# 单行函数

## 字符函数

### Upper

SELECT Upper ('abcde') FROM dual ;

SELECT \* FROM emp WHERE ename=UPPER('smith') ;

### Lower

SELECT lower('ABCDE') FROM dual ;

### Initcap

函数INITCAP()是将每个单词的第一个字母大写,其它字母变为小写返回.

Select initcap(ename) from emp;

### Concat

Select concat(‘a’,’b’) from dual;

Select ‘a’|| ‘b’from dual;

### Substr

Select substr(„abcde‟,length(„abcde‟)-2) from dual;

Select substr(„abcde‟,-3,3) from dual;

### Length

select length('abcde') from dual;

### Replace

select replace ('abcdeffddses','s','j') from dual;

### Instr

Instr(a,b) 找出字符串b在字符串a中出现的下标位置

Select instr('Hello World','or') from dual; 8 indexof

### Lpad

lpad('Smith',10,'\*') 左侧填充 lpad() \*\*\*\*\*Smith

select lpad('Smith',10,'\*') from dual;

### Rpad

rpad('Smith',10,'\*') 右侧填充 rpad()Smith\*\*\*\*\*

select rpad('Smith',10,'\*') from dual;

### Trim

trim(' Mr Smith ') 过滤首尾空格 trim() Mr Smith

select trim(' Mr Smith ') from dual;

## 数值函数

### Round

如果 **num\_digits** 大于 0（零），则将数字四舍五入到指定的小数位。

如果 **num\_digits** 等于 0，则将数字四舍五入到最接近的整数。

如果 **num\_digits** 小于 0，则在小数点左侧前几位进行四舍五入。

select round(412,-2) from dual;

select round(412.313,2) from dual;

若要始终进行向上舍入（远离 0），请使用 **ROUNDUP** 函数。

若要始终进行向下舍入（朝向 0），请使用 **ROUNDDOWN** 函数。

若要将某个数字四舍五入为指定的倍数（例如，四舍五入为最接近的 0.5 倍），请使用 **MROUND** 函数。

### Mod

mod函数是一个求余函数，其格式为： mod(nExp1,nExp2)，即是两个数值表达式作除法运算后的余数

select mod(-5,-2) from dual;

select mod(5,2) from dual;

### Trunc

用法1：TRUNC函数返回处理后的数值，其工作机制与ROUND函数极为类似，只是该函数不对指定小数前或后的部分做相应舍入选择处理，而统统截去。

select trunc(412.13,-2) from dual;

用法2：

下面是该函数的使用情况：

trunc(sysdate,'yyyy') --返回当年第一天.

trunc(sysdate,'mm') --返回当月第一天.

trunc(sysdate,'d') --返回当前星期的第一天

## 日期函数

### Months\_between()

months\_between函数返回两个日期之间的月份数。如果两个日期月份内天数相同，或者都是某个月的最后一天，返回一个整数，否则，返回数值带小数，以每天1/31月来计算月中剩余天数。

select months\_between(sysdate,to\_date('2016-05-16','yyyy-MM-dd')) from dual;

### Add\_months()

ADD\_MONTHS是运算函数，函数将一个日期上加上一指定的月份数，所以，日期中的日是不变的

select add\_months(sysdate,1) from dual;

### Next\_day()

(ie: SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY)

在Oracle是提供了next\_day求指定日期的下一个日期.

select next\_day(sysdate,'星期一') from dual;

### Last\_day

last\_day是个函数，返回某个时间值的当月最后一天

select last\_day(sysdate) from dual;

### 日期加法

select sysdate,add\_months(sysdate,12) from dual;         --加1年

select sysdate,add\_months(sysdate,1) from dual;         --加1月

select sysdate,to\_char(sysdate+7,'yyyy-mm-dd HH24:MI:SS') from dual; --加1星期

select sysdate,to\_char(sysdate+1,'yyyy-mm-dd HH24:MI:SS') from dual; --加1天

select sysdate,to\_char(sysdate+1/24,'yyyy-mm-dd HH24:MI:SS') from dual;--加1小时

select sysdate,to\_char(sysdate+1/24/60,'yyyy-mm-dd HH24:MI:SS') from dual;--加1分钟

select sysdate,to\_char(sysdate+1/24/60/60,'yyyy-mm-dd HH24:MI:SS') from dual; --加1秒

### 日期减法

select sysdate,add\_months(sysdate,-12) from dual;         --减1年

select sysdate,add\_months(sysdate,-1) from dual;         --减1月

select sysdate,to\_char(sysdate-7,'yyyy-mm-dd HH24:MI:SS') from dual; --减1星期select sysdate,to\_char(sysdate-1,'yyyy-mm-dd HH24:MI:SS') from dual;  --减1天

select sysdate,to\_char(sysdate-1/24,'yyyy-mm-dd HH24:MI:SS') from dual; --减1小时

select sysdate,to\_char(sysdate-1/24/60,'yyyy-mm-dd HH24:MI:SS') from dual; --减1分钟

select sysdate,to\_char(sysdate-1/24/60/60,'yyyy-mm-dd HH24:MI:SS') from dual; --减1秒

## 转换函数

### To\_char

参考文档：

<http://baike.baidu.com/view/6011617.htm>

select to\_char(sysdate,'yyyy') from dual;

select to\_char(sysdate,'fmyyyy-mm-dd') from dual;

select to\_char(sal,'L999,999,999') from emp;

select to\_char(sysdate,’D’) from dual;//返回星期

### To\_number

select to\_number('13')+to\_number('14') from dual;

### To\_date

Select to\_date(„20090210‟,‟yyyyMMdd‟) from dual;

## 通用函数

### NVL()函数

[NVL](http://baike.baidu.com/view/1091825.htm)(E1, E2)的功能为：如果E1为NULL，则函数返回E2，否则返回E1本身

select nvl(comm,0) from emp;

### NVL2()函数

[NVL](http://baike.baidu.com/view/1091825.htm)2(E1, E2, E3)的功能为：如果E1为NULL，则函数返回E3，若E1不为null，则返回E2。

### NULLIF()函数

如果两个指定的[表达式](http://baike.baidu.com/view/420676.htm)相等，则返回[空值](http://baike.baidu.com/view/7258064.htm)。

语法NULLIF **(***expression1***,** *expression2***)**

### COALESCE()函数

语法为COALESCE(表达式1,表达式2,...,表达式n)，n>=2,此表达式的功能为返回第一个不为空的表达式，如果都为空则返回空值。

依次考察各参数表达式，遇到非 null 值即停止并返 回该值。 select empno, ename, sal, comm, coalesce(sal+comm, sal, 0)总收入 from emp;

### CASE 表达式

select empno, ename, sal, case deptno when 10 then '财务部' when 20 then '研发部' when 30 then '销售部' else '未知部门' end 部门 from emp;

### DECODE()函数

和 case 表达式类似，decode()函数也用于实现多路分支结构

select empno, ename, sal, decode(deptno, 10, '财务部', 20, '研发部', 30, '销售部', '未知部门') 部门 from emp;

# 分组函数

## COUNT，Avg，max，min，sum

## Group by

where having

group by 有一个原则,就是 select 后面的所有列中,没有使用聚合函数的列,必须出现在 group by 后面（重要）

# 多表查询

## 笛卡尔集(Cross Join)

Select \* from emp,dept;

