard  deviation)  текущей  строки  по  отношению  к группе |
| **STDDEV\_POP(выражение)** | Эта  функция  вычисляет  стандартное  отклонение генеральной  совокупности  (population  standard deviation)  и  возвращает  квадратный  корень  из дисперсии  генеральной  совокупности  (population variance).  Она  возвращает  значение,  совпадающее  с квадратным корнем из результата функции VAR\_POP |
| **STDDEV\_SAMP(выражение)** | Эта  функция  вычисляет  накопленное  стандартное отклонение  выборки  (cumulative  sample  standard deviation)  и  возвращает  квадратный  корень  выборочной дисперсии  (sample variance). Она возвращает значение, совпадающее  с квадратным корнем из результата функции VAR\_SAMP |
| **SUM(выражение)** | Вычисляет общую сумму значений  выражения для группы |
| **VAR\_POP(выражение)** | Эта  функция  возвращает  дисперсию  генеральной совокупности  для  набора  числовых  значений  (значения NULL игнорируются) |
| **VAR\_SAMP(выражение)** | Эта  функция  возвращает  выборочную  дисперсию  длянабора  числовых  значений  (значения  NULL игнорируются) |
| **VARIANCE(выражение)** | Возвращает  дисперсию  для  выражения.  Сервер  Oracle вычисляет дисперсию как: - 0, если количество строк в группе = 1;            -VAR\_SAMP, если количество строк в группе больше 1 |