START

1. Sql基础篇

数据库操作语言：DML select insert update delete merge

数据库定义语言：DDL create alter drop truncate

事务控制语言: TCL (transaction control language) commit rollback savepoint

数据控制语言： DCL（data control）grant revoke

1. 数据类型

Char varchar2 number date timestamp clob blob

1. 常用转换 date char number 三者类型经常转换

select TO\_DATE('2015-08-19','YYYY-MM-DD') AS A\_DAY FROM DUAL;

SELECT TO\_CHAR(SYSDATE,'YYYY-MM-DD') AS TODAY FROM DUAL;

SELECT TO\_TIMESTAMP('2015-08-19 17:40:32.11','YYYY-MM-DD HH24:MI:SS.FF') AS A\_DAY FROM DUAL;

SELECT TO\_NUMBER(REPLACE(TO\_CHAR(SYSDATE,'YYYY-MM-DD'),'-')) FROM DUAL;

SELECT 1+1 AS RESULT FROM DUAL;

1. 常用sql语句

select deptno,decode(deptno,10,'部门1',20,'部门2',30,'部门3') from EMP;

select deptno,case when deptno = 10 then '部门1' when deptno = 20 then '部门2' else '部门3' END FROM EMP;

1. 分组 组函数：sum min max avg count

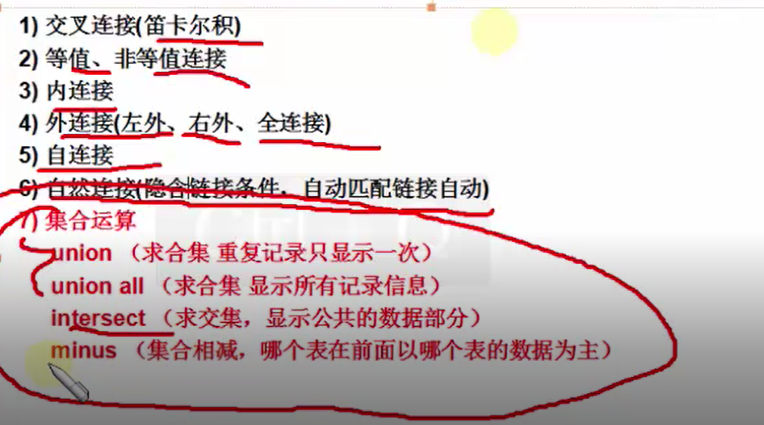
Group by … having 根据一个或者多个列对结果进行分组

查询每个部门的平均薪水之后显示部门平均薪水大于2000的部门编号和其平均薪水

Select DEPTNO,AVG(SAL) AS TEST FROM EMP GROUP BY DEPTNO HAVING AVG(SAL) > 2000

SELECT DEPTNO,SUM(SAL),AVG(SAL) FROM EMP group by DEPTNO

1. 集合查询



子查询

非关联子查询

主查询和子查询是相对独立的，

select A.ENAME,A.SAL from EMP A where a.DEPTNO = (select b.DEPTNO from DEPT b where b.LOC = 'NEW YORK')

关联子查询

子查询不构成一个完整的sql语句

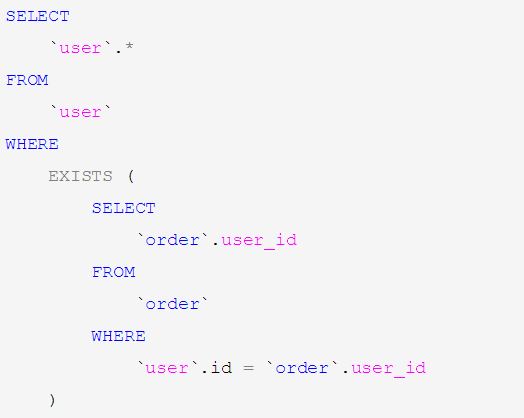
SELECT A.DEPTNO,(SELECT B.LOC FROM DEPT B WHERE B.DEPTNO = A.DEPTNO) FROM EMP A;

**IN 和 EXISTS**

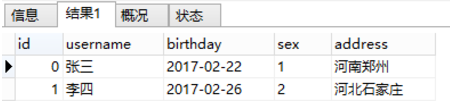
IN 在查询时首先查询内表，然后内表和外表做一次笛卡尔积，然后按照条件筛选。所以相对内表比较小的时候，in的速度比较快

