

```
alter table test add(name varchar2(20));
```

```
alter table test drop column sex;  
alter table test set unused column sex;  
alter table test drop unused columns;
```

---

更改用户密码

```
sql>alter user 管理员 identified by 密码;
```

---

创建表空间的数据文件

```
sql>create tablespace test datafile 'd:\oracle\binbo.dbf' size 10m;
```

---

创建用户

```
sql>create user 用户名 identified by 用户名;
```

---

bfile 类型实例

创建目录

```
create directory tnkdir as 'c:\';
```

删除目录

```
drop directory tnkdir
```

授权

```
grant read on directory tnkdir to scott;
```

建表

```
create table bfiletest(id number(3), fname bfile);
```

添加数据

```
insert into bfiletest values(1,bfilename('TMPDIR','tmptest.java'));
```

---

查看用户

```
sql>show user
```

---

检查语句是否有错

```
show error
```

---

锁定用户

```
sql>alter user 用户名 account lock
```

---

解除用户

```
sql>alter user 用户名 account unlock
```

---

删除用户

```
sql>drop user zl;
```

---

给用户创建表权限

```
sql>grant create table to 用户名;
```

---

授管理员权限

sql>grant dba to 用户名;

给用户登录权限

sql>grant connect to 用户名

给用户无限表空间权限

sql>grant unlimited tablespace to 用户名;

收回权限

sql>revoke dba from 用户名;

查看用户下所有的表

SQL>select \* from user\_tables;

查看名称包含 log 字符的表

SQL>select object\_name,object\_id from user\_objects  
where instr(object\_name,'LOG')>0;

查看某表的创建时间

SQL>select object\_name,created from user\_objects where  
object\_name=upper('&table\_name');

查看某表的大小

SQL>select sum(bytes)/(1024\*1024) as "size(M)" from user\_segments  
where segment\_name=upper('&table\_name');

查看放在 ORACLE 的内存区里的表

SQL>select table\_name,cache from user\_tables where instr(cache,'Y')>0;

再添加一个表空间的数据文件

sql>alter tablespace test add datafile 'd:\oracle\test1.dbf' size 10m;

建表 SQL>create table studen(stuno int,stuname varchar(8) not null,stubirth date default  
to\_date('1987-5-9','YYYY-MM-DD'));

向表结构中加入一列 SQL>alter table studen add(stuphoto varchar(9));

从表结构中删除一列 SQL>alter table studen drop column stuphoto;

修改表一列的长度 SQL>alter table studen modify(stuno number(4));

隐藏将要删除的一列 SQL>alter table studen set unused column stuphoto;

删除隐藏的列 SQL>alter table studen drop unused columns;

向表中加入约束 SQL>alter table studen add constraint pk primary key(stuno);

删除约束 SQL>alter table studen drop constraint pk;

---

---

创建表

sql>create table 用户名(name varchar2(20),password varchar(20)) tablespace 空间名;

---

---

添加字段

sql>alter table test add(column\_x char(10) not null);

---

---

更改字段

sql>alter table emp modify(column\_x char (20));

---

---

删除字段

如待删除域属于某个索引，则不允许删除操作，必须将此域先设置为 NULL。

sql>alter table emp modify(column\_x null);

sql>update emp set column\_x=null;

sql>commit;

sql>alter table emp drop(column\_x);

---

---

选择表空间

sql>alter user 用户名 default tablespace test;

---

---

管理员删除别的用户中的表

sql>drop table 用户名.表名;

---

---

退出

sql>exit;

---

---

默认进入

sql>sqlplus "/" as sysdba"

---

---

查看数据库

sql>show parameter block;

---

---

写大量语句用记事本，新建方式。

输入"ed"回车

保存后

输入"/"运行;

---

---

查询用户有多少表

sql>select \* from tab;

---

---

SQLServer 取时间

sql>select getdate

oracle 取时间

```
sql>sysdate;
```

---

操作表结构数据库定义语言命令  
(不记录在日志文件中)

create table 建表

```
sql>create table test(name varchar2(20),age date,sex char(2));
```

```
sql>insert into test(name,age,sex) values('aa',sysdate,'男');
```

```
sql>insert into test(name,age,sex) values('bb',to_date('1888-8-8','yyyy-aa-dd hh24:mi:ss'),'男');
```

```
sql>select * from test;
```

---

查询男和女总数

```
sql>select sex,count(sex) from test group by sex;
```

test 表中数据输入 test1 表中

SQLServer---select \* into test1 from test;

oracle---create table test1 as select \* from test;

---

更改会话时间

```
sql>alter session set nls_date_format='yyyy-mm-dd hh24:mi:ss';
```

---

```
sql>show parameter block 表和视图
```

```
sql>show parameter date 查数据结构
```

---

SQLServer 中

--删除表中相同数据

```
sql>create table test1 as select distinct * from test;
```

--删除表数据

```
sql>truncate table test;
```

--把 test 中数据输入到 test1 中

```
sql>insert into test(select * from test1);
```

---

rowid(表中存储地址相当表 id)和 rownum(表序号)称伪列(用法)

```
sql>select name,age,sex,rowid,rownum from test1;
```

查出前三行

```
sql>select * from test where rownum<=3;
```

查出后三行

```
sql>select * from (select name n,age a,sex s,rownum r from test) where r>(select count(*) from test)-3;
```

删除后三行

```
SQL> delete from test where name not in(select name from test where rownum<=(select count(*) from test)-3);
```

删除相同行

```
sql>delete from test where rowid not in(select max(rowid) from test group by name,age,sex);
```

删除所有表

```
sql>select 'drop table' ||tname|| ':' from tab;
sql>spool c:\test.sql;
sql>select 'drop table' ||tname|| ':' from tab;
sql>spool off
sql>@c:\test.sql;
```

---

alter table 修改表  
truncate table 节段表(只删除数据)  
drop table 删除表

---

查看表结构  
desc 表名;

---

查出成绩的前三名  
sql>select \* from (select \* from stu order by score desc) where rownum<=3;

---

更改字符集  
SQL>startup mount  
SQL>alter system enable restricted session;  
SQL>alter system set job\_queue\_processes=0;  
SQL>alter database open;  
SQL>alter database character set ZHS16GBK;  
SQL>shutdown  
SQL>startup

---

将一张表或几张表中的域重新组合后插入新表。  
假定原先的两张表为 emp,work, 现选择部分数据域合并为 emp\_work  
建立 emp\_work  
SQL>insert into emp\_new select a.no, sysdate, a.name, b.service\_duration from emp a, work b  
where a.no=b.no;  
SQL>commit;  
这样的方式仍然要使用回滚段, 为加快数据迁移速度, 可将 insert 替换成 insert /\*+APPEND\*/  
(大小写不论), 指示 oracle 以直通方式直接写数据文件, 绕过回滚空间。  
SQL>insert /\*+APPEND\*/ into emp\_new select a.no, sysdate, a.name, b.service\_duration from  
emp a, work b where a.no=b.no;  
SQL>commit;

---

DDL 数据定义语言(create,alter,drop)  
DML 数据操纵语言(insert,select,delete,update)  
TCL 事务控制语言(commit,savepoint,rollback)  
DCL 数据控制语言(GRANT REVOKE)

---

一个表中的某一列输到另一个表中

```
insert into stu1(name)(select name from stu);
```

---

事务

```
rollback;
```

```
insert into stu1(name)(select name from stu);
```

```
commit;提交
```

---

COMMIT - 提交并结束事务处理

ROLLBACK - 撤销事务中已完成的工作

SAVEPOINT - 标记事务中可以回滚的点

```
SQL> update order_master set del_date = '30-8 月-05' WHERE orderno <= 'o002';
```

```
SQL> savepoint mark1;
```

```
SQL> delete FROM order_master WHERE orderno = 'o002';
```

```
SQL> savepoint mark2;
```

```
SQL> rollback TO SAVEPOINT mark1;
```

```
SQL> COMMIT;
```

---

换名

```
set sqlprompt "scott>";
```

---

GRANT 授予权限

```
SQL> GRANT SELECT ON vendor_master TO accounts WITH GRANT OPTION;
```

REVOKE 撤销已授予的权限

```
SQL> REVOKE SELECT, UPDATE ON order_master FROM MARTIN;
```

---

比较操作符

```
SQL> SELECT vencode, venname, tel_no
```

```
FROM vendor_master
```

```
WHERE venname LIKE 'j__s';
```

```
SQL> SELECT orderno FROM order_master
```

```
WHERE del_date IN ( '06-1 月-05' , '05-2 月-05');
```

```
SQL> SELECT itemdesc, re_level
```

```
FROM itemfile
```

```
WHERE qty_hand < max_level/2;
```

---

逻辑操作符

```
SQL> SELECT * FROM order_master
```

```
WHERE odate > '10-5 月-05'
```

```
AND del_date < '26-5 月-05' ;
```

---

集合操作符将两个查询的结果组合成一个结果

```
SQL> SELECT orderno FROM order_master
```

```
MINUS
```

```
SELECT orderno FROM order_detail;
```

```
select * from scott.stu
```

union (all) 重复的去掉[intersect 把相同的取出来][minus 显示不相同的数]

```
select * from stu
```

显示相同的数据

```
select name from stu intersect select name from stu1;
```

连接操作符

连接操作符用于将多个字符串或数据值合并成一个字符串

```
SQL> SELECT (venname||' 的地址是 '
```

```
||venadd1||' '||venadd2 ||' '||venadd3) address
```

```
FROM vendor_master WHERE vencode='V001';
```

操作符的优先级

SQL 操作符的优先级从高到低的顺序是：

算术操作符 -----最高优先级

连接操作符

比较操作符

NOT 逻辑操作符

AND 逻辑操作符

OR 逻辑操作符 -----最低优先级

用来转换空值的函数

NVL

NVL2

NULLIF

```
SELECT itemdesc, NVL(re_level,0) FROM itemfile;
```

```
SELECT itemdesc, NVL2(re_level,re_level,max_level) FROM itemfile;
```

```
SELECT itemdesc, NULLIF(re_level,max_level) FROM itemfile;
```

GROUP BY 和 HAVING 子句

GROUP BY 子句

用于将信息划分为更小的组

每一组行返回针对该组的单个结果

HAVING 子句

用于指定 GROUP BY 子句检索行的条件

```
SELECT p_category, MAX(itemrate) FROM itemfile GROUP BY p_category;
```

```
SELECT p_category, MAX(itemrate) FROM itemfile GROUP BY p_category HAVING
```



p\_category NOT IN ('accessories');

---

ROW\_NUMBER (row\_number)返回连续的排位，不论值是否相等

RANK(rank) 具有相等值的行排位相同，序数随后跳跃

DENSE\_RANK(dense\_rank) 具有相等值的行排位相同，序号是连续的

SELECT d.dname, e.ename, e.sal, DENSE\_RANK()

OVER (PARTITION BY e.deptno ORDER BY e.sal DESC)

AS DENRANK

FROM emp e, dept d WHERE e.deptno = d.deptno;

---

日期函数

ADD\_MONTHS(当前只加月)

alter session set nls\_date\_format='yyyymmdd hh24miss';

select add\_months(sysdate,2) from dual;

---

MONTHS\_BETWEEN(前面时间减后面时间=得之间月差)

select months\_between(sysdate,to\_date('2007-6-10','yyyy-mm-dd')) from dual;

---

LAST\_DAY(求得当前月的最后一天)

select last\_day(sysdate) from dual;