## 等值连接(Equijoin)(Natural join..on)

select empno, ename, sal, emp.deptno, dname from emp, dept where emp.deptno = dept.deptno;

## 非等值连接(Non-Equijoin)

select ename,empno,grade from emp,salgrade where sal between losal and hisal;

## 自连接(Self join)

**select e.empno,e.ename,m.empno,m.ename from emp e,emp m where e.mgr = m.empno; select e.empno,e.ename,m.empno,m.ename from emp e,emp m where m.mgr = e.empno;**

## 左外联接（Left Outer Join ）

**select s.sid,s.sname,s1.sid,s1.sname from student s,student1 s1 where s.sid=s1.sid(+);**

**select empno,ename,dname from emp left outer join dept on emp.deptno = dept.deptno;**

## 右外联接（Right Outer Join）

**select s.sid,s.sname,s1.sid,s1.sname from student s,student1 s1 where s.sid(+)=s1.sid;**

**select empno,ename,dname from emp right outer join dept on emp.deptno = dept.deptno;**

## 满外联接（Full Outer Join）

**select empno,ename,dname from emp full outer join dept on emp.deptno = dept.deptno;**

# 集合操作

## UNION

**并集，所有的内容都查询，重复的显示一次**

## UNION ALL

**并集，所有的内容都显示，包括重复的**

## INTERSECT

交集：

**只显示重复的**

## MINUS

**差集：**

**只显示对方没有的（跟顺序是有关系的） 首先建立一张只包含 20 部门员工信息的表： CREATE TABLE emp20 AS SELECT \* FROM emp WHERE deptno=20 ;**

**1、 验证 UNION 及 UNION ALL UNION：SELECT \* FROM emp UNION SELECT \* FROM emp20 ; 使用此语句重复的内容不再显示了 UNION ALL：SELECT \* FROM emp UNION ALL SELECT \* FROM emp20 ; 重复的内容依然显示**

**2、 验证 INTERSECT**

**SELECT \* FROM emp INTERSECT SELECT \* FROM emp20 ;**

**只显示了两个表中彼此重复的记录。**

**3、 MINUS**

**返回差异的记录**

**SELECT \* FROM emp MINUS SELECT \* FROM emp20 ;**

**只显示了两张表中的不同记录**

**满链接也可以用以下的方式来表示：**

**select t1.id,t2.id from table1 t1,table t2 where t1.id=t2.id(+) union select t1.id,t2.id from table1 t1,table t2 where t1.id(+)=t2.id**

# 子查询

## 单行子查询

**select \* from emp where sal > (select sal from emp where empno = 7566);**

## 子查询空值/多值问题

**如果子查询未返回任何行，则主查询也不会返回任何结果**

**(空值)select \* from emp where sal > (select sal from emp where empno = 8888);**

**如果子查询返回单行结果，则为单行子查询，可以在主查 询中对其使用相应的单行记录比较运算符**

**(正常)select \* from emp where sal > (select sal from emp where empno = 7566);**

**如果子查询返回多行结果，则为多行子查询，此时不允许 对其使用单行记录比较运算符**

**(多值)select \* from emp where sal > (select avg(sal) from emp group by deptno);//非法**

## 多行子查询

**select \* from emp where sal > any(select avg(sal) from emp group by deptno);**

**select \* from emp where sal > all(select avg(sal) from emp group by deptno);**

**select \* from emp where job in (select job from emp where ename = 'MARTIN' or ename = 'SMITH');**

## TopN 查询

**select \* from emp where rownum=1 or rownum=2;**

## 分页查询

**select \* from (select rownum no,e.\* from (select \* from emp order by sal desc) e where rownum<=5 ) where no>=3;**

**select \* from (select rownum no,e.\* from (select \* from emp order by sal desc) e) where no>=3 and no<=5;**

## exists

**EXISTS 的执行流程**

**select \* from t1 where exists ( select null from t2 where y = x )**

**可以理解为:**

**for x in ( select \* from t1 )**

**loop**

**if ( exists ( select null from t2 where y = x.x )**

**then**

**OUTPUT THE RECORD**

**end if**

**end loop**

## 对于 in 和 exists 的性能区别:

**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 是不对 NULL 进行处理**

**如：**

**select 1 from dual where null in (0,1,2,null)**

**为空**

# 高级查询

## 随机返回 5 条记录

Select \* from (select ename,job from emp order by dbms\_random.value()) where rownum<=5

## 处理空值排序

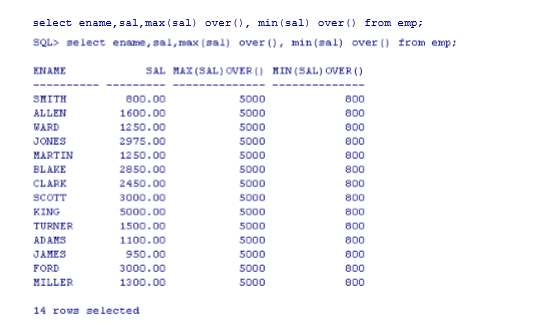
**select \* from emp order by comm desc nulls last(first);**

## 查询跳过表中的偶数行

**select ename from (select row\_number() over (order by ename) rn,ename from emp) x where mod(rn,2)=1;**

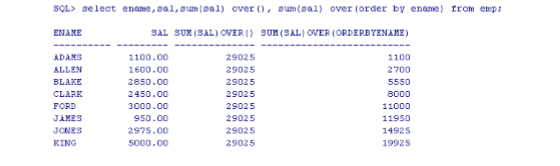
## 查询员工信息与其中工资最高最低员工

**select ename,sal,max(sal) over(), min(sal) over() from emp;**



## 连续求和

**select ename,sal,sum(sal) over(), sum(sal) over(order by ename) from emp; sum(sal) over(order by ename)指的是连续求和.是以 ename 来排序的。若有两个这 样的窗口函数，以后面的排序为主。**



## 分部门连续求和

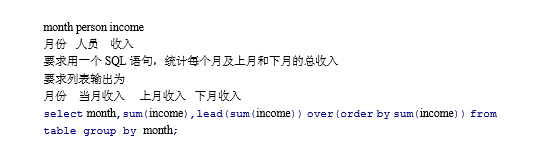
**select deptno,sal ,sum(sal) over (partition by deptno order by ename) as s from emp;分部门连续求和**

**sum(sal) over (partition by deptno) 分部门求和**



## 得到当前行上一行或者下一行的数据

select ename,sal,lead(sal) over(order by sal) aaa ,lag(sal) over(order by sal) bbb from emp;



# 数据字典

## 查询某用户下所有表

select table\_name from all\_tables where owner='SCOTT';

## 查询 EMP 表中所有字段（列）

select \* from all\_tab\_columns where table\_name='EMP';

## 列出表的索引列

select \* from sys.all\_ind\_columns where table\_name='EMP'; select \* from sys.all\_ind\_columns where upper(table\_name)='CAREUSERHAM';

## 列出表中约束

select \* from all\_constraints where table\_name='EMP';

## 在 oracle 中描述数据字典视图

select table\_name ,comments from dictionary where table\_name like '%TABLE%';

# DDL改变表结构操作

## 创建表

Create table student( Sid number(10), Sname varchar2(10) ) tablespace tt;