EXISTS指定一个子查询，检测行的存在。遍历循环外表，然后看外表中记录又没有和内表的数据一样的。匹配上就将结果放到结果集中

具体例子：



这条sql语句的执行结果和上面的in的执行结果是一样的。

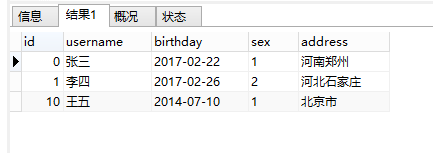


　　　　但是，不一样的是它们的执行流程完全不一样：

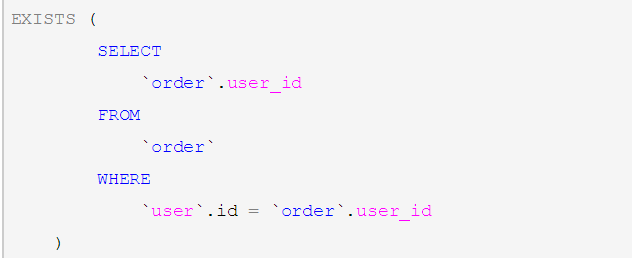
　　　　使用exists关键字进行查询的时候，首先，我们先查询的不是子查询的内容，而是查我们的主查询的表，也就是说，我们先执行的sql语句是：

SELECT `user`.\* FROM `user`

得到的结果如下：



然后，根据表的每一条记录，执行以下语句，依次去判断where后面的条件是否成立：



区别和应用场景：

in 和 exists的区别: 如果子查询得出的结果集记录较少，主查询中的表较大且又有索引时应该用in, 反之如果外层的主查询记录较少，子查询中的表大，又有索引时使用exists。其实我们区分in和exists主要是造成了驱动顺序的改变(这是性能变化的关键)，如果是exists，那么以外层表为驱动表，先被访问，如果是IN，那么先执行子查询，所以我们会以驱动表的快速返回为目标，那么就会考虑到索引及结果集的关系了 ，另外IN时不对NULL进行处理。

in 是把外表和内表作hash 连接，而exists是对外表作loop循环，每次loop循环再对内表进行查询。一直以来认为exists比in效率高的说法是不准确的。

EXISTS是先执行外表的sql语句，得到一个结果，然后再去匹配where条件是否成立,外表较小，且内表较大，且又有索引的情况下建议使用EXISTS。

IN是先执行内表的sql语句，得到一个表，然后和外表结果做一个笛卡儿积，得到一个表，然后按照条件进行筛选。所以如果子查询结果记录较少，主查询中表较大且又有索引的情况下用in

查询大小领导的员工

SELECT \* FROM EMP A WHERE EXISTS (SELECT 1 FROM EMP B WHERE A.EMPNO = B.MGR)

SELECT 1 代表一个值

查询哪个部门不存在员工的部门信息

SELECT \* FROM DEPT D WHERE NOT EXISTS (SELECT 1 FROM EMP E WHERE E.DEPTNO = D.DEPTNO)

DISTINCT 关键字

Select name from student where name in (‘zhang’,’wang’,’zhao’)

Select name from student where name exist (‘zhang’,’wang’,’zhao’)

Select name from student where name = ‘zhang’ or name = ‘wang’ or name = ‘zhao’

1. SQL基础续

BETWEEN AND / ROWNUM < 3

1. SQL高级

Create table myemp AS SELECT \* FROM EMP WHERE DETPNO =10

INSERT INTO TABLE1 (F1,F2,…) SELECT \* FROM TABLE2

1. MERGE INTO TABLE1

USING 与表A产生关联的字段

ON 和表A产生关联

Merge into products p using (select \* from new\_products) np on (p.product\_name = np.product\_name)

When matched then

Update set p.product\_name = np.product\_name

When not matched then

Insert (p. product\_id,p. product\_name,p.category)values (np.product\_id, np.product\_name, np.category)

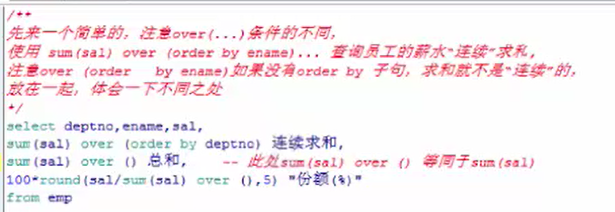
1. 递归查询,分层查询； 父节点 = 子节点 （向上查询） 反之向下查询

select \* from EMP

START WITH EMPNO = 7369 OR EMPNO = 7968

CONNECT BY PRIOR MGR = EMPNO

1. Over 字句



按照DEPTNO 进行分区（PARTITION）然后进行求和

SELECT DEPTNO ,ENAME,SAL,

SUM(SAL) OVER(PARTITION BY DEPTNO ORDER BY ENAME) 部门连续求和,

SUM(SAL) OVER(PARTITION BY DEPTNO) 部门总合,

100 \* ROUND(SAL/SUM(SAL) OVER (PARTITION BY DEPTNO),4) 部门份额,

100 \* ROUND(SAL/SUM(SAL) OVER (),4) 总份额

from emp

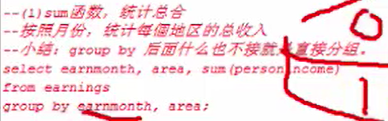
select DEPTNO ,ename,sal,

sum(SAL) OVER (PARTITION BY DEPTNO ORDER BY ENAME) DEPT\_SUM,

SUM(SAL) OVER (ORDER BY DEPTNO ,ENAME) SUM --连续的求总和

FROM EMP;