---

ROUND(round 年-月-日-->四舍五入)

select round(2.3) from dual;

select round(to\_date('2007-6-10','yyyy-mm-dd'),'year') from dual;

select round(to\_date('2007-6-10','yyyy-mm-dd'),'month') from dual;

select round(to\_date('2007-6-10','yyyy-mm-dd'),'day') from dual;

---

NEXT\_DAY(下一星期的星期二)

select next\_day(to\_date('2007-6-10','yyyy-mm-dd'),'星期二') from dual;

---

TRUNC(trunc)

---

EXTRACT(extract)

select extract(year from date '1998-03-07') from dual;

select extract(month from to\_date ('1998-03-07','yyyy-mm-dd')) from dual;

---

2008 年 2 月有多少天

inbo---->select extract(day from last\_day(to\_date ('2008-02-07','yyyy-mm-dd'))) from dual;

---

2003-4-3 与 1956-3-1 之间有多少天

inbo---->select round(months\_between(to\_date('2003-4-3','yyyy-mm-dd'),to\_date('1956-3-1','yyyy-mm-dd'))/12) from dual;

---

把两边的 9 去掉

```
select trim('9' from '9999dddddd99999') from dual;
```

去空格

```
select trim(' ' from ' 9999dddddd99999') from dual;
```

---

---

函数	输入	输出
Initcap(char)	Select initcap('hello') from dual;	Hello
Lower(char)	Select lower('FUN') from dual;	fun
Upper(char)	Select upper('sun') from dual;	SUN
Ltrim(char,set)	Select ltrim('xyzadams','xyz') from dual;	adams
Rtrim(char,set)	Select rtrim('xyzadams','ams') from dual;	xyzad
Translate(char, from, to)	Select translate('jack','j','b') from dual;	back
Replace(char,searchstring,[rep string])	Select replace('jack and jue','j','bl') from dual;	black and blue
Instr(char, m, n)	Select instr('worldwide','d') from dual;	5
Substr(char, m, n)	Select substr('abcdefg',3,2) from dual;	cd
Concat(expr1,expr2)	Select concat('Hello',' world') from dual;	Hello world

---

---

数字函数接受数字输入并返回数值结果

函数	输入	输出
Abs(n)	Select abs(-15) from dual;	15
Ceil(n)	Select ceil(44.778) from dual;	45
Cos(n)	Select cos(180) from dual;	-.5984601
Cosh(n)	Select cosh(0) from dual;	1
Floor(n)	Select floor(100.2) from dual;	100
Power(m,n)	Select power(4,2) from dual;	16
Mod(m,n)	Select mod(10,3) from dual;	1
Round(m,n)	Select round(100.256,2) from dual;	100.26
Trunc(m,n)	Select trunc(100.256,2) from dual;	100.25
Sqrt(n)	Select sqrt(4) from dual;	2
Sign(n)	Select sign(-30) from dual;	-1

=====

字符函数

查看有多少个字符

```
SQL> SELECT LENGTH('frances') FROM dual;
```

-----

```
SQL> SELECT vencode,
           DECODE(vencode,'frances','Francis') name
           FROM vendor_master WHERE vencode='v001';
```

-----

查找人是否存在 加字段 decode 主明是否有人

```
select name,decode(name,'rbb','有人') from stu;
```

=====

排续

```
select dense_rank() over(partition by sex order by score) from test;
select row_number() over(order by score),name,sex,score from test;
select rank() over(order by score) from test;
select dense_rank() over(order by score) from test;
```

=====

====

创建同义词

```
SQL> create public synonym test for rbb.test;
```

```
SQL> create synonym test for mytest;
```

同一类的才可以替换,同义词替换同义词

替换

```
SQL> create or replace synonym emp_syn for scott.emp;
```

\*\*\*\*\*

\*\*\*\*\*

创建序列

```
SQL>create sequence xule increment by 1 start with 1 maxvalue 999;
```

increment by 增长值

start with 起始值

maxvalue 最大值

minvalue 最小值

nocycle 不循环

cache 10 缓存

xule.nextval =====下一个序列的值

xule.currval =====可以查询序列当前的值

更改序列 start with 不能改

```
alter sequence xule maxvalue 100 [cycle nocycle];
```

\*\*\*\*\*

\*\*\*\*\*

## 序列用法

```
SQL>create table xl(name varchar2(4));
```

```
SQL>insert into test values(xule.nextval);
```

```
SQL>select xl.currval from dual;
```

```
*****
*****
```

## 删除序列

```
drop sequence x;
```

```
desc user_sequences
```

```
*****
*****
```

## 创建视图 视图中可以使用函数和表达式

```
create or replace view
```

```
*****
*****
```

## 创建视图

```
SQL> create or replace view 视图名 as select * from rbb union all select * from rbbb union all
select * from test;
```

```
SQL> create or replace view 视图名 as
```

```
2 select empno as 编号,ename as 姓名 from scott.emp
```

```
3 where deptno=10;
```

```
=====
=====
```

如果在当前用户下没有这个视图就创建此视图

如果有此视图就覆盖此视图

```
create or replace view view_name as select empno,ename from emp where deptno=10;
```

```
*****
*****
```

在创建视图前要为当前用户授权

```
grant resource to scott;
```

```
create or replace view v_sal as select ename,sal from emp order by sal desc;
```

```
*****
*****
```

## 使用视图

```
select * from v_sal;
```

```
*****
*****
```

## 删除一个视图

```
drop view view_name;
```

```
*****
*****
```

## 重新编译已有的视图

```
alter view view_name compile;
```

```
*****
```

数据字典 =====desc user\_views

\*\*\*\*\*

常用的转换函数有

TO\_CHAR

SELECT TO\_CHAR(sysdate,'YYYY"年"fmMM"月"fmDD"日" HH24:MI:SS') FROM dual;

TO\_DATE

SELECT TO\_DATE('2005-12-06', 'yyyy-mm-dd') FROM dual;

TO\_NUMBER

SELECT TO\_NUMBER('100') FROM dual;

\*\*\*\*\*

集合操作符

union all 连接两个表或者多个表为一个视图

MINUS 操作符返回从第一个查询结果中排除第二个查询中出现的行。

INTERSECT 操作符只返回两个查询的公共行。

\*\*\*\*\*

\*\*\*\*\*

锁定的优点

- 1.一致性 - 一次只允许一个用户修改数据
- 2.完整性 - 为所有用户提供正确的数据。如果一个用户进行了修改并保存，所做的修改将反映给所有用户
- 3.并行性 - 允许多个用户访问同一数据

行级锁和表级锁

行级锁:是一种排他锁,防止其他事务修改此行.

解锁: 提交事务(commit),(rollback)

-----  
更新表数据:update test set score=80 where name='xiaoli';

-----  
自动提交

set autocommit on

set autocommit off

-----  
锁定某行更新语句

select \* from scott.test where name='xiaoli' for update;

SELECT \* FROM order\_master WHERE vencode='V002' FOR UPDATE OF odate,del\_date;  
select \* from scott.test where name='xiaoli' for update of score;

select \* from scott.test atest,test b where a.name=b.name and b.name='bbb' for update of b.score;

-----  
等待 update

select \* from scott.test where name='xiaoli' for update wait 2;  
select \* from scott.test where name='xiaoli' for update nowait;

-----  
表级锁：锁定整个表

表级锁语法:lock table 表名 in mode mode;

-----  
行共享 row share--行排他 row exclusive--共享 share-共享行排他 share row exclusive----排他 exclusive

-----  
行共享(row share):lock table scott.test in (row share) mode;

[其他用户.行共享---其他用户.行排他---其他用户.共享---其他用户.共享行排他---其他用户.不可以(排他)]

-----  
行排他(row exclusive):lock table scott.test in (row exclusive) mode;

[其他用户.行共享---其他用户.行排他---其他用户.不可以(共享)---其他用户.不可以(共享行排他)---其他用户.不可以(排他)]

-----  
共享(share):lock table scott.test in (share) mode;

[其他用户.行共享---其他用户.不可以(行排他)---其他用户.共享---其他用户.不可以(共享行排他)---其他用户.不可以(排他)]

-----  
共享行排他(share row exclusive):lock table scott.test in (share row exclusive) mode;

[其他用户.行共享,其他用户.不可以(行排他),其他用户.不可以(共享),其他用户.不可以(共享行排他),其他用户.不可以(排他)]

-----  
排他(exclusive):lock table scott.test in (exclusive) mode;

[其他用户.不可以(行共享),其他用户.不可以(行排他),其他用户.不可以(共享),其他用户.不可以(共享行排他),其他用户.不可以(排他)]

---

## 死锁

当两个事务相互等待对方释放资源时，就会形成死锁

Oracle 会自动检测死锁，并通过结束其中的一个事务来解决死锁

---

## 表分区

### ---范围分区

```
create table test(name varchar2(20),sex char(2),score number(3))
partition by range(score)
(
partition p1 values less than (50) tablespace users,
partition p2 values less than (80),
partitiom p3 values less than (maxvalue)
)
select * from test partition(p1) union select * from test partitiom(p3);
```

---

### 删除分区

```
alter table test drop partition p3;
```

### 添加分区

```
alter table test add partition p3 values less than (maxvalue);
```

### 拆分分区

```
alter table test split partition p2 at(60)
into (partition p21,partition p22);
```

### 合并分区

```
alter table test merge partitions p21,p22 into partition p2;
```

### 截断分区(删除数据)

```
alter table test truncate partition p3;
```

### 现有表分区

```
create table str as select * from student;
drop table student;
create table student(
    studentid integer not null,
    studentname varchar2(20),
    score integer
)
partition by range(score)(
    partition p1 values less than(60),
    partition p2 values less than(75),
    partition p3 values less than(85),
```

```

        partition p4 values less than(maxvalue)
    )
insert into student(select * from stu);

```

```

select * from test scott.emp@tsinghua

```

```

*****
*****

```

## 表分区

Oracle 允许用户对表进一步的规化，即对表进一步拆分，将表分成若干个逻辑部分，每个部分称其为表分区

优点：增强可用性，单个分区出现故障，不影响其他分区

均衡的 I/O，不同的分区可以映射到不同的磁盘 改善性能

```

*****
*****

```

### ①范围分区法

```

create table st(
    studentid integer not null,
    studentname varchar2(20),
    score integer
)
partition by range(score)(
    partition p1 values less than(60),
    partition p2 values less than(75),
    partition p3 values less than(85),
    partition p4 values less than(maxvalue)
)
=====select * from stu partition(p1)=====

```

### ②散列分区

```

create table st(deptno int,deptname varchar(14))
partition by hash(deptno)(
    partition p1,partition p2
)

```

### 组合分区

```

alter table test coalesce partition;

```

```

*****
*****

```

### ③复合分区

范围分区和列表分区

```

create table salgrade(

```



```

grade number(2),local number(2),hisal number(2)
)
partition by range(grade)
subpartition by list(local)
(
partition p1 values less than(10)

(
subpartition p1a values('湖北'),
subpartition p1b values(default)
),
partition p2 values less than(20)
(
subpartition p1a values('河南'),
subpartition p1b values(default)
),
partition p3 values less than(30)
(
subpartition p1a values('上海'),
subpartition p1b values(default)
)
)

```

范围分区和散列分区

```

create table salgrade(
grade number(2),local number(2),hisal number(2)
)
partition by range(grade)
subpartition by hash(local)
[subpartitions 5]
(
partition p1 values less than(10)(subpartition p1a,subpartition p1b),
partition p2 values less than(20)(subpartition p2a,subpartition p2b),
partition p3 values less than(30)(subpartition p3a,subpartition p3b)
)

```

```

-----
create table salg(
grade number(2),local number(2),hisal number(2)
)
partition by range(grade)
subpartition by hash(local)
subpartitions 3
(
partition p1 values less than(10),