以上 tablespace 不是必须的。默认不写，则创建在登录的用户所在的表空间上

## 使用子查询创建表

create table myemp as select \* from emp;

create table myemp as select \* from emp where deptno=10;

create table myemp as select \* from emp 1=2;

## 添加字段

Alter table student add age number(5);

## 修改字段

Alter table student modify age number(10);

alter table table2 rename column result to result2;

## 删除字段

Alter table student drop column age;

## 清空表数据

Truncate table student;

正常情况下删除数据，如果发现删除错了，则可以通过 rollback 回滚。如果使用了截断表， 则表示所有的数据不可恢复了.所以速度很快（更详细的说明可查看 Oracle 体系结构）

## 删除表

Drop table student;

## 重命名表

Rename student to student1;

# DML(改变数据结构）

## insert 语句

表间数据拷贝 insert into dept1(id, name) select deptno, dname from dept;

## update 语句

将编号为 7779 用户的工作换成编号为 7566 的雇员的工作和所属上级。

UPDATE myemp SET(job,mgr) = (SELECT job,mgr FROM myemp WHERE empno=7566) WHERE empno=7779 ;

如果子查询中返回的是空，则目标字段也更新成 NULL.

## delete 语句

Delete from emp;

## merge 语句

create table test1(eid number(10), name varchar2(20),birth date,salary number(8,2));

insert into test1 values (1001, '张三', '20-5 月-70', 2300);

insert into test1 values (1002, '李四', '16-4 月-73', 6600);

select \* from test1;

create table test2(eid number(10), name varchar2(20),birth date,salary number(8,2));

select \* from test2;

merge into test2

using test1 on(test1.eid = test2.eid )

when matched then

update set name = test1.name, birth = test1.birth, salary = test1.salary

when not matched then

insert (eid, name, birth, salary) values(test1.eid, test1.name, test1.birth, test1.salary);

select \* from test2;

# 约束

## NOT NULL：非空约束

CREATE TABLE person ( pid NUMBER , name VARCHAR(30) NOT NULL ) ; -- 插入数据 INSERT

INTO person(pid,name) VALUES (11,'张三'); -- 错误的数据，会受到约束限制，无法插入

## PRIMARY KEY：主键约束

不能重复，不能为空

例如：身份证号不能为空。

现在假设 pid 字段不能为空，且不能重复。

DROP TABLE person ;

CREATE TABLE person ( pid NUMBER PRIMARY KEY , name VARCHAR(30) NOT NULL ) ;

-- 插入数据 INSERT INTO person(pid,name) VALUES (11,'张三');

-- 主键重复了 INSERT INTO person(pid,name) VALUES (11,'李四');

## UNIQUE：唯一约束，值不能重复（空值除外）

人员中有电话号码，电话号码不能重复。

DROP TABLE person ;

CREATE TABLE person ( pid NUMBER PRIMARY KEY NOT NULL , name VARCHAR(30) NOT NULL , tel VARCHAR(50) UNIQUE ) ;

-- 插入数据 INSERT INTO person(pid,name,tel) VALUES (11,'张三','1234567');

-- 电话重复了 INSERT INTO person(pid,name,tel) VALUES (12,'李四','1234567');

## CHECK：条件约束，插入的数据必须满足某些条件

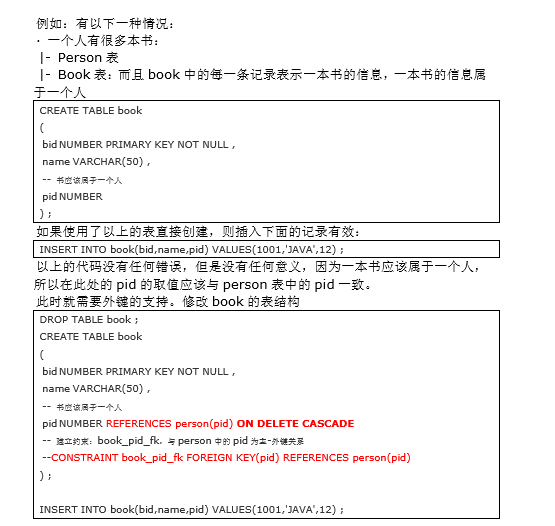
DROP TABLE person ;

CREATE TABLE person ( pid NUMBER PRIMARY KEY NOT NULL , name VARCHAR(30) NOT NULL , tel VARCHAR(50) NOT NULL UNIQUE , age NUMBER CHECK(age BETWEEN 0 AND 150) ) ;

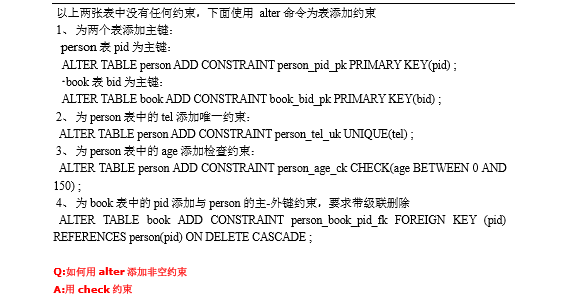
-- 插入数据 INSERT INTO person(pid,name,tel,age) VALUES (11,'张三','1234567',30);

-- 年龄的输入错误 INSERT INTO person(pid,name,tel,age) VALUES (12,'李四','2345678',-100);

## Foreign Key：外键



## 添加表后添加约束



## 删除约束

ALTER TABLE book DROP CONSTRAINT person\_book\_pid\_fk ;

alter table student drop unique(tel);

## 启用约束

ALTER TABLE book enable CONSTRAINT person\_book\_pid\_fk ;

## 禁用约束

ALTER TABLE book disable CONSTRAINT person\_book\_pid\_fk ;

# 视图

视图：是一个封装了各种复杂查询的语句，就称为视图。

## 创建视图

CREATE VIEW 视图名字(字段) AS 子查询

建立一个只包含 20 部门雇员信息的视图（雇员的编号、姓名、工资）

CREATE VIEW empv20 (empno,ename,sal)

AS

SELECT empno,ename,sal FROM emp WHERE deptno=20 ;

例如：将之前的一个复杂语句包装成视图 显示部门内最低工资比 20 部门最低工资要高的部门的编号及部门内最低工资：

SELECT deptno,MIN(sal) FROM emp

GROUP BY deptno

HAVING

MIN(sal)>(SELECT MIN(sal)

FROM emp WHERE deptno=20) ;

此时就可以将上面的复杂查询语句建立一张视图，之后查询视图即可。

## 高级视图

如果要创建一个同名的视图，则必须先将之前的视图删除掉，再进行创建：

DROP VIEW empv20 ;

有些时候如果先删除再创建操作会比较麻烦，所以有时候最好的方式：如果视图存在则先 自动删除，之后自动创建。

CREATE OR REPLACE VIEW empv20 (deptno,msal) AS (SELECT deptno,MIN(sal) FROM emp GROUP BY deptno HAVING MIN(sal)>(SELECT MIN(sal) FROM emp WHERE deptno=20)) ;

## WITH CHECK OPTION保护视图的创建规则

之前是按照部门编号建立的视图，所以不能修改部门编号 update empv20 SET ename='tom' where empno=7369 ;  可以更新，更新的是名字，成功