1. Sum 语句

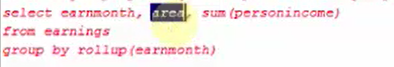


Select month,area, sum(p) from earnings group by month,area;



1. Group by字句下图中·的earmonth,area都要放到group by里面，group by一般和聚合函数在一起使用才有意义，使用group by的两个要素：

   (1) 出现在select后面的字段 要么是是聚合函数中的,要么就是group by 中的.  
   (2) 要筛选结果 可以先使用where 再用group by 或者先用group by 再用having



1. Rank / dense\_rank / row\_number
2. Select earnmonth 月份,area 地区,sname 打工,persionincome 收入,

Sum(persionincome) over (partition by sname order by persionincome asc) 总收入

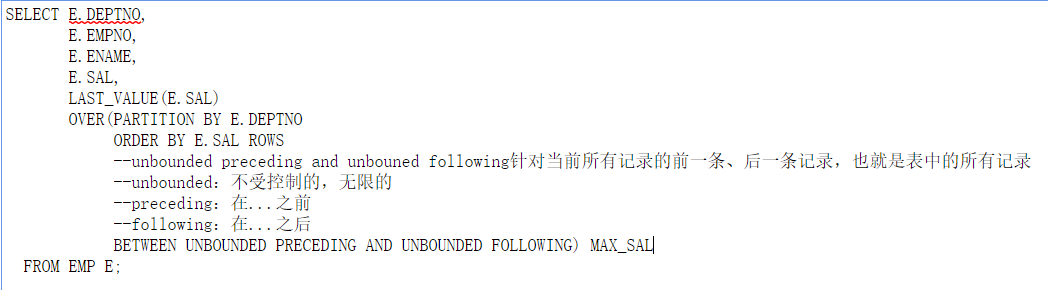
From earnings;

通过sname分区，然后根据persionincome分组

分析函数专题：

1. **分析函数是什么？**  
   分析函数是[**Oracle**](http://lib.csdn.net/base/oracle)专门用于解决复杂报表统计需求的功能强大的函数，它可以在数据中进行分组然后计算基于组的某种统计值，并且每一组的每一行都可以返回一个统计值。
2. **分析函数和聚合函数的不同之处是什么？**  
   普通的聚合函数用group by分组，每个分组返回一个统计值，而分析函数采用partition by分组，并且每组每行都可以返回一个统计值。
3. **分析函数的形式**  
   分析函数带有一个**开窗函数over()，包含三个分析子句:分组(partition by), 排序(order by), 窗口(rows) ，**他们的使用形式如下：over(partition by xxx order by yyy rows between zzz)。  
   注：窗口子句在这里我只说rows方式的窗口,range方式和滑动窗口也不提

--显示各部门员工的工资，并附带显示该部分的最高工资。



按照deptno分组，然后计算每组值的总和

SELECT EMPNO,

ENAME,

DEPTNO,

SAL,

SUM(SAL) OVER(PARTITION BY DEPTNO ORDER BY ENAME) max\_sal

FROM SCOTT.EMP;

示例目的：对各部门进行分组，并附带显示第一行至当前行的汇总

SELECT EMPNO,

ENAME,

DEPTNO,

SAL,

--注意ROWS BETWEEN unbounded preceding AND current row 是指第一行至当前行的汇总

SUM(SAL) OVER(PARTITION BY DEPTNO

ORDER BY ENAME

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) max\_sal

FROM SCOTT.EMP;

分析函数（以及与其配合的开窗函数over（））是在整个sql查询结束后(sql语句中的order by的执行比较特殊)再进行的操作, 也就是说sql语句中的order by也会影响分析函数的执行结果：

a) 两者一致：如果sql语句中的order by满足与分析函数配合的开窗函数over（）分析时要求的排序，即sql语句中的order by子句里的内容和开窗函数over（）中的order by子句里的内容一样，

那么sql语句中的排序将先执行,分析函数在分析时就不必再排序；  
b) 两者不一致：如果sql语句中的order by不满足与分析函数配合的开窗函数over（）分析时要求的排序，即sql语句中的order by子句里的内容和开窗函数over（）中的order by子句里的内容不一样，

那么sql语句中的排序将最后在分析函数分析结束后执行排序。

摘自：http://blog.csdn.net/haiross/article/details/15336313