```

```
partition p2 values less than(20),
partition p3 values less than(30)
)
```

```
*****
*****
```

#### ④列表分区

```
create table test stu(id int,name varchar(20),add varchar(8))
partition by list(add)
(
partition p1 values('中国'),
partition p2 values('英国'),
partition p3 values(default)
)
```

```
*****
*****
```

#### 移动分区

```
alter table test move partition p5 tablespace users;
```

```
*****
*****
```

#### 修改存档

```
SQL> shutdown immediate
```

数据库已经关闭。

已经卸载数据库。

ORACLE 例程已经关闭。

```
SQL> startup mount
```

ORACLE 例程已经启动。

```
Total System Global Area 135338868 bytes
Fixed Size                  453492 bytes
Variable Size               109051904 bytes
Database Buffers            25165824 bytes
Redo Buffers                 667648 bytes
数据库装载完毕。
```

```
SQL> alter database archivelog;
```

数据库已更改。

```
alter database open;
```

```
SQL> archive log list;
```

数据库日志模式	存档模式
自动存档	禁用
存档终点	d:\oracle\ora92\RDBMS
最早的概要日志序列	1
下一个存档日志序列	2
当前日志序列	2

```
SQL> alter system set log_archive_dest=true scope=spfile;
```

系统已更改。

```
SQL> alter database open;
```

数据库已更改。

```
SQL> spool off
```

```
*****  
*****
```

PL/SQL(过程化语言) 声明部分 执行语句部分 异常处理部分

```
identifier constant datatype not null  
[:=|default expr];
```

```
declare  
my number(5);  
begin  
    select quantity into my  
from products where product='wawa'  
for update of quantity;  
if my>0 then  
update products set quantity=quantity+1  
where product='wawa';  
insert into purchase_record  
values('wawawa',sysdate);  
end if;  
commit;  
Exception  
where others then  
dbms_output.put_line('chucuo'||SQLERRM);  
END;
```

```
declare icode varchar2(6)
p_catg varchar2(20);
c_catg constant datatype:=0.10
```

#### 数字类型

##### number

- decrmdl
- int/integer
- real(实数)
- binary\_integer(带符号的整数)
- pls\_integer(同上)

#### 字符类型

##### character

- char 3276
- Raw(2000)
- long/long Raw(32760)
- Rowid/rowid()
- varchar2 (string(nchar/nvarchar)/varchar)

#### 日期时间

##### date

- timeStamp(固定日期 dd-mm-yy 秒 6 位)
- 子 timestamp with time zone
- ti timestamp(9)

#### 布尔

##### boolean

- true
- false
- null

#### 打印出时间

##### declare

```
test_tz timestamp with time zone;
begin test_tz=to_timestamp_tz('2006-6-22 09:07:11','yyyy-mm-dd hh24:mi:ss');
dbms_output.put_line(test_tz);
end;
```

#### lob 类型

- BFILE
- BLOB
- CLOB
- NCLOB

#### 属性类型

- %type %rowtype

---

bfile 类型实例

创建目录

```
create directory tnpdir as 'c:\';
```

删除目录

```
drop directory tnpdir
```

授权

```
grant read on directory tnpdir to scott;
```

建表

```
create table bfiletest(id number(3), fname bfile);
```

添加数据

```
insert into bfiletest values(1,bfilename('TMPDIR','tmpstest.java'));
```

---

向数据库中添加图片

```
create directory images as 'c:\images';
```

```
grant read on directory images to scott;
```

```
create table my_diagrams(  
  chapter_descr varchar2(40);  
  diagram_no integer,  
  diagram blob  
);
```

declare

```
  l_bfile bfile;
```

```
  l_blob blob;
```

begin

```
  insert into my_diagrams(diagram)
```

```
  values(empty_blob())
```

```
  return diagram into l_blob;
```

```
  l_bfile:=bfilename('images','\nvimage.jpg');
```

```
  dbms_lob.open(l_bfile,dbms_lob.file_readonly);
```

```
  dbms_lob.loadfromfile(l_blob,l_bfile,dbms_lob.getlength(l_bfile));
```

```
  dbms_lob.close(l_bfile);
```

```
  commit;
```

```
end;
```

---

%type 实例 查询

declare

```
dtr dept.dname%type;
```

begin

```
select dname into str from dept where deptno=30;
```

```
dbms_output.put_line(str);
```

```
end;  
set serverout on
```

---

---

%rowtype 实例

```
declare  
row dept%rowtype;  
begin  
select * into row from dept where deptno=30;  
dbms_output.put_line(row.dname||' '||row.deptno||' '||row.loc);  
//异常  
exception  
when no_data_found then  
    dbms_output.put_line('没有数据');  
when too_many_rows(others) then  
    dbms_output.put_line('太多拉');  
end;
```

---

---

格式

if 条件 then

elsif 条件 then

else

end if

---

---

格式

```
begin  
case '&grade'  
    when 'a' then dbms_output.put_line('优异');  
    when 'b' then dbms_output.put_line('良好');  
    else dbms_output.put_line('其它')  
end case;  
end;
```

---

---

外界变量

```
var vnm varchar2(20);  
begin  
.v:='aaaaa';
```

```
end;  
打印  
print v
```

---

loop 实例

```
begin  
loop  
exit when 3>4;  
  
end loop;  
end;
```

---

while 实例

```
begin  
while (条件)condition loop  
语句体;  
end loop;  
end;
```

---

循环实例

正

```
begin  
for c in 1..10  
loop  
dbms_output.put_line(c);  
end loop  
end;
```

倒

```
begin  
for c in reverse(倒) 1..10  
loop  
dbms_output.put_line(c);  
end loop  
end;
```

---

declare

```
num number(3):=1;  
begin  
while num<10 loop  
dbms_output.put_line(num);  
num:=num+1;  
end loop;
```

```
end;
```

```
declare
```

```
    num number(3):=1;
```

```
begin
```

```
    loop
```

```
        dbms_output.put_line(num);
```

```
        exit when num>10;--退出
```

```
        num:=num+1;
```

```
    end loop;
```

```
end;
```

---

goto 实例

```
DECLARE
```

```
    qtyhand itemfile.qty_hand%type;
```

```
    relevel itemfile.re_level%type;
```

```
BEGIN
```

```
    SELECT qty_hand,re_level INTO qtyhand,relevel
```

```
    FROM itemfile WHERE itemcode = 'i201';
```

```
    IF qtyhand < relevel THEN
```

```
        GOTO updation;
```

```
    ELSE
```

```
        GOTO quit;
```

```
    END IF;
```

```
    <<updation>>
```

```
    UPDATE itemfile SET qty_hand = qty_hand + re_level
```

```
    WHERE itemcode = 'i201';
```

```
    <<quit>>
```

```
    NULL;
```

```
END;
```

---

动态 SQL 查询

```
declare
```

```
cl varchar2(20);
```

```
va varchar2(20);
```

```
tb varchar2(20);
```

```
nm number(13);
```

```
begin
```

```
tb:='&table';
```

```
cl:='&aadd';
```

```
nm:='&num';
```

```
EXECUTE IMMEDIATE
```

```
'select '||cl||' from '||tb||' where '||cl||'=:1' into va using nm;
```



```
dbms_output.put_line(va);
end;
```

---

动态 SQL

declare

```
sql_stmt varchar2(200);
emp_id number(4):=7566;
emp_rec emp% rowtype;
```

begin

Execute immediate

```
'create table bonus1(id number,amt number)';
```

```
sql_stmt:='select * from emp where empno=:id';
```

```
Execute immediate sql_stmt into emp_rec using emp_id;
```

end;

---

declare

```
aaa varchar2(20);
```

```
num number(10);
```

```
bbb varchar2(20);
```

begin

```
aaa:='&aaa';
```

```
num=&kkk;
```

```
execute immediate 'select '||aaa||' from test where age=:a'into bbb using num;(标准 SQL 语句)
```

```
dbms_output.put_line(bbb);
```

end;

into 变量(给值)

:a(外界参数) using bb(绑定常量)

---

自己定义异常

declare

```
invar exception;
```

```
cate varchar2(10);
```

begin

```
cate:='&cate';
```

```
if cate not in('aa','ff','dd') then
```

```
raise invar;
```

```

else
dbms_output.put_line('你输入类别是'||cate);
end if;
exception
when invar then
dbms_output.put_line('无法认识这个类别!');
raise_application_error(-20200,'自己写');
end;
让数据库真正出错
raise_application_error(-20200,'自己写');

```

例子 2

```

declare
rate itemfile.itemrate%type;
ratee exception;
begin
select nvl(itemrate,0) into rate from itemfile
where itemcode='i207';
if rate=0 then
raise ratee;
else
dbms_output.put_line('项费率是'||rate);
end if;
exception
when ratee then
RAISE_APPLICATION_ERROR(-20001, '未指定项费率');
end;

```

---

create procedure 存储过程

---

创建标准索引

```

SQL> CREATE INDEX item_index ON itemfile (itemcode)
TABLESPACE index_tbs;

```

重建索引

```

SQL> ALTER INDEX item_index REBUILD;

```

删除索引

```

SQL> DROP INDEX item_index;

```

唯一索引确保在定义索引的列中没有重复值

Oracle 自动在表的主键列上创建唯一索引

使用 CREATE UNIQUE INDEX 语句创建唯一索引

```

SQL> CREATE UNIQUE INDEX item_index
ON itemfile (itemcode);

```

组合索引是在表的多个列上创建的索引

索引中列的顺序是任意的

如果 SQL 语句的 WHERE 子句中引用了组合索引的所有列或大多数列，则可以提高检索速度

```
SQL> CREATE INDEX comp_index
```

```
ON itemfile(p_category, itemrate);
```

反向键索引反转索引列键值的每个字节

通常建立在值是连续增长的列上，使数据均匀地分布在整个索引上

创建索引时使用 REVERSE 关键字

```
SQL> CREATE INDEX rev_index
```

```
ON itemfile (itemcode) REVERSE;
```

```
SQL> ALTER INDEX rev_index REBUILD NOREVERSE;
```

位图索引适合创建在低基数列上

位图索引不直接存储 ROWID，而是存储字节位到 ROWID 的映射

减少响应时间

节省空间占用

```
SQL> CREATE BITMAP INDEX bit_index
```

```
ON order_master (orderno);
```

基于一个或多个列上的函数或表达式创建的索引

表达式中不能出现聚合函数

不能在 LOB 类型的列上创建

创建时必须具有 QUERY REWRITE 权限

```
SQL> CREATE INDEX lowercase_idx
```

```
ON toys (LOWER(toyname));
```

```
SQL> SELECT toyid FROM toys
```

```
WHERE LOWER(toyname)='doll';
```

与索引有关的数据字典视图有：

USER\_INDEXES — 用户创建的索引的信息

USER\_IND\_PARTITIONS — 用户创建的分区索引的信息

USER\_IND\_COLUMNS — 与索引相关的表列的信息

```
SQL> SELECT INDEX_NAME, TABLE_NAME, COLUMN_NAME  
FROM USER_IND_COLUMNS
```

```
ORDER BY INDEX_NAME, COLUMN_POSITION;
```

可以将索引存储在不同的分区中

与分区有关的索引有三种类型：

局部分区索引 — 在分区

表上创建的索引，在每个表分区上创建独立的索引，索引的分区范围与表一致

全局分区索引 — 在分区表或非分区表上创建的索引，索引单独指定分区的范围，与表的分区范围或是否分区无关

全局非分区索引 — 在分区表上创建的全局普通索引，索引没有被分区

```
SQL> CREATE TABLE ind_org_tab (
```

```
vencode NUMBER(4) PRIMARY KEY,
```

```
venname VARCHAR2(20)
```

```
)
```

ORGANIZATION INDEX;

与索引有关的数据字典视图有:

USER\_INDEXES — 用户创建的索引的信息

USER\_IND\_PARTITIONS — 用户创建的分区索引的信息

USER\_IND\_COLUMNS — 与索引相关的表列的信息

```
SQL> SELECT INDEX_NAME, TABLE_NAME, COLUMN_NAME
       FROM USER_IND_COLUMNS
       ORDER BY INDEX_NAME, COLUMN_POSITION;
```

----游标简介

逐行处理查询结果,经编程的方式访问数据

---游标类型:

隐式游标:在 PL/SQL 程序中执行 DML SQL 语句时自动创建隐式游标。

显式游标:显式游标用于处理返回多行的查询。

REF 游标:REF 游标用于处理运行时才能确定的动态 SQL 查询的结果

-----隐式游标的属性有:

%FOUND — SQL 语句影响了一行或多行时为 TRUE

%NOTFOUND — SQL 语句没有影响任何行时为 TRUE

%ROWCOUNT — SQL 语句影响的行数

%ISOPEN — 游标是否打开,始终为 FALSE

删除游标

```
delete from table_name where cursor of cursor_name;
```

-----隐式游标示例

-----too\_many\_rows 的用法!

```
1 declare
2   empid varchar2(20);
3 begin
4   select name into empid from test;
5   exception
6   when too_many_rows then
7     dbms_output.put_line('该查询多于两行!');
8* end;
```

SQL> /

该查询多于两行!