## WITH READ ONLY（只读，不可修改）

视图最好不要轻易的修改

CREATE OR REPLACE VIEW empv20 (empno,ename,sal,deptno) AS SELECT empno,ename,sal,deptno FROM emp WHERE deptno=20 WITH READ ONLY;

现在任意的字段都不可更改，所以现在的视图是只读的。

如果视图的基表有多行查询（比如：group by,distinct）那么该视图也是只读

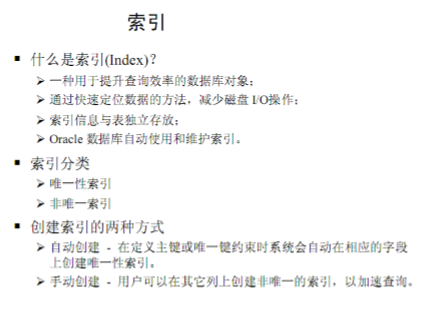
## 查看视图

Select text from user\_views;查看视图的创建语句

# 索引

select \* from user\_indexes 查询现有的索引

select \* from user\_ind\_columns 可获知索引建立在那些字段上



## 索引优缺点

建立索引的优点

1.大大加快数据的检索速度;

2.创建唯一性索引，保证数据库表中每一行数据的唯一性;

3.加速表和表之间的连接;

4.在使用分组和排序子句进行数据检索时，可以显著减少查询中分组和排序的时间。

索引的缺点

1. 索引需要占物理空间。

2.当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，降低了数据的 维护速度。

## 创建索引的原则

创建索引：

创建索引一般有以下两个目的：

维护被索引列的唯一性和提供快速访问表中数据 的策略。

--在 select 操作占大部分的表上创建索引；

--在 where 子句中出现最频繁的列上创建索引；

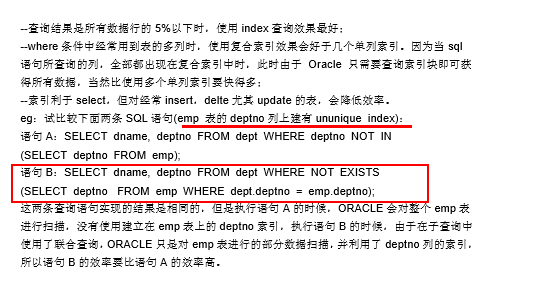
--在选择性高的列上创建索引（补充索引选择性，最高是 1，eg：primary key）

--复合索引的主列应该是最有选择性的和 where 限定条件最常用的列，并以此类推第二

列……。

--小于 5M 的表，最好不要使用索引来查询，表越小，越适合用全表扫描。

## 使用索引原则



----where 子句中的这个字段，必须是复合索引的第一个字段；

eg：一个索引是按 f1, f2, f3 的次序建立的，若 where 子句是 f2 = : var2, 则因为 f2 不是索 引的第 1 个字段，无法使用该索引。

---- where 子句中的这个字段，不应该参与任何形式的计算：任何对列的操作都将导致表 扫描，它包括数据库函数、计算表达式等等，查询时要尽可能将操作移至等号右边。

----应尽量熟悉各种操作符对 Oracle 是否使用索引的影响：

以下这些操作会显式 （explicitly）地阻止 Oracle 使用索引： is null ; is not null ; not in; !=; like ; numeric\_col+0;date\_col+0; char\_col||' '; to\_char; to\_number，to\_date 等。 Eg： Select jobid from mytabs where isReq='0' and to\_date (updatedate) >= to\_Date ( '2001-7-18', 'YYYY-MM-DD')；--updatedate 列的索引也不会生效。

## 创建索引

create index abc on student(sid,sname); create index abc1 on student(sname,sid); 这两种索引方式是不一样的 索引 abc 对 Select \* from student where sid=1; 这样的查询语句更有效 索引 abc1 对 Select \* from student where sname=‟louis‟; 这样的查询语句更有效 因此建立索引的时候，字段的组合顺序是非常重要的。一般情况下，需要经常访问的字段 放在组合字段的前面

## 索引的存储

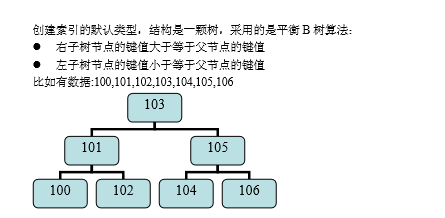
索引和表都是独立存在的。在为索引指定表空间的时候，不要将被索引的表和索引指向同 一个表空间，这样可以避免产生 IO 冲突。使 Oracle 能够并行访问存放在不同硬盘中的索 引数据和表数据，更好的提高查询速度

## 删除索引

drop index PK\_DEPT1;

## 索引类型

### B 树索引（B-Tree Index）



### 位图索引(BitMap Index)

如果表中的某些字段取值范围比较小，比如职员性别、分数列 ABC 级等。只有两个值。 这样的字段如果建 B 树索引没有意义，不能提高检索速度。这时我们推荐用位图索引 Create BitMap Index student on(sex);

## 管理索引

1）先插入数据后创建索引 向表中插入大量数据之前最好不要先创建索引，因为如果先建立索引。那么在插入每行 数据的时候都要更改索引。这样会大大降低插入数据的速度。

2）设置合理的索引列顺序

3）限制每个表索引的数量

4）删除不必要的索引

5）为每个索引指定表空间

6）经常做 insert，delete 尤其是 update 的表最好定期 exp/imp 表数据，整理数据，降低碎片（缺点：要停应用，以保持数据一致性，不实用）；

有索引的最好定期 rebuild 索引（rebuild 期间只允许表的 select 操作，可在数据库较空闲时间提交），以降低索引碎片，提高效率.

# Sql优化

## SQL 优化的实质

就是在：

结果正确的前提下，用优化器可以识别的语句，充份利用索引，执行过程中访问尽量少的数据块，减少表扫描的 I/O 次数，尽量避免全表扫描和其他额外开销。

oracle 数据库常用的两种优化器：

RBO（rule-based-optimizer）和 CBO(cost-based-optimizer)。目前更多地采用 CBO(cost-based-optimizer)基于开销的优化器。

在 CBO 方式下，Oracle 会 根据表及索引的状态信息来选择计划；

在 RBO 方式下，Oracle 会根据自己内部设置的一些规则来决定选择计划，

例如 oracle 会根据以下优先级来选择执行计划（越靠前，rank 越低， 越快）：

## 尽量少用 IN 操作符

基本上所有的 IN 操作符都可以用 EXISTS 代替，在选择 IN 或 EXIST 操作时，要根据主子 表数据量大小来具体考虑

## 尽量用 NOT EXISTS 或者外连接替代 NOT IN 操作符

因为 NOT IN 不能应用表的索引

## 尽量不用“<>”或者“!=”操作符

不等于操作符是永远不会用到索引的，因此对它的处理只会产生全表扫描。比如：a<>0 改为 a>0 or a<0

## 在设计表时，把索引列设置为 NOT NULL

判断字段是否为空一般是不会应用索引的，因为 B 树索引是不索引空值的。

## 尽量不用通配符“%”或者“\_”作为查询字符串的第一个字符

当通配符“%”或者“\_”作为查询字符串的第一个字符时，索引不会被使用。比如用 T 表中 Column1 LIKE „%5400%‟ 这个条件会产生全表扫描，如果改成 Column1 ‟X5400%‟ OR Column1 LIKE ‟B5400%‟ 则会利用 Column1 的索引进行两个范围的查询，性能肯定大大提 高