PL/SQL 过程已成功完成。

---

-----no\_data\_found 的用法!

SQL> set serverout on

SQL> ed

已写入文件 afiedt.buf

```
1  declare
2  empid varchar2(20);
3  desig varchar2(20);
4  begin
5  empid:='&emp';
6  select name into desig from test where name=empid;
7  dbms_output.put_line('你查询的名字是'||desig);
8  exception
9  when no_data_found then
10 dbms_output.put_line('没有时间! ');
11* end;
```

SQL> /

输入 emp 的值: xiaoli

原值 5: empid:='&emp';

新值 5: empid:='xiaoli';

你查询的名字是:xiaoli

PL/SQL 过程已成功完成。

SQL> /

输入 emp 的值: ss

原值 5: empid:='&emp';

新值 5: empid:='ss';

没有时间!

PL/SQL 过程已成功完成。

---

SQL> set serveroutput on

SQL> begin

```
2  update test set name='renbinbo' where name='binbo';
3  if sql%found then
4  dbms_output.put_line('表已经更新!');
5  end if;
```

```
6 end;
```

```
7 /
```

test\_t 表中 name 也已经更新!

表已经更新!

---

---

```
SQL>
```

```
declare
```

```
aa varchar2(20);
```

```
bb varchar2(20);
```

```
begin
```

```
bb:='&bb';
```

```
select score into aa from test where name=bb;
```

```
if sql%found then
```

```
dbms_output.put_line(bb||'的分数为:'||aa);
```

```
end if;
```

```
end;
```

```
SQL> /
```

输入 bb 的值: renbinbo

原值 5: bb:='&bb';

新值 5: bb:='renbinbo';

renbinbo 的分数为:100

PL/SQL 过程已成功完成。

---

---

```
SQL> ed
```

已写入文件 afiedt.buf

```
1 declare
```

```
2 my_toy rbb.test.name%type;
```

```
3 cursor toy_cur is
```

```
4 select name from test where name='xiaoli';
```

```
5 begin
```

```
6 open toy_cur;
```

```
7 loop
```

```
8 fetch toy_cur into my_toy;
```

```
9 exit when toy_cur%notfound;
```

```
10 dbms_output.put_line('你查询人的姓名:'||my_toy);
```

```
11 end loop;
```

```
12 close toy_cur;
```

```
13* end;
```

```
SQL> /
```

你查询人的姓名:xiaoli

PL/SQL 过程已成功完成。

SQL> ed

已写入文件 afiedt.buf

```
1  declare
2  name_n rbb.test.name%type;
3  sex_s rbb.test.name%type;
4  sex_t rbb.test.name%type;
5  cursor test_t is
6  select name,sex,score from test;
7  begin
8  open test_t;
9  dbms_output.put_line('你所查资料列表:');
10 loop
11 fetch test_t into name_n,sex_s,sex_t;
12 exit when test_t%notfound;
13 dbms_output.put_line(name_n||'  '||sex_s||'  '||sex_t);
14 end loop;
15 close test_t;
16* end;
17 /
```

你所查资料列表:

```
xiaoli  女   90
renbinbo 男  100
xiaoming 男   89
xiaowang 男   91
xiaohua  女   98
yunfeng  男   88
wangming 男   78
wuming  男   98
xiaobin  男   68
binbin   男   44
tianhua  女   55
liyun   女   65
```

PL/SQL 过程已成功完成。

=====

bibno-->ed

已写入文件 afiedt.buf

```
1  declare
```

```

2  cursor test_cur is
3  select name,sex,score from test;
4  begin
5  dbms_output.put_line('用户资料列表:');
6  for namet in test_cur
7  loop
8  dbms_output.put_line(namet.name||' '||namet.sex||' '||namet.score);
9  end loop;
10* end;
11 /

```

用户资料列表:

```

xiaoli 女 90
renbinbo 男 100
xiaoming 男 89
xiaowang 男 91
xiaohua 女 98
yunfeng 男 88
wangming 男 78
wuming 男 98
xiaobin 男 68
binbin 男 44
tianhua 女 55
liyun 女 65

```

PL/SQL 过程已成功完成。

=====

带参数的显式游标

SET SERVEROUTPUT ON

SQL> DECLARE

```

    desig    VARCHAR2(20);
    emp_code VARCHAR2(5);
    empnm     VARCHAR2(20);
    CURSOR emp_cur(desigparam VARCHAR2) IS
        SELECT empno, ename FROM employee
        WHERE designation=desig;

```

BEGIN

```

    desig:= '&desig';
    OPEN emp_cur(desig);
    LOOP
        FETCH emp_cur INTO emp_code,empnm;
        EXIT WHEN emp_cur%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(emp_code||' '||empnm);
    
```



```

        END LOOP;
        CLOSE emp_cur;
    END;

```

---

```

SET SERVEROUTPUT ON
SQL> DECLARE
    new_price NUMBER;
    CURSOR cur_toy IS
        SELECT toyprice FROM toys WHERE toyprice<100
        FOR UPDATE OF toyprice;
BEGIN
    OPEN cur_toy;
    LOOP
        FETCH cur_toy INTO new_price;
        EXIT WHEN cur_toy%NOTFOUND;
        UPDATE toys
        SET toyprice = 1.1*new_price
        WHERE CURRENT OF cur_toy;
    END LOOP;
    CLOSE cur_toy;
    COMMIT;
END;

```

---

游标变量的功能强大，可以简化数据处理

游标变量的优点有：

1. 可从不同的 SELECT 语句中提取结果集
2. 可以作为过程的参数进行传递
3. 可以引用游标的所有属性
4. 可以进行赋值运算

使用游标变量的限制：

1. 不能在程序包中声明游标变量
  2. FOR UPDATE 子句不能与游标变量一起使用
  3. 不能使用比较运算符
- 
- 

创建过程

```
create procedure test_b(test varchar2,test1 number)
```

```
as
begin
```

```
dbms_output.put_line(test);
dbms_output.put_line(test1);
end;
```

```
create procedure test_c(test varchar2,test1 char)
as
aa varchar2(20);
bb char(10);
begin
select name into aa from test where name=test;
dbms_output.put_line(aa);
    select age into bb from test where age=test1;
dbms_output.put_line(bb);
end;
```

---

创建函数

```
create or replace function test_binbo return varchar2
as
begin
return '我爱你!';
end
执行:
select test_binbo from dual;
```

```
create or replace function test_binbo return varchar2
as
aa varchar2(20);
bb char(3);
begin
bb:='&bb';
select name into aa from test where sex=bb;
return 'name';
end;
执行:
select test_binbo from dual;
```

```
create or replace function item_price_range(price number)
return varchar2 as
min_price number;
```

```

max_price number;
begin
select max(itemrate),min(temrate) into max_price,min_price
from test;
if price>=min_price and price<=max_price then
return '将计就计机';
else
return '哩哩啦啦理论';
end if;
end;
执行:
select test_binbo from dual;

```

---

自主事务处理

```

CREATE OR REPLACE PROCEDURE p1 AS
  b VARCHAR2(50);
BEGIN
  UPDATE vendor_master SET venadd1='10 Walls Street'
  WHERE vencode='V002';
  P2();
  SELECT venadd1 INTO b
  FROM vendor_master WHERE vencode='V002';
  DBMS_OUTPUT.PUT_LINE(b);
END;
/
执行
EXECUTE p1;

```

```

CREATE OR REPLACE PROCEDURE p2 AS
  a VARCHAR2(50);
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  SELECT venadd1 INTO a
  FROM vendor_master WHERE vencode='V002';
  DBMS_OUTPUT.PUT_LINE(a);
  ROLLBACK;
END;
/

```

---

创建程序包

SQL> ed

已写入文件 afiedt.buf

```
1  create or replace package package_me as
2  procedure proc_test(test varchar2);
3  function fun_test(funt varchar2) return varchar2;
4* end;
5  /
```

程序包已创建。

已写入文件 afiedt.buf

```
1  create or replace package body package_me as
2  procedure proc_test(test varchar2) as
3  nam varchar2(20);
4  begin
5  select name into nam from test where name=test;
6  dbms_output.put_line('你所查的人的姓名是'||nam);
7  end;
8  function fun_test(funt varchar2) return varchar2 as
9  funn varchar2(20);
10 begin
11 select next_day(funt,'星期六')-7 into funn from dual;
12 return funn;
13 end;
14* end package_me;
SQL> /
```

程序包主体已创建。

SQL> select package\_me.fun\_test('2008-10-16') from dual;

PACKAGE\_ME.FUN\_TEST('2008-10-16')

-----

2008-10-11

SQL> exec package\_me.proc\_test('xiaoli');

你所查的人的姓名是:xiaoli

PL/SQL 过程已成功完成。

```
=====
create or replace package pack_me as
procedure order_pr(orn varchar2);
function order_fu(onr varchar2) return varchar2;
end pack_me;
```

/

```
CREATE OR REPLACE PACKAGE BODY pack_me AS
  PROCEDURE order_proc (orno VARCHAR2) IS
    stat CHAR(1);
  BEGIN
    SELECT ostatus INTO stat FROM order_master
    WHERE orderno = orno;
    IF stat = 'p' THEN
      DBMS_OUTPUT.PUT_LINE('暂挂的订单');
    ELSE
      DBMS_OUTPUT.PUT_LINE('已完成的订单');
    END IF;
  END order_proc;
```

```
FUNCTION order_fun(ornos VARCHAR2)
  RETURN VARCHAR2
  IS
    icode   VARCHAR2(5);
    ocode   VARCHAR2(5);
    qtyord  NUMBER;
    qtydeld NUMBER;
  BEGIN
    SELECT qty_ord, qty_deld, itemcode, orderno
    INTO   qtyord, qtydeld, icode, ocode
    FROM order_detail
    WHERE orderno = ornos;
    IF qtyord < qtydeld THEN
      RETURN ocode;
    ELSE
      RETURN icode;
    END IF;
  END order_fun;
END pack_me;
```

/

=====

执行

```
EXECUTE pack_me.order_proc('o002');
```

```
DECLARE
  msg VARCHAR2(10);
BEGIN
```

```
msg := pack_me.order_fun('o002');
DBMS_OUTPUT.PUT_LINE('值是 ' || msg);
END;
```

/

```
CREATE OR REPLACE PACKAGE BODY cur_pack AS
  CURSOR ord_cur(vcode VARCHAR2)
  RETURN order_master%ROWTYPE IS
  SELECT * FROM order_master
  WHERE VENCODE=vcode;
  PROCEDURE ord_pro(vcode VARCHAR2) IS
    or_rec order_master%ROWTYPE;
  BEGIN
    OPEN ord_cur(vcode);
    LOOP
      FETCH ord_cur INTO or_rec;
      EXIT WHEN ord_cur%NOTFOUND;
      DBMS_OUTPUT.PUT_Line('返回的值为' || or_rec.orderno);
    END LOOP;
  END ord_pro;
END cur_pack;
```

```
EXEC cur_pack.ord_pro('V001');
```

---

```
COLUMN OBJECT_NAME FORMAT A18
```

```
SELECT object_name, object_type
FROM USER_OBJECTS
WHERE object_type IN ('PROCEDURE', 'FUNCTION',
  'PACKAGE', 'PACKAGE BODY');
```

```
DESC USER_SOURCE
```

```
COLUMN LINE FORMAT 9999
COLUMN TEXT FORMAT A50
```

```
SELECT line, text FROM USER_SOURCE
WHERE NAME='TEST';
```

```
DESC pack_me;
```

---

数据库级触发器

```
CREATE TABLE system.session_info (  
    username    VARCHAR2(30),  
    logontime   DATE,  
    session_id  VARCHAR2(30),  
    ip_addr     VARCHAR2(30),  
    hostname    VARCHAR2(30),  
    auth_type   VARCHAR2(30)  
);
```

显示

```
set serverout on
```

```
create or replace trigger trg_session_info before logoff on database
```

```
declare
```

```
session_id varchar2(30);
```

```
ip_addr     varchar2(30);
```

```
hostname    varchar2(30);
```

```
auth_type   varchar2(30);
```

```
logontime   date;
```

```
begin
```

```
    select sys_context('userenv','sessionid') -- 会话编号
```

```
        -- 用户登录的客户端 IP 地址
```

```
    select sys_context('userenv','ip_address') into ip_addr from dual;
```

```
        -- 用户登录的客户端主机名
```

```
    select sys_context('userenv','host') into hostname from dual;
```

```
        -- 登录认证方式，数据库认证或外部认证
```

```
    select sys_context('userenv','authentication_type') into auth_type from dual;
```

```
    insert into system.session_info values (user,sysdate,session_id,ip_addr,hostname,auth_type);
```

```
end;
```

```
SELECT * FROM system.session_info;
```

---

对表 employees 创建触发器

```
create or replace trigger tr_employee after update on employees
```

```
for each row
```

```
begin
```

```
    if(:new.salary>40000) then
```

```
        raise_application_error(-20002,'职员工资不能超过 40000');
```

```
    end if;
```

```
end;
```

---

```

create or replace procedure demo(salary in number) as
  cursor_name integer;
  rows_processed interger;
begin
  cursor_name:=dbms_sql.open_cursor;
  dbms_sql.parse(cursor_name,'delete          from          salary_records          where
empsal>:temp_sal',dbms_sql.native);
  dbms_sql.bind_variable(cursor_name,':temp_sal',salay);
  rows_processed:=dbms_sql.execute(cursor_name);
  dbms_sql.close_cursor(crusor_name);
exception
  when others then
    dbms_sql.close_cursor(cursor_name);
end;

```

---

1. 写一个带程序包的函数，只要传入文件名和地址就可以把这个文件的内容存到 BLOB 类型的字段中。

```

binbo>create directory tnkdir as 'c:\bfile';
binbo>grant read on directory tnkdir to scott;
binbo>CREATE TABLE my_dia
(
  chapter_descr VARCHAR2(40),
  diagram_no INTEGER,
  diagram BLOB
);

```

```

DECLARE
  l_bf  BFILE;
  l_bl  BLOB;
BEGIN
  INSERT INTO my_dia (diagram)
  VALUES (EMPTY_BLOB())
  RETURN diagram INTO l_bl;
  l_bf := BFILENAME('jsp', 'test.jsp');
  DBMS_LOB.OPEN(l_bf, DBMS_LOB.FILE_READONLY);
  DBMS_LOB.LOADFROMFILE(l_bl, l_bf, DBMS_LOB.GETLENGTH(l_bf));
  DBMS_LOB.CLOSE(l_bf);
  COMMIT;
END;

```

---

2. 有一张表，字段的值是这样的： name varchar2(20),sex char(2),score number(3)。其中的 SCORE 字段为分数字段。请用一条 SQL 语句把九十分以上的显示为 A。九十到七十分的为 B。七十分以下的为 C。



```
binbo>create table test(name varchar2(20),sex char(2),score number(3));
binbo>select name,sex,case when score<=70 then 'C'
when score<=90 and score>70 then 'B'
when score>90 then 'A'
end case from test;
```

---

3.有一个表，其中有一个字段为自动增长的数据类型。请在 ORACLE 中实现。

```
binbo>create table test(id number,name varchar2(20));
create sequence seq_test increment by 1 start with 1 maxvalue 999;
create or replace trigger tr_test before insert or update of id on test
for each row
begin
if insert into then
select seq_test.nextval into :new.id from dual;
else
raise_application_error(-20002,'不允许更新 ID 序列!');
end if;
end;
```

---

4.如何删除一个用户下的所有表。

```
binbo>spool c:\test.sql
binbo>select 'drop table '||tname||';'from tab;
binbo>spool off
binbo>@c:\test;
```

---

5. 如何把数据库的日志模式从归档模式变为非归档模式

```
binbo>shutdown immediate
binbo>startup mount
binbo>alter database archivelog;
binbo>archive log list;
binbo>alter system set log_archive_dest=false scope=spfile;
binbo>alter database open;
```

---

6. 建立一个用户和表空间，在这个用户和表空间下建立一张表。并授予 SCOTT 用户查询权利。

```
binbo>create user binbo identified by binbo;
binbo>create tablespace test datafile 'e:\test.dbf' size 10m;
binbo>GRANT SELECT ON scott.test to scott;
```

---

7. 写一个过程，计算某个月有多少天。

```
create or replace procedure dept(test in varchar2)
```

```

as
aa varchar2(20);
begin
select extract(day from last_day(to_date(test,'yyyy-mm'))) into aa from dual;
dbms_output.put_line(aa);
end;

```

---

8. 有一章表，字段为 name,sex,score，score 字段为分数字段，查询出这个班的第五名到第七名的人的姓名。

```

binbo>create table test(name varchar2(20),sex char(3),score number(3));
binbo>select * from (select name n,score sc,rownum r from (select name,score,rownum from test
order by score desc)) where r between 5 and 7;

```

---

9. 查询出当前这个星期的星期六是几号。

```

binbo>select next_day(sysdate,'星期六') from dual;

```

---

10. 做一个外键关联的两个表。然后用触发器做级联更新。

```

create table test(name varchar2(20),sex char(3),score number(3));
create table test_t(name varchar2(20));

```

```

create or replace trigger test_test before insert or update of name on test
for each row
begin
if inserting then
insert into test_t(name) values (:new.name);
dbms_output.put_line('test_t 表中 name 也已经插入!');
elseif updating then
update test_t set name=:new.name where name=:old.name;
dbms_output.put_line('test_t 表中 name 也已经更新!');
elseif deleting then
delete from test_t where name=:old.name;
dbms_output.put_line('test_t 表中 name 也已经删除!');
else
raise_application_error(-20002,'不允许更新 test 表中的 name 字段');
end if;
end;

```

---

---从外界向数据库中插入数据

```

SQL> create table test_file(name varchar(30),shell varchar2(30));
表已创建。

```

G:\盘 data.ctl:(tab 键隔开时间用 x'09')

load data into table test\_file fields terminated by '==(name,shell);

G:\盘 data.txt:

aaaaaaaa==11111111

bbbbbbbb==22222222

ccccccc==33333333

binbo==hehehehe

C:\Documents and Settings\Administrator>sqlldr rbb/rbb control=G:\data.ctl data=G:\data.txt

SQL\*Loader: Release 9.2.0.1.0 - Production on 星期二 7月 10 20:37:47 2007

Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

达到提交点，逻辑记录计数 3

达到提交点，逻辑记录计数 4

SQL> select \* from test\_file;

NAME	SHELL
aaaaaaaa	11111111
bbbbbbbb	22222222
ccccccc	33333333
binbo	hehehehe

---

```
declare
  result clob;
  xmlstr varchar2(32767);
  line varchar2(2000);
  line_no integer:=1;
begin
  result:=dbms_xmlquery.getxml('select * from test');
  xmlstr:=dbms_lob.substr(result,32767);
  loop
  exit when xmlstr is null;
  line:=substr(xmlstr,1,instr(xmlstr,chr(10))-1);
  dbms_output.put_line(line_no||'|'||line);
  xmlstr:=substr(xmlstr,instr(xmlstr,chr(10))+1);
```

```
        line_no:=line_no+1;
    end loop;
end;
SQL> /
```

PL/SQL 过程已成功完成。

```
SQL> select instr('abcdefsfssdfabcabcdfs','bc',2,2) "instring" from dual;
```

```
instring
-----
        14
```

---

---创建表中自动增长列(利用触发器)

//创建表

```
create table test_bin(id number(3),name varchar2(20));
```

//创建序列

```
create sequence test_sq increment by 1 start with 1 maxvalue 1000;
```

//创建触发器

行级触发器

```
create or replace trigger test_test before insert or update of id on test_bin
```

```
for each row
```

```
begin
```

```
if inserting then
```

```
select test_sq.nextval into :new.id from dual;
```

```
else
```

```
raise_application_error(-20002,'不允许更新 id');
```

```
end if;
```

```
end;
```

---

语句级触发器

```
create or replace trigger trgdemo after insert or update or delete on order_master
```

```
begin
```

```
if updating then
```

```
    dbms_output.put_line('已更新 order_master 中的数据');
```

```
elseif deleting then
```

```
    dbms_output.put_line('已删除 order_master 中的数据');
```

```
elseif inserting then
```

```
    dbms_output.put_line('已在 order_master 中插入数据');
```

```
end if;
```

end;

---

instead of 触发器(主要用在视图中[视图中只能有 for each row])

```
create or replace trigger upda_ord instead of update on ord_view
for each row
begin
    update order_master set vencode=new.vencode where orderno=new.orderno;
    dbms_output.put_line('已激活触发器');
end;
```

---

触发器由三部分组成:

触发器语句 (事件)

定义激活触发器的 DML 事件和 DDL 事件

触发器限制

执行触发器的条件, 该条件必须为真才能激活触发器

触发器操作 (主体)

包含一些 SQL 语句和代码, 它们在发出了触发器语句且触发限制的值是真时运行

---

触发器类型

DDL 触发器

数据库级触发器

DML 触发器

语句级触发器

行级触发器

INSTEAD OF 触发器

---

模式触发器

```
create table dropped_obj(obj_name varchar2(30),obj_type varchar2(20),drop_date date);
```

```
create or replace trigger log_drop_obj after drop on schema
begin
    insert into dropped_obj values(ora_dict_obj_name,ora_dict_obj_type,sysdate);
end;
```