## Where子句中避免在索引列上使用计算

如果索引不是基于函数的，那么当在 Where 子句中对索引列使用函数时，索引不再起作用。 因此 Where 子句中避免在索引列上使用计算。 比如： substr(no,1,4)=‟5400‟，优化处理：no like „5400%‟ trunc(hiredate)=trunc(sysdate) ， 优 化 处 理 ： hiredate >=trunc(sysdate) and hiredate <trunc(sysdate+1)

## 用“>=”替代“>”

大于或小于操作符一般情况下是不用调整的，因为它有索引就会采用索引查找，但有 的情况下可以对它进行优化，如一个表有 100 万记录，一个数值型字段 A， 30 万记录的 A=0，30 万记录的 A=1，39 万记录的 A=2，1 万记录的 A=3。那么执行 A>2 与 A>=3 的效果就有很大的区别了，因为 A>2 时 ORACLE 会先找出为 2 的记录索引再 进行比较，而 A>=3 时 ORACLE 则直接找到=3 的记录索引。

## 利用 SGA 共享池，避开 parse 阶段

同一功能同一性能不同写法 SQL 的影响

如一个 SQL 在 A 程序员写的为 Select \* from zl\_yhjbqk

B 程序员写的为 Select \* from dlyx.zl\_yhjbqk（带表所有者的前缀）

C 程序员写的为 Select \* from DLYX.ZLYHJBQK（大写表名）

D 程序员写的为 Select \* from DLYX.ZLYHJBQK（中间多了空格）

以上四个 SQL 在 ORACLE 分析整理之后产生的结果及执行的时间是一样的，但是从 ORACLE 共享内存 SGA 的原理，可以得出 ORACLE 对每个 SQL 都会对其进行一次分析， 并且占用共享内存，如果将 SQL 的字符串及格式写得完全相同则 ORACLE 只会分析一次， 共享内存也只会留下一次的分析结果，这不仅可以减少分析 SQL 的时间，而且可以减少共 享内存重复的信息，ORACLE 也可以准确统计 SQL的执行频率。

不同区域出现的相同的 Sql 语句要保证查询字符完全相同，建议经常使用变量来代替常量， 以尽量使用重复 sql 代码，以利用 SGA 共享池，避开 parse 阶段，防止相同的 Sql 语句被多次分析，提高执行速度。 因此使用存储过程，是一种很有效的提高 share pool 共享率，跳过 parse 阶段，提高效率的 办法。

WHERE 后面的条件顺序要求

WHERE 后面的条件，表连接语句写在最前，可以过滤掉最大数量记录的条件居后。 比如： Select \* from zl\_yhjbqk where dy\_dj = '1KV 以下' and xh\_bz=1

Select \* from zl\_yhjbqk where xh\_bz=1 and dy\_dj = '1KV 以下'

以上两个 SQL 中 dy\_dj（电压等级）及 xh\_bz（销户标志）两个字段都没进行索引，所以执行的时候都是全表扫描，第一条SQL的dy\_dj = '1KV以下'条件在记录集内比率为99%， 而 xh\_bz=1 的比率只为 0.5%，在进行第一条 SQL 的时候 99%条记录都进行 dy\_dj 及 xh\_bz的比较，而在进行第二条 SQL 的时候 0.5%条记录都进行 dy\_dj 及 xh\_bz 的比较，以此可以 得出第二条 SQL 的 CPU 占用率明显比第一条低。

## 使用表的别名，并将之作为每列的前缀

当在 Sql 语句中连接多个表时，使用表的别名，并将之作为每列的前缀。这样可以减少解析时间。

## 进行了显式或隐式的运算的字段不能进行索引

比如： ss\_df+20>50，优化处理：ss\_df>30 „X‟||hbs\_bh>‟X5400021452‟，优化处理：hbs\_bh>‟5400021542‟ sk\_rq+5=sysdate，优化处理：sk\_rq=sysdate-5 hbs\_bh=5401002554，优化处理：hbs\_bh=‟ 5401002554‟，注：此条件对 hbs\_bh 进行隐 式的 to\_number 转换，因为 hbs\_bh 字段是字符型。

## 用 UNION ALL 代替 UNION

UNION 是最常用的集操作，使多个记录集联结成为单个集，对返回的数据行有唯一性要求， 所以 oracle 就需要进行 SORT UNIQUE 操作（与使用 distinct 时操作类似），如果结果集又 比较大，则操作会比较慢； UNION ALL 操作不排除重复记录行，所以会快很多，如果数据本身重复行存在可能性较 小时，用 union all 会比用 union 效率高很多！

## 其他操作

尽量使用 packages： Packages 在第一次调用时能将整个包 load 进内存，对提高性能有帮 助。

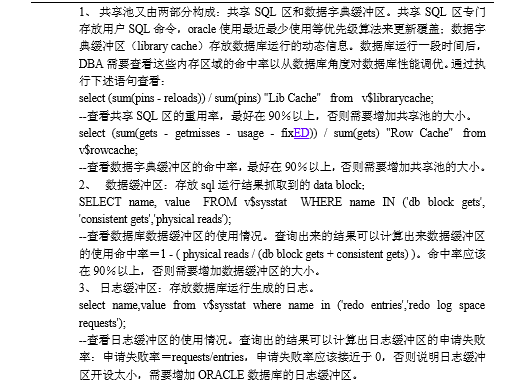
尽量使用 cached sequences 来生成 primary key ：提高主键生成速度和使用性能。

很好地利用空间：如用 VARCHAR2 数据类型代替 CHAR 等

使用 Sql 优化工具：sqlexpert；toad；explain-table；PL/SQL；OEM

## 通过改变 oracle 的 SGA 的大小

SGA：数据库的系统全局区。 SGA 主要由三部分构成：共享池、数据缓冲区、日志缓冲区



序列

## 创建序列

Create sequence myseq

Start with 1

Increment by 1

Order

cache 20

Nocycle;

## NextVal,CurrVal

Select myseq.nextval from dual;

Select myseq.currval from dual; (必须先有 nextval，才能有 currval)

Insert into table1 values(myseq.nextval) 这时候已经是 2 了

## Cycle，Cache

而用了 nocycle，就可以确保当该序列用于多张表的时候，ID 是唯一的

用 cycle 时，用法如下：

create sequence myseq2

start with 1

increment by 1

cycle maxvalue 3

nocache ;

这样到 3 之后，要会重新从 1 开始

如果指定 CACHE 值，ORACLE就可以预先在内存里面放置一些 sequence，这样存取的快些。cache里 面的取完后，oracle自动再取一组到 cache。 使用 cache或许会跳号， 比如数据库突然不正常 down 掉 （shutdown abort),cache中的sequence就会丢失. 所以可以在create sequence的时候用nocache防止 这种情况

不能改变当前值，但是可以改变增量 Alter sequence myseq increment by 3;

## 同义词

在任何一个用户下，都可以直接访问 dual,而不需要加上前缀的用户名如:scott.emp Select \* from dual;

为什么？因为同义词的存在

Dual 其实是 sys 用户下的一张表 select table\_name from user\_tables where lower(table\_name) = 'dual';