---

启用和禁用触发器

```
alter trigger aiu_name disable;
```

```
alter trigger aiu_name enable;
```

---

删除触发器

```
drop trigger aiu_name;
```

---

user\_triggers 数据字典视图包含有关触发器的信息

```
select trigger_name from user_triggers where table_name='emp';
```

```
select trigger_type,triggering_event,when_clause from user_triggers
where trigger_name='biu_emp_deptno';
```

---

---

dbms\_output 包显示 pl/sql 块和子程序的调试信息

set serveroutput on

BEGIN

DBMS\_OUTPUT.PUT\_LINE('打印三角形');

FOR i IN 1..9 LOOP

FOR j IN 1..i LOOP

DBMS\_OUTPUT.PUT('\*');

END LOOP for\_j;

DBMS\_OUTPUT.NEW\_LINE;

END LOOP for\_i;

END;

打印三角形

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

PL/SQL 过程已成功完成。

---

---

1.DBMS\_LOB 包提供用于处理大型对象的过程和函数

2.DBMS\_XMLQUERY 包用于将查询结果转换为 XML 格式

DECLARE result CLOB;

xmlstr VARCHAR2(32767);

line VARCHAR2(2000);

line\_no INTEGER := 1;

BEGIN

result := DBMS\_XMLQuery.getXml('SELECT \* FROM test');

xmlstr := DBMS\_LOB.SUBSTR(result,32767);

LOOP

EXIT WHEN xmlstr IS NULL;

line := SUBSTR(xmlstr,1,INSTR(xmlstr,CHR(10))-1);

DBMS\_OUTPUT.PUT\_LINE(line\_no || ':' || line);

xmlstr := SUBSTR(xmlstr,INSTR(xmlstr,CHR(10))+1);

```
    line_no := line_no + 1;
END LOOP;
END;
```

---

一些常用的内置程序包：

DBMS\_OUTPUT 包输出 PL/SQL 程序的调试信息  
DBMS\_LOB 包提供操作 LOB 数据的子程序  
DBMS\_XMLQUERY 将查询结果转换为 XML 格式  
DBMS\_RANDOM 提供随机数生成器  
UTL\_FILE 用于读写操作系统文本文件

---

触发器

```
CREATE OR REPLACE TRIGGER biu_emp_deptno
BEFORE INSERT OR UPDATE OF deptno
ON emp
FOR EACH ROW
WHEN (New.deptno <> 40)
BEGIN
    :New.comm := 0;
END;
```

/

触发器已创建

---

----没有表还不能测试

```
CREATE VIEW ord_view AS
SELECT order_master.orderno, order_master.ostatus,
       order_detail.qty_deld, order_detail.qty_ord
FROM order_master, order_detail
WHERE order_master.orderno = order_detail.orderno;
```

```
CREATE OR REPLACE TRIGGER order_mast_insert
INSTEAD OF INSERT ON ord_view
REFERENCING NEW AS n
FOR EACH ROW
DECLARE
    CURSOR ecur IS SELECT * FROM order_master
                   WHERE order_master.orderno = :n.orderno;
    CURSOR dcur IS SELECT * FROM order_detail
                   WHERE order_detail.orderno = :n.orderno;
```

```

a ecur%rowtype;
b dcur%rowtype;
BEGIN
  OPEN ecur;
  OPEN dcur;
  FETCH ecur into a;
  FETCH dcur into b;
  IF dcur%notfound THEN
    INSERT INTO order_master(orderno,ostatus)
    VALUES(:n.orderno, :n.ostatus);
  ELSE
    UPDATE order_master SET order_master.ostatus = :n.ostatus
    WHERE order_master.orderno = :n.orderno;
  END IF;
  IF ecur%notfound THEN
    INSERT INTO order_detail(qty_ord,qty_deld,orderno)
    VALUES(:n.qty_ord, :n.qty_deld, :n.orderno);
  ELSE
    UPDATE order_detail
    SET order_detail.qty_ord = :n.qty_ord,
        order_detail.qty_deld = :n.qty_deld
    WHERE order_detail.orderno = :n.orderno;
  END IF;
  CLOSE ecur;
  CLOSE dcur;
END;
/

=====

CREATE TABLE dropped_obj
(
  obj_name VARCHAR2(30),
  obj_type VARCHAR2(20),
  drop_date DATE
);

CREATE OR REPLACE TRIGGER log_drop_obj
AFTER DROP ON SCHEMA
BEGIN
  INSERT INTO dropped_obj
  VALUES (ORA_DICT_OBJ_NAME, ORA_DICT_OBJ_TYPE, SYSDATE);
END;
/

```



```
ALTER TRIGGER biu_emp_deptno DISABLE;
```

```
ALTER TRIGGER biu_emp_deptno ENABLE;
```

```
DROP TRIGGER biu_emp_deptno;
```

```
DESC USER_TRIGGERS;
```

```
=====
```

```
DECLARE
```

```
  l_num    NUMBER;
```

```
  counter  NUMBER;
```

```
BEGIN
```

```
  counter:=1;
```

```
  WHILE counter <= 10
```

```
  LOOP
```

```
    l_num := ABS((DBMS_RANDOM.RANDOM MOD 100)) + 1;
```

```
    DBMS_OUTPUT.PUT_LINE(l_num);
```

```
    counter := counter + 1;
```

```
  END LOOP;
```

```
END;
```

```
/
```

```
40
```

```
4
```

```
35
```

```
52
```

```
68
```

```
5
```

```
94
```

```
38
```

```
49
```

```
51
```

```
PL/SQL 过程已成功完成。
```

```
=====
```

```
---查询出表中数据转换为 xml 格式
```

```
-- 以 SYSTEM 用户登录执行命令
```

```
CREATE DIRECTORY TEST_DIR AS 'C:\DEVELOP';
```

```
GRANT READ, WRITE ON DIRECTORY TEST_DIR TO SCOTT;
```

```
-- 以 SCOTT 用户登录
```

```
DECLARE
```

```
  src CLOB;
```

```

xmlfile UTL_FILE.FILE_TYPE;
length INTEGER;
buffer VARCHAR2(16384);
BEGIN
    src := DBMS_XMLQuery.getXml('select * from emp');
    length := DBMS_LOB.GETLENGTH(src);
    DBMS_LOB.READ(src, length, 1, buffer);
    xmlfile := UTL_FILE.FOPEN('TEST_DIR', 'employees.xml', 'w');
    UTL_FILE.PUT(xmlfile, buffer);
    UTL_FILE.FCLOSE(xmlfile);
END;
/

```

---

-----读取 xml 格式的文件

UTL\_FILE 包用于读写操作系统文本文件

操作文件的一般过程是打开、读或写、关闭

UTL\_FILE 包指定文件路径依赖于 DIRECTORY 对象

1.CREATE DIRECTORY TEST\_DIR AS 'C:\DEVELOP';

2.GRANT READ, WRITE ON DIRECTORY TEST\_DIR TO SCOTT

SET SERVEROUT ON FORMAT WRAPPED

DECLARE

input\_file UTL\_FILE.FILE\_TYPE;

input\_buffer VARCHAR2(4000);

BEGIN

input\_file := UTL\_FILE.FOPEN('TEST\_DIR', 'employees.xml', 'r');

FOR I IN 1..11 LOOP

UTL\_FILE.GET\_LINE(input\_file, input\_buffer);

DBMS\_OUTPUT.PUT\_LINE(input\_buffer);

END LOOP;

UTL\_FILE.FCLOSE(input\_file);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('-----');

END;

---

oracle 数据库重生\$rman

oracle 数据库备份两种类型:物理备份和逻辑备份

---

oracle 故障有 4 种类型

1.语句故障

在执行 SQL 语句过程中发生的逻辑故障可导致语句故障。如果用户编写的 SQL 语句无效，就会发生逻辑故障

## 2.用户进程故障

当用户程序出错而无法访问数据库时发生用户进程故障。导致用户进程故障的原因是异常断开连接或异常终止进程

## 3. 实例故障

当 Oracle 的数据库实例由于硬件或软件问题而无法继续运行时，就会发生实例故障

## 4.介质故障

在数据库无法正确读取或写入某个数据库文件时，会发生介质故障

---

## oracle 导出导入模式

### 1.完全数据库

导出和导入整个数据库中的所有对象

### 2.表

导出和导入一个或多个指定的表或表分区

### 3.用户

导出和导入一个用户模式中的所有对象

### 4.表空间

导出和导入一个或多个指定的表空间中的所有对象

---

导出实用程序有以下常用命令参数

**USERID** 确定执行导出实用程序的用户名和口令

**BUFFER** 确定导出数据时所使用的缓冲区大小，其大小用字节表示

**FILE** 指定导出的二进制文件名称，默认的扩展名是.dmp

**FULL** 指定是否以全部数据库方式导出，只有授权用户才可使用此参数

**OWNER** 要导出的数据库用户列表

**HELP** 指定是否显示帮助消息和参数说明

**ROWS** 确定是否要导出表中的数据

**TABLES** 按表方式导出时，指定需导出的表和分区的名称

**PARFILE**指定传递给导出实用程序的参数文件名

**TABLESPACES** 按表空间方式导出时，指定要导出的表空间名

按用户方式导出数据

```
exp scott/tiger@tsinghua file=scott_back owner=scott
```

按表方式导出数据

```
exp scott/tiger@tsinghua tables=(emp, dept) file=scott_back_tab
```

按表空间方式导出数据

```
exp system/zl@tsinghua tablespaces=(users) file=tbs_users
```

使用参数文件导出数据

```
exp system/zl parfile='C:\parameters.txt'
```

---

导入实用程序有如下常用命令参数

**USERID** 指定执行导入的用户名和密码

**BUFFER** 指定用来读取数据的缓冲区大小，以字节为单位

**COMMIT** 指定是否在每个数组（其大小由 **BUFFER** 参数设置）插入后进行提交

**FILE** 指定要导入的二进制文件名

**FROMUSER** 指定要从导出转储文件中导入的用户模式

**TOUSER** 指定要将对象导入的用户名。**FROMUSER** 与 **TOUSER** 可以不同

**FULL** 指定是否要导入整个导出转储文件

**TABLES** 指定要导入的表的列表

**ROWS** 指定是否要导入表中的行

**PARFILE** 指定传递给导入实用程序的参数文件名，此文件可以包含这里列出的所有参数

**IGNORE** 导入时是否忽略遇到的错误，默认为 N

**TABLESPACES** 按表空间方式导入，列出要导入的表空间名

将整个文件导入数据库

```
imp tsinghua/tsinghua@tsinghua file=item_back.dmp ignore=y full=y
```

将 scott 用户的表导入到 martin 用户

```
imp system/zl@tsinghua file=scott_back fromuser=scott touser=martin tables=(emp,dept)
```

使用参数文件导入数据

```
imp system/oracle parfile='C:\parameters.txt'
```

---

归档日志方式下的数据库:自动归档和手动归档

手动归档允许用户手动归档非活动日志文件文件的已填充组

自动归档对非活动日志文件文件进行自动归档

---

数据库可在两种方式下运行：非归档日志方式和归档日志方式

非归档日志方式可以避免实例故障，但无法避免介质故障。在此方式下，数据库只能实施冷备份。

归档日志方式产生归档日志，用户可以使用归档日志完全恢复数据库。

---

导出和导入实用程序的特点有：

可以按时间保存表结构和数据

- 1.允许导出指定的表，并重新导入到新的数据库中
  - 2.可以把数据库迁移到另外一台异构服务器上
  - 3.在两个不同版本的 Oracle 数据库之间传输数据
  - 4.在联机状态下进行备份和恢复
  - 5.可以重新组织表的存储结构，减少链接及磁盘碎片
- 

oracle 表输格式为 xml 全过程.txt

```
SQL> conn sys/sys as sysdba
```

已连接。

```
SQL> drop directory test_dir
```

```
2 /
```

目录已丢弃。

```
SQL> CREATE DIRECTORY TEST_DIR AS 'C:\';
```

目录已创建。

```
SQL> GRANT READ, WRITE ON DIRECTORY TEST_DIR TO rbb;
```

授权成功。

```
SQL> conn rbb/rbb
```

已连接。

```
SQL> ed
```

已写入文件 afiedt.buf

```
1  DECLARE
2      src CLOB;
3      xmlfile UTL_FILE.FILE_TYPE;
4      length INTEGER;
5      buffer VARCHAR2(16384);
6  BEGIN
7      src := DBMS_XMLQuery.getXml('select * from liuxing');
8      length := DBMS_LOB.GETLENGTH(src);
9      DBMS_LOB.READ(src, length, 1, buffer);
10     xmlfile := UTL_FILE.FOPEN('TEST_DIR', 'emp.xml', 'w');
11     UTL_FILE.PUT(xmlfile, buffer);
12     UTL_FILE.FCLOSE(xmlfile);
13* END;
SQL> /
```

PL/SQL 过程已成功完成。

---

SQL> ed

已写入文件 afiedt.buf

```
1  declare
2  lname number;
3  counter number;
4  begin
5  counter:=1;
6  while counter<=10
7  loop
8  lname:=dbms_random.random;
9  dbms_output.put_line(lname);
10 counter:=counter+1;
11 end loop;
12* end;
```

SQL> /

```
277652640
-479979827
-1049652647
-1006595853
1252280346
196435204
466478280
-85782435
-1489036577
-927786638
```

PL/SQL 过程已成功完成。

已用时间: 00: 00: 00.00

---

修改表名

```
alter table old_table_name rename to new_table_name;
```

---

估算 SQL 执行的 I/O 数

SQL>SET AUTOTRACE ON ;

SQL>SELECT \* FROM TABLE;

OR

```
SQL>SELECT * FROM v$filestat ;
```

---

如何查有多少个数据库实例

```
SQL>SELECT * FROM V$INSTANCE;
```

---

查询数据库有多少表

```
SQL>select * from all_tables;
```

---

显示测试 SQL 语句执行所用的时间

```
SQL>set timing on ;
```

---

监控事例的等待

```
select event,sum(decode(wait_Time,0,0,1)) "Prev",  
sum(decode(wait_Time,0,1,0)) "Curr",count(*) "Tot"  
from v$session_Wait  
group by event order by 4;
```

---

回滚段的争用情况

```
select name, waits, gets, waits/gets "Ratio"  
from v$rollstat C, v$rollname D  
where C.usn = D.usn;
```

---

监控表空间的 I/O 比例

```
select B.tablespace_name name,B.file_name "file",A.phyrds pyr,  
A.phyblkrd pbr,A.phywrt pyw,A.phyblkwrt pbw  
from v$filestat A, dba_data_files B  
where A.file# = B.file_id  
order by B.tablespace_name;
```

---

监控文件系统的 I/O 比例

```
select substr(C.file#,1,2) "#", substr(C.name,1,30) "Name",  
C.status, C.bytes, D.phyrds, D.phywrt  
from v$datafile C, v$filestat D
```

```
where C.file# = D.file#;
```

---

在某个用户下找所有的索引

```
select user_indexes.table_name, user_indexes.index_name, uniqueness, column_name
from user_ind_columns, user_indexes
where user_ind_columns.index_name = user_indexes.index_name
and user_ind_columns.table_name = user_indexes.table_name
order by user_indexes.table_type, user_indexes.table_name,
user_indexes.index_name, column_position;
```

---

监控 SGA 的命中率

```
select a.value + b.value "logical_reads", c.value "phys_reads",
round(100 * ((a.value+b.value)-c.value) / (a.value+b.value)) "BUFFER HIT RATIO"
from v$sysstat a, v$sysstat b, v$sysstat c
where a.statistic# = 38 and b.statistic# = 39
and c.statistic# = 40;
```

---

监控 SGA 中字典缓冲区的命中率

```
select parameter, gets, Getmisses, getmisses/(gets+getmisses)*100 "miss ratio",
(1-(sum(getmisses)/ (sum(gets)+sum(getmisses))))*100 "Hit ratio"
from v$rowcache
where gets+getmisses <> 0
group by parameter, gets, getmisses;
```

---

监控 SGA 中共享缓存区的命中率，应该小于 1%

```
select sum(pins) "Total Pins", sum(reloads) "Total Reloads",
sum(reloads)/sum(pins) *100 libcache
from v$librarycache;
```

```
select sum(pinhits-reloads)/sum(pins) "hit radio",sum(reloads)/sum(pins) "reload percent"
from v$librarycache;
```

---

显示所有数据库对象的类别和大小

```
select count(name) num_instances, type, sum(source_size) source_size,
sum(parsed_size) parsed_size, sum(code_size) code_size, sum(error_size) error_size,
sum(source_size) +sum(parsed_size) +sum(code_size) +sum(error_size) size_required
from dba_object_size
group by type order by 2;
```

---

监控 SGA 中重做日志缓存区的命中率，应该小于 1%

```
SELECT name, gets, misses, immediate_gets, immediate_misses,
Decode(gets,0,0,misses/gets*100) ratio1,
Decode(immediate_gets+immediate_misses,0,0,
immediate_misses/(immediate_gets+immediate_misses)*100) ratio2
```



```
FROM v$latch WHERE name IN ('redo allocation', 'redo copy');
```

---

监控内存和硬盘的排序比率，最好使它小于 .10，增加 sort\_area\_size

```
SELECT name, value FROM v$sysstat WHERE name IN ('sorts (memory)', 'sorts (disk)');
```

---

监控当前数据库谁在运行什么 SQL 语句

```
SELECT osuser, username, sql_text from v$session a, v$sqltext b
where a.sql_address =b.address order by address, piece;
```

---

监控字典缓冲区

```
SELECT (SUM(PINS - RELOADS)) / SUM(PINS) "LIB CACHE" FROM V$LIBRARYCACHE;
SELECT (SUM(GETS - GETMISSES - USAGE - FIXED)) / SUM(GETS) "ROW CACHE"
FROM V$ROWCACHE;
SELECT SUM(PINS) "EXECUTIONS", SUM(RELOADS) "CACHE MISSES WHILE
EXECUTING" FROM V$LIBRARYCACHE;
```

---

后者除以前者,此比率小于 1%,接近 0%为好。

```
SELECT SUM(GETS) "DICTIONARY GETS",SUM(GETMISSES) "DICTIONARY CACHE
GET MISSES"
FROM V$ROWCACHE
```

---

监控 MTS

```
select busy/(busy+idle) "shared servers busy" from v$dispatcher;
```

此值大于 0.5 时，参数需加大

```
select sum(wait)/sum(totalq) "dispatcher waits" from v$queue where type='dispatcher';
```

```
select count(*) from v$dispatcher;
```

```
select servers_highwater from v$mmts;
```

servers\_highwater 接近 mts\_max\_servers 时，参数需加大

---

知道当前用户的 ID 号

```
SQL>SHOW USER;
```

OR

```
SQL>select user from dual;
```

---

查看碎片程度高的表

```
SELECT segment_name table_name , COUNT(*) extents
FROM dba_segments WHERE owner NOT IN ('SYS', 'SYSTEM') GROUP BY segment_name
HAVING COUNT(*) = (SELECT MAX( COUNT(*) ) FROM dba_segments GROUP BY
segment_name);
```

---

知道表在表空间中的存储情况

```
select segment_name,sum(bytes),count(*) ext_quan from dba_extents where  
tablespace_name='&tablespace_name' and segment_type='TABLE' group by  
tablespace_name,segment_name;
```

---

知道索引在表空间中的存储情况

```
select segment_name,count(*) from dba_extents where segment_type='INDEX' and  
owner='&owner'  
group by segment_name;
```

---

知道使用 CPU 多的用户 session 11 是 cpu used by this session

```
select a.sid,spid,status,substr(a.program,1,40) prog,a.terminal,osuser,value/60/100 value  
from v$session a,v$process b,v$sesstat c  
where c.statistic#=11 and c.sid=a.sid and a.paddr=b.addr order by value desc;
```

---

5、说明：

[illegible]

增加一个列: `Alter table tabname add column col type`  
`alter table tabanme add column col`

[illegible]

添加主键: `Alter table tabname add primary key(col)`

删除主键: `Alter table tabname drop primary key(col)`

[illegible]

创建索引: `create [unique] index idxname on tablename(col...`)

注：索引是不可更改的，想更改必须删除重新建。

[illegible]

创建视图: create view viewname as select statement

删除视图: `drop view viewname`

[illegible]

## 10、说明：几个简单的基本的 sql 语句

选择: `select * from table1 where 范围`

插入: insert into table1(field1,field2) values(value1,value2)

删除: delete from table1 where 范围

更新: update table1 set field1=value1 where 范围

查找: `select * from table1 where field1 like ' %value1%'` ---like 的语法很精妙, 查资料!

排序: `select * from table1 order by field1,field2 [desc]`

总数: select count \* as totalcount from table1

求和: `select sum(field1) as sumvalue from table1`

平均: `select avg(field1) as avgvalue from table1`

最大: `select max(field1) as maxvalue from table1`

最小: `select min(field1) as minvalue from table1`

[illegible]

## 11、说明：几个高级查询运算词

下列语句创建 STAFF 表中 20 部门的非经理人员视图,其中薪水和佣金不通过基表显示。

```
CREATE VIEW STAFF_ONLY
```

AS SELECT ID, NAME, DEPT, JOB, YEARS

FROM STAFF

WHERE JOB <> 'Mgr' AND DEPT=20

在创建视图之后，下列语句显示视图的内容：

```
SELECT * FROM STAFF ONLY
```

A: UNION union 运算符

UNION 运算符通过组合其他两个结果表（例如 TABLE1 和 TABLE2）并消去表中任何重复行而派生出一个结果表。当 ALL 随 UNION 一起使用时（即 UNION ALL），不消除重复行。两种情况下，派生表的每一行不是来自 TABLE1 就是来自 TABLE2。

B: EXCEPT except 运算符

EXCEPT 运算符通过包括所有在 TABLE1 中但不在 TABLE2 中的行并消除所有重复行而派生出一个结果表。当 ALL 随 EXCEPT 一起使用时 (EXCEPT ALL), 不消除重复行。

### C: INTERSECT intersect 运算符

INTERSECT 运算符通过只包括 TABLE1 和 TABLE2 中都有的行并消除所有重复行而派生出一个结果表。当 ALL 随 INTERSECT 一起使用时 (INTERSECT ALL), 不消除重复行。

注：使用运算词的几个查询结果行必须是一致的。

12、说明：使用外连接

A、left outer join:

左外连接（左连接）：结果集既包括连接表的匹配行，也包括左连接表的所有行。

SQL: select a.a, a.b, a.c, b.c, b.d, b.f from a LEFT OUT JOIN b ON a.a = b.c

B: right outer join:

右外连接(右连接)：结果集既包括连接表的匹配连接行，也包括右连接表的所有行。

C: full outer join:

全外连接：不仅包括符号连接表的匹配行，还包括两个连接表中的所有记录。

其次，大家来看一些不错的 sql 语句

1、说明：复制表(只复制结构,源表名: a 新表名: b) (Access 可用)

法一: select \* into b from a where 1<>1

法二: select top 0 \* into b from a

2、说明：拷贝表(拷贝数据,源表名: a 目标表名: b) (Access 可用)

insert into b(a, b, c) select d,e,f from b;

3、说明：跨数据库之间表的拷贝(具体数据使用绝对路径) (Access 可用)

insert into b(a, b, c) select d,e,f from b in '具体数据库' where 条件

例子: ..from b in "&Server.MapPath(".")&"\data.mdb" &" where..