作用： 很方便的操作不同用户下的对象 能使两个应用程序使用不同的名字指向同一张表 使用不同的用户指向同一张表的。

Create synonym dept for soctt.dept;(这样创建的同义词是私有的，只有创建者才能用)

Drop synonym dept;

Create public synonym dept for soctt.dept;（这样创建的同义词才是公有的） Drop public synonym dept;

# PL SQL

## PL/SQL 块

是在 SQL 语言之上发展起来的一种应用，可以集中的处理各种复杂的 SQL 操 作。 组成：

DECLARE：

声明部分

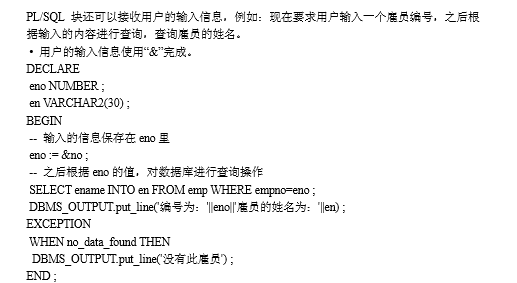
BEGIN

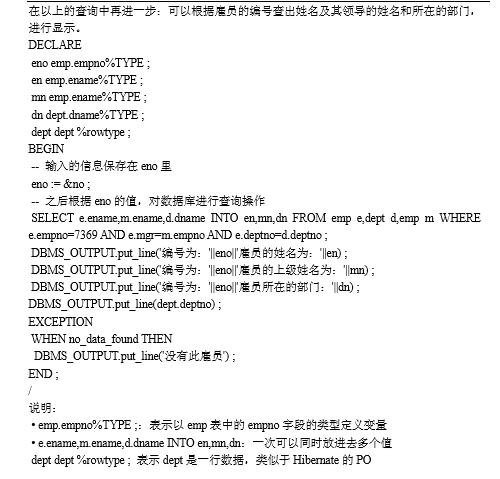
编写主题

EXCEPTION

捕获异常

END ;





## Loop 循环（do…while）

PL/SQL 之中也包含了：循环、分支等条件控制语句

Loop 循环(do…while) 格式：

LOOP 循环的语句 ;

EXIT WHEN 终止条件 ;

循环条件必须更改 ;

END LOOP ;

循环输出 1~10。

DECLARE cou NUMBER ;

BEGIN -- 必须给一个初始值

cou := 1 ;

LOOP

DBMS\_OUTPUT.put\_line('cou = '||cou) ;

EXIT WHEN cou>10 ;

cou := cou + 1 ;

END LOOP ;

END ; -- 此循环是先执行一次之后再进行判断

## while 循环

格式：

while(判断循环的条件)

loop 循环的语句 ;

循环条件的改变 ;

End loop ;

使用此语句修改上面的程序：

DECLARE cou NUMBER ;

BEGIN

-- 必须给一个初始值

cou := 1 ;

WHILE(cou<10)

LOOP DBMS\_OUTPUT.put\_line('cou = '||cou) ;

cou := cou + 1 ;

END LOOP ;

END ; / 此语句，是先判断，之后如果条件满足则执行，与 while 循环类似。

## for 循环

格式：

FOR 变量名称 in 变量的初始值..结束值

LOOP 循环语句 ;

END LOOP ;

DECLARE cou NUMBER ;

BEGIN

FOR cou IN 1..10

LOOP DBMS\_OUTPUT.put\_line('cou = '||cou) ;

END LOOP ;

END ;

## IF 语句

条件判断 格式：

IF 条件 THEN

满足条件时，执行此语句

END IF ;

DECLARE cou NUMBER ;

BEGIN cou := 11 ;

IF cou>10 THEN

DBMS\_OUTPUT.put\_line('cou = '||cou) ;

END IF ;

END ;

## IF…ELSE 语句

如果 IF 满足了，则执行，否则执行 ELSE

DECLARE cou NUMBER ;

BEGIN cou := 1 ;

IF cou>10 THEN

DBMS\_OUTPUT.put\_line('cou = '||cou) ;

ELSE

DBMS\_OUTPUT.put\_line('条件不满足') ;

END IF ;

END ;

## IF…ELSE…IF…ELSE 语句

DECLARE

cou NUMBER ;

BEGIN cou := 1 ;

IF cou>10 THEN

DBMS\_OUTPUT.put\_line('cou = '||cou) ;

ELSIF cou<5 THEN

DBMS\_OUTPUT.put\_line('值小于 5') ;

ELSE DBMS\_OUTPUT.put\_line('条件不满足') ;

END IF ; END ;

## GOTO 语句

无条件跳转语句

DECLARE eno emp.empno%TYPE ;

sal emp.sal%TYPE ;

BEGIN

eno := &en ;

SELECT sal INTO sal FROM emp WHERE empno=eno ;

IF sal>3500 THEN

goto po1 ;

ELSIF

sal>2000 THEN

goto po2 ;

ELSE goto po3 ;

END IF ;

<<po1>> DBMS\_OUTPUT.put\_line('高工资。。。') ;

<<po2>> DBMS\_OUTPUT.put\_line('中等工资。。') ;

<<po3>> DBMS\_OUTPUT.put\_line('底工资。。。') ;

END ;

# 游标

游标是一种 PL/SQL 控制结构；可以对 SQL 语句的处理进行显示控制，便于对表的行数据 逐条进行处理。 游标并不是一个数据库对象，只是存留在内存中

## 操作步骤：

• 声明游标

• 打开游标

• 取出结果，此时的结果取出的是一行数据

• 关闭游标 到底那种类型可以把一行的数据都装进来

• 此时使用 ROWTYPE 类型，此类型表示可以把一行的数据都装进来。

## 游标的属性

oracle 游标有4个属性： %ISOPEN ， %FOUND ， %NOTFOUND， %ROWCOUNT

*--%ISOPEN 判断游标是否被打开，如果打开%ISOPEN 等于true,否则等于false*

*--%FOUND %NOTFOUND 判断游标所在的行是否有效，如果有效，则%FOUNDD等于true，否则等于false*

*-- %ROWCOUNT 返回当前位置为止游标读取的记录行数。*

## 例子

查询雇员编号为 7369 的信息（肯定是一行信息）。

DECLARE

eno emp.empno%TYPE ;

empInfo emp%ROWTYPE ;

BEGIN

eno := &en ;

SELECT \* INTO empInfo FROM emp WHERE empno=eno ;

DBMS\_OUTPUT.put\_line('雇员编号：'||empInfo.empno) ;

DBMS\_OUTPUT.put\_line('雇员姓名：'||empInfo.ename) ;

END ;

## 使用 for 循环操作游标（比较常用）

DECLARE

*-- 声明游标*

CURSOR mycur IS

SELECT \* FROM emp where empno = -1;

empInfo emp%ROWTYPE;

cou NUMBER;

BEGIN

*-- 游标操作使用循环，但是在操作之前必须先将游标打开*

FOR empInfo IN mycur LOOP

cou := mycur%ROWCOUNT;

DBMS\_OUTPUT.put\_line(cou || '雇员编号：' || empInfo.empno);

DBMS\_OUTPUT.put\_line(cou || '雇员姓名：' || empInfo.ename);

END LOOP;

END;

## 编写第一个游标，输出全部的信息

DECLARE

*-- 声明游标*

CURSOR mycur IS

SELECT \* FROM emp;

*-- List (EmpPo)*

empInfo emp%ROWTYPE;

BEGIN

*-- 游标操作使用循环，但是在操作之前必须先将游标打开*

OPEN mycur;

*-- 使游标向下一行*

FETCH mycur

INTO empInfo;

*-- 判断此行是否有数据被发现*

WHILE (mycur%FOUND) LOOP

DBMS\_OUTPUT.put\_line('雇员编号：' || empInfo.empno);

DBMS\_OUTPUT.put\_line('雇员姓名：' || empInfo.ename);

*-- 修改游标，继续向下*

FETCH mycur

INTO empInfo;

END LOOP;

END;

## 也可以使用另外一种方式循环游标：LOOP…END LOOP

DECLARE

*-- 声明游标*

CURSOR mycur IS

SELECT \* FROM emp;

empInfo emp%ROWTYPE;

BEGIN

*-- 游标操作使用循环，但是在操作之前必须先将游标打开*

OPEN mycur;

LOOP

*-- 使游标向下一行*

FETCH mycur

INTO empInfo;

EXIT WHEN mycur%NOTFOUND;

DBMS\_OUTPUT.put\_line('雇员编号：' || empInfo.empno);

DBMS\_OUTPUT.put\_line('雇员姓名：' || empInfo.ename);

END LOOP;

END;

注意 1：

在打开游标之前最好先判断游标是否已经是打开的。

通过 ISOPEN 判断，

格式：游标%ISOPEN

IF mycur%ISOPEN THEN null ;

ELSE OPEN mycur ;

END IF ;

注意 2：

可以使用 ROWCOUNT 对游标所操作的行数进行记录。

DECLARE

*-- 声明游标*

CURSOR mycur IS

SELECT \* FROM emp;

empInfo emp%ROWTYPE;

cou NUMBER;

BEGIN

*-- 游标操作使用循环，但是在操作之前必须先将游标打开*

IF mycur%ISOPEN THEN

null;

ELSE

OPEN mycur;

END IF;

LOOP

*-- 使游标向下一行*

FETCH mycur

INTO empInfo;

EXIT WHEN mycur%NOTFOUND;

cou := mycur%ROWCOUNT;

DBMS\_OUTPUT.put\_line(cou || '雇员编号：' || empInfo.empno);

DBMS\_OUTPUT.put\_line(cou || '雇员姓名：' || empInfo.ename);

END LOOP;

END;

## 函数

函数就是一个有返回值的过程。

定义一个函数：此函数可以根据雇员的编号查询出雇员的年薪

CREATE OR REPLACE FUNCTION myfun(eno emp.empno%TYPE) RETURN NUMBER AS

rsal NUMBER;

BEGIN

SELECT (sal + nvl(comm, 0)) \* 12 INTO rsal FROM emp WHERE empno = eno;

RETURN rsal;

END;

调用此函数：SELECT myfun(7369) FROM dual ;

# 存储过程

## 过程（存储过程）

与过程相比，存储过程是存在数据库中的一个对象

如果编译错误。可以用 show errors or show errors procedure mypro

现在定义一个简单的过程，就是打印一个数字

CREATE OR REPLACE PROCEDURE myproc

AS

i NUMBER ;

BEGIN i := 100 ;

DBMS\_OUTPUT.put\_line('i = '||i) ;

END ;

## 执行过程： exec 过程名字

## 例子

下面编写一个过程，要求，可以传入部门的编号，部门的名称，部门的位置，之后调用此 过程就可以完成部门的增加操作。

CREATE OR REPLACE PROCEDURE myproc(dno dept.deptno%TYPE,

name dept.dname%TYPE,

dl dept.loc%TYPE)

AS

cou NUMBER;

BEGIN

*-- 判断插入的部门编号是否存在，如果存在则不能插入*

SELECT COUNT(deptno) INTO cou FROM dept WHERE deptno = dno;

IF cou = 0 THEN

*-- 可以增加新的部门*

INSERT INTO dept (deptno, dname, loc) VALUES (dno, name, dl);

DBMS\_OUTPUT.put\_line('部门插入成功！');

ELSE

DBMS\_OUTPUT.put\_line('部门已存在，无法插入！');

END IF;

END;

## 过程的参数类型

• IN：值传递，默认的

• IN OUT：带值进，带值出

• OUT：不带值进，带值出

## IN OUT 类型

CREATE OR REPLACE PROCEDURE myproc(dno IN OUT dept.deptno%TYPE,

name dept.dname%TYPE,

dl dept.loc%TYPE) AS

cou NUMBER;

BEGIN

*-- 判断插入的部门编号是否存在，如果存在则不能插入*

SELECT COUNT(deptno) INTO cou FROM dept WHERE deptno = dno;

IF cou = 0 THEN

*-- 可以增加新的部门*

INSERT INTO dept (deptno, dname, loc) VALUES (dno, name, dl);

DBMS\_OUTPUT.put\_line('部门插入成功！');

*-- 修改 dno 的值*

dno := 1;

ELSE

DBMS\_OUTPUT.put\_line('部门已存在，无法插入！');

dno := -1;

END IF;

END;

编写 PL/SQL 块验证过程：

DECLARE deptno dept.deptno%TYPE ;

BEGIN

deptno := 12 ;

myproc(deptno,'开发','南京') ;

DBMS\_OUTPUT.put\_line(deptno) ;

END ;

# 触发器

存放在数据库中，并被隐含执行的存储过程。在 Oracle8i 之前，只允许给予表或者视图的 的 DML 的操作，而从 Oracle8i 开始，不仅可以支持 DML 触发器，也允许给予系统事件和 DDL 的操作

## 语句触发器

### Before 语句触发器

例如：禁止工作人员在休息日改变雇员信息

create or replace trigger tr\_src\_emp

before insert or update or delete on emp

begin

if to\_char(sysdate, 'DY', 'nls\_date\_language=AMERICAN') in ('SAT', 'SUN') then

raise\_application\_error(-20001,

'can‟t modify user information in weekend');

end if;

end;

### 使用条件谓语

create or replace trigger tr\_src\_emp

before insert or update or delete on emp

begin

if to\_char(sysdate, 'DY') in ('星期六', '星期天') then

case

when inserting then

raise\_application\_error(-20001, 'fail to insert');

when updating then

raise\_application\_error(-20001, 'fail to update');

when deleting then

raise\_application\_error(-20001, 'fail to delete');

end case;

end if;

end;

### after 语句触发器

例如：为了统计在 EMP 表上的增、删、改的次数。

先建一张表

Create table audit\_table( Name varchar2(20),ins int,upd int,del int,starttime date,endtime date); 然后建立触发器

Create or replace trigger tr\_audit\_emp

After insert or update or delete on emp

Declare

v\_temp int;

Begin

Select count(\*) into v\_temp from audit\_table Where name = ‟EMP‟;

If v\_temp = 0 then

Insert into audit\_table values („EMP‟, 0, 0, 0, sysdate, null);

End if;

Case

When inserting then

Update audit\_table

set ins = ins + 1, endtime = sysdate

where name = ‟EMP‟;

When updating then

Update audit\_table

set upd = upd + 1, endtime = sysdate

where name = ‟EMP‟;

When deleting then

Update audit\_table

set del = del + 1, endtime = sysdate

where name = ‟EMP‟;

End case;

End;

## 行触发器

执行 DML 操作时，每作用一行就触发一次触发器

### Bofre 行触发器

例如：确保员工工资不能低于原有工资

Create or replace trigger tr\_emp\_sal

before update of sal on emp

for each row

begin

if :new.sal < :old.sal then

raise\_application\_error(-20010, 'sal should not be less');

end if;

end;

### after 行触发器

例如：统计员工工资变化

Create table audit\_emp\_change( Name varchar2(10), Oldsal number(6,2), Newsal number(6,2), Time date);

Create or replace trigger tr\_sal\_sal

after update of sal on emp

for each row

declare

v\_temp int;

begin

select count(\*)

into v\_temp

from audit\_emp\_change

where name = :old.ename;

if v\_temp = 0 then

insert into audit\_emp\_change

values

(:old.ename, :old.sal, :new.sal, sysdate);

else

update audit\_emp\_change

set oldsal = :old.sal, newsal = :new.sal, time = sysdate

where name = :old.ename;

end if;

end;

### 限制行触发器

Create or replace trigger tr\_sal\_sal

after update of sal on emp

for each row

when (old.job = ‟SALESMAN‟)

declare

v\_temp int;

begin

select count(\*)

into v\_temp

from audit\_emp\_change

where name = :old.ename;

if v\_temp = 0 then

insert into audit\_emp\_change

values

(:old.ename, :old.sal, :new.sal, sysdate);

else

update audit\_emp\_change

set oldsal = :old.sal, newsal = :new.sal, time = sysdate

where name = :old.ename;

end if;

end;

# 用户和权限

## --创建用户

create user test identified by test123;

## --把连接权限授予用户

grant connect to test;

drop user test;

## --修改用户

alter user test default tablespace users;

alter user test identified by test111;

alter user test account lock;

alter user test account unlock;

grant resource to test;

## --对象授权

grant select on scott.dept to test;

grant execute on scott.proc\_test to test;

## --级联授权

*--对象授权的级联授权用 with grant option*

*--级联授权后test用户也有授权给其他用户select on scott.dept 的权限*

grant select on scott.dept to test with grant option;

grant insert on scott.dept to test with grant option;

grant all on scott.dept to test with grant option;

## --系统授权

grant create table to test;

grant create user to test;

grant drop user to test;

## --级联授权

*--系统的级联授权用 with admin option*

grant create user to test with admin option;

*--通过角色控制权限*

create role myrole;

grant select on scott.dept to myrole;

grant select on scott.emp to myrole;

grant myrole to test;

*--一般开发用到的权限*

grant connect to test;

grant resource to test;

*--如果要查询其他用户的表，使用下面语句*

grant select any table to test;

## --解除授权

*--解除对象授权*

revoke select on scott.dept from test;

revoke select any table from test;

*--解除系统授权*

revoke create user from test;

*--解除角色授权*

revoke myrole from test;

# insert all和merge

create table orderinfo

(

order\_id integer primary key,

salary number(5) not null

)

create table smallorder

(

order\_id integer primary key,

salary number(5) not null

)

create table largeorder

(

order\_id integer primary key,

salary number(5) not null

)

create table specialorder

(

order\_id integer primary key,

salary number(5) not null

)

insert into orderinfo values(1, 100);

insert into orderinfo values(2, 600);

insert into orderinfo values(3, 700);

insert into orderinfo values(4, 200);

insert into orderinfo values(5, 300);

select \* from orderinfo

--同时插入多个表

## insert all

into smallorder values(order\_id,salary)

into largeorder values(order\_id,salary)

select \* from orderinfo;

--同时插入多个表(带条件)

insert all

when salary >= 0 and salary < 300 then

into smallorder values(order\_id,salary)

when salary >= 700 then

into specialorder values(order\_id,salary)

when salary >= 300 then

into largeorder values(order\_id,salary)

select \* from orderinfo; --700同时插入largeorder表和specialorder表

## insert first

when salary >= 0 and salary < 300 then

into smallorder values(order\_id,salary)

when salary >= 700 then

into specialorder values(order\_id,salary)

when salary >= 300 then

into largeorder values(order\_id,salary)

select \* from orderinfo; --700只插入specialorder表，与条件语句的顺序有关

select \* from smallorder;

select \* from largeorder;

select \* from specialorder

## --MERGE INTO的用法

create table desctable

(

id integer primary key,

salary number(5) not null

)

create table srctable

(

id integer primary key,

name varchar2(20),

salary number(5) not null

)

insert into desctable values(1, 100);

insert into desctable values(2, 200);

insert into desctable values(3, 300);

insert into srctable values(1, '', 100);

insert into srctable values(3, '', 1000);

insert into srctable values(4, '', 400);

MERGE INTO desctable d

USING (select id, salary from srctable) s --表名也可以

ON(d.id = s.id)

WHEN MATCHED THEN

UPDATE set d.salary = d.salary \* 1.5

WHEN NOT MATCHED THEN

INSERT VALUES(s.id, s.salary);

select \* from desctable

select \* from srctable

# 查看表结构信息

## --查看表的结构

select a.column\_name 字段名,

a.data\_type 数据类型,

a.data\_length 长度,

a.data\_precision 整数位,

a.data\_scale 小数位,

a.nullable 允许空值,

a.data\_default 缺省值,

b.comments 备注

from user\_tab\_columns a, user\_col\_comments b

where a.column\_name = b.column\_name

and a.table\_name = b.table\_name

and a.table\_name = upper('emps');

## --查看列的注释

select \* from user\_col\_comments where table\_name = upper('emps');

## --查看表的注释

select \* from user\_tab\_comments where table\_name = upper('emps');

## --查看表的约束

select \* from user\_constraints where table\_name = upper('emps');

create unique index idx\_unique\_ename on

emps(name);

## --查看索引

select \* from user\_indexes where table\_name = upper('emps');

# 创建表inline

create table department

(

id number(2) not null primary key,

name varchar2(14) unique

)

drop table tb;

## --创建表，有匿名约束

create table tb(

id number not null primary key,

name varchar2(20) unique check(length(name)>2),

deptid number references department(id) on delete cascade); -- 或者set null

drop table def\_table;

## --创建有名字的约束

-- default 应该紧跟在数据类型后面

create table def\_table(

col\_id number(5),

name varchar2(20 char) default 'noname' constraint ck\_name\_nn not null enable ,--disable

email varchar2(200 char) default 'email',

constraint pk\_col\_id primary key(col\_id),

constraint ck\_name\_len check(length(name)>3)

);

select \* from sys.user\_tables;

## --判断表是否存在，存在则删除，然后在建立

declare

t varchar(200) :='tb';

tname number;

begin

select count(table\_name) into tname from sys.user\_tables where table\_name=Upper(t);

if tname=0 then --如果使用select table\_name into tname，不存在是会报错

dbms\_output.put\_line(t || ' not exists.');

else

dbms\_output.put\_line(t || ' already exists.');

execute immediate 'drop table ' || t;

end if;

execute immediate 'create table ' || t || '(

id number not null primary key,

name varchar2(20) unique check(length(name)>2),

deptid number references dept(id) on delete cascade)';

dbms\_output.put\_line(t || ' created .');

end;