4、说明：子查询(表名 1: a 表名 2: b)

select a,b,c from a where a IN (select d from b) 或者: select a,b,c from a where a IN (1,2,3)

5、说明：显示文章、提交人和最后回复时间

select a.title,a.username,b.adddate from table a,(select max(adddate) adddate from table where

table.title=a.title) b

6、说明：外连接查询(表名 1: a 表名 2: b)

select a.a, a.b, a.c, b.c, b.d, b.f from a LEFT OUT JOIN b ON a.a = b.c

7、说明：在线视图查询(表名 1: a)

select \* from (SELECT a,b,c FROM a) T where t.a > 1;

8、说明：between 的用法,between 限制查询数据范围时包括了边界值,not between 不包括

select \* from table1 where time between time1 and time2

select a,b,c, from table1 where a not between 数值 1 and 数值 2

9、说明：in 的使用方法

select \* from table1 where a [not] in ( '值 1' , '值 2' , '值 4' , '值 6' )

10、说明：两张关联表，删除主表中已经在副表中没有的信息

delete from table1 where not exists ( select \* from table2 where table1.field1=table2.field1 )

11、说明：四表联查问题：

select \* from a left inner join b on a.a=b.b right inner join c on a.a=c.c inner join d on a.a=d.d where  
.....

12、说明：日程安排提前五分钟提醒

SQL: select \* from 日程安排 where datediff('minute',f 开始时间,getdate())>5

13、说明：一条 sql 语句搞定数据库分页

select top 10 b.\* from (select top 20 主键字段,排序字段 from 表名 order by 排序字段 desc) a,  
表名 b where b.主键字段 = a.主键字段 order by a.排序字段

14、说明：前 10 条记录

select top 10 \* form table1 where 范围

15、说明：选择在每一组 b 值相同的数据中对应的 a 最大的记录的所有信息(类似这样

的用法可以用于论坛每月排行榜,每月热销产品分析,按科目成绩排名,等等.)

```
select a,b,c from tablename ta where a=(select max(a) from tablename tb where tb.b=ta.b)
```

16、说明: 包括所有在 TableA 中但不在 TableB 和 TableC 中的行并消除所有重复行而派生出一个结果表

```
(select a from tableA ) except (select a from tableB) except (select a from tableC)
```

17、说明: 随机取出 10 条数据

```
select top 10 * from tablename order by newid()
```

18、说明: 随机选择记录

```
select newid()
```

19、说明: 删除重复记录

```
Delete from tablename where id not in (select max(id) from tablename group by col1,col2,...)
```

20、说明: 列出数据库里所有的表名

```
select name from sysobjects where type='U'
```

21、说明: 列出表里的所有的

```
select name from syscolumns where id=object_id('TableName')
```

22、说明: 列示 type、vender、pcs 字段, 以 type 字段排列, case 可以方便地实现多重选择, 类似 select 中的 case。

```
select type,sum(case vender when 'A' then pcs else 0 end),sum(case vender when 'C' then pcs else 0 end),sum(case vender when 'B' then pcs else 0 end) FROM tablename group by type
```

显示结果:

type	vender	pcs
电脑	A	1
电脑	A	1
光盘	B	2
光盘	A	2
手机	B	3
手机	C	3



23、说明：初始化表 table1

```
TRUNCATE TABLE table1
```

24、说明：选择从 10 到 15 的记录

```
select top 5 * from (select top 15 * from table order by id asc) table_别名 order by id desc
```

随机选择数据库记录的方法（使用 Randomize 函数，通过 SQL 语句实现）

对存储在数据库中的数据来说，随机数特性能给出上面的效果，但它们可能太慢了。你不能要求 ASP “找个随机数” 然后打印出来。实际上常见的解决方案是建立如下所示的循环：

```
Randomize
RNumber = Int(Rnd*499) + 1

While Not objRec.EOF
If objRec("ID") = RNumber THEN
... 这里是执行脚本 ...
end if
objRec.MoveNext
Wend
```

这很容易理解。首先，你取出 1 到 500 范围内的一个随机数（假设 500 就是数据库内记录的总数）。然后，你遍历每一记录来测试 ID 的值、检查其是否匹配 RNumber。满足条件的话就执行由 THEN 关键字开始的那一块代码。假如你的 RNumber 等于 495，那么要循环一遍数据库花的时间可就长了。虽然 500 这个数字看起来大了些，但相比更为稳固的企业解决方案这还是个小型数据库了，后者通常在一个数据库内就包含了成千上万条记录。这时候不就死定了？

采用 SQL，你就可以很快地找出准确的记录并且打开一个只包含该记录的 recordset，如下所示：

```
Randomize
RNumber = Int(Rnd*499) + 1

SQL = "SELECT * FROM Customers WHERE ID = " & RNumber

set objRec = ObjConn.Execute(SQL)
Response.WriteRNumber & " = " & objRec("ID") & " " & objRec("c_email")
```

不必写出 RNumber 和 ID，你只需要检查匹配情况即可。只要你对以上代码的工作满意，

你自可按需操作“随机”记录。Recordset 没有包含其他内容，因此你很快就能找到你需要的记录这样就大大降低了处理时间。

### 再谈随机数

现在你下定决心要榨干 Random 函数的最后一滴油，那么你可能会一次取出多条随机记录或者想采用一定随机范围内的记录。把上面的标准 Random 示例扩展一下就可以用 SQL 应对上面两种情况了。

为了取出几条随机选择的记录并存放在同一 recordset 内，你可以存储三个随机数，然后查询数据库获得匹配这些数字的记录：

```
SQL = "SELECT * FROM Customers WHERE ID = " & RNumber & " OR ID = " & RNumber2 & " OR ID = " & RNumber3
```

假如你想选出 10 条记录（也许是每次页面装载时的 10 条链接的列表），你可以用 BETWEEN 或者数学等式选出第一条记录和适当数量的递增记录。这一操作可以通过好几种方式来完成，但是 SELECT 语句只显示一种可能（这里的 ID 是自动生成的号码）：  
SQL = "SELECT \* FROM Customers WHERE ID BETWEEN " & RNumber & " AND " & RNumber & "+ 9"

注意：以上代码的执行目的不是检查数据库内是否有 9 条并发记录。

随机读取若干条记录，测试过

Access 语法：SELECT top 10 \* From 表名 ORDER BY Rnd(id)

Sql server:select top n \* from 表名 order by newid()

mysql select \* From 表名 Order By rand() Limit n

Access 左连接语法(最近开发要用左连接,Access 帮助什么都没有,网上没有 Access 的 SQL 说明,只有自己测试, 现在记下以备后查)

语 法      select table1.fd1,table1.fd2,table2.fd2 From table1 left join table2 on table1.fd1,table2.fd1 where ...

使用 SQL 语句 用...代替过长的字符串显示

语法：

SQL 数据库： select case when len(field)>10 then left(field,10)+'...' else field end as news\_name,news\_id from tablename

Access 数据库： SELECT iif(len(field)>2,left(field,2)+'...',field) FROM tablename;

Conn.Execute 说明

## Execute 方法

该方法用于执行 SQL 语句。根据 SQL 语句执行后是否返回记录集，该方法的使用格式分为以下两种：

1. 执行 SQL 查询语句时，将返回查询得到的记录集。用法为：

Set 对象变量名=连接对象.Execute("SQL 查询语言")

Execute 方法调用后，会自动创建记录集对象，并将查询结果存储在该记录对象中，通过 Set 方法，将记录集赋给指定的对象保存，以后对象变量就代表了该记录集对象。

2. 执行 SQL 的操作性语言时，没有记录集的返回。此时用法为：

连接对象.Execute "SQL 操作性语句" [, RecordAffected][, Option]

- RecordAffected 为可选项，此出可放置一个变量，SQL 语句执行后，所生效的记录数会自动保存到该变量中。通过访问该变量，就可知道 SQL 语句队多少条记录进行了操作。

- Option 可选项，该参数的取值通常为 adCMDText，它用于告诉 ADO，应该将 Execute 方法之后的第一个字符解释为命令文本。通过指定该参数，可使执行更高效。

- BeginTrans、RollbackTrans、CommitTrans 方法

这三个方法是连接对象提供的用于事务处理的方法。BeginTrans 用于开始一个事物；RollbackTrans 用于回滚事务；CommitTrans 用于提交所有的事务处理结果，即确认事务的处理。

事务处理可以将一组操作视为一个整体，只有全部语句都成功执行后，事务处理才算成功；若其中有一个语句执行失败，则整个处理就算失败，并恢复到处里前的状态。

BeginTrans 和 CommitTrans 用于标记事务的开始和结束，在这两个之间的语句，就是作为事务处理的语句。判断事务处理是否成功，可通过连接对象的 Error 集合来实现，若 Error 集合的成员个数不为 0，则说明有错误发生，事务处理失败。Error 集合中的每一个 Error 对象，代表一个错误信息。

\*\*\*\*\* 各 种 数 据 库 连 接 方 法  
\*\*\*\*\*

=====TestSQLServer.java 连接 SQLServer=====

```
import java.sql.*;
import java.util.*;
public class TestSQLServer {
    public static void main (String args[]) {
        try {
            Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
            String url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=testdb";
            String user = "sa";
            String password = "sa";
            Connection conn = DriverManager.getConnection(url,user,password);
            Statement stat = conn.createStatement();
            ResultSet result = stat.executeQuery("SELECT * FROM users");
            result.next();
            System.out.println(result.getString(1));
            System.out.println(result.getString(2));
            System.out.println(result.getString(3));
            result.close();
            stat.close();
            conn.close();
        }
    }
}
```

```

    } catch(ClassNotFoundException en){
        System.out.println("数据库驱动找不到！");
        en.printStackTrace();
    } catch(SQLException ex) {
        while (ex != null) {
            ex.printStackTrace();
            ex = ex.getNextException();
        }
    } catch(Exception e){
        System.out.println("其他未知异常！");
        e.printStackTrace();
    }
}
}
}

```

=====TestMysql.java 连接 mysql 数据库=====

```

package org.binbo.dom;
import java.sql.*;
public class TestMysql{
    public static void main (String args[]){
        try{
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            String url = "jdbc:mysql://localhost:3306/binbo";
            String user="root";
            String password= "binbo";
            Connection conn = DriverManager.getConnection(url,user,password);

            Statement stat = conn.createStatement();
            ResultSet result = stat.executeQuery("SELECT * FROM testxml");
            result.next();
            System.out.println(result.getString(1));
            System.out.println(result.getString(2));
            System.out.println(result.getString(3));
            result.close();
            stat.close();
            conn.close();
        } catch(ClassNotFoundException en){
            System.out.println("数据库驱动找不到！");
            en.printStackTrace();
        } catch(SQLException ex) {
            while (ex != null) {
                ex.printStackTrace();
            }
        }
    }
}

```

```

        ex = ex.getNextException();
    }
} catch (Exception e) {
    System.out.println("其他未知异常！");
    e.printStackTrace();
}
}

}

=====TestOracle.java 连接 oracle 数据库=====

```

```

package org.binbo.dom;
import java.sql.*;
public class TestOracle{
    public static void main (String args[]) {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
            String url = "jdbc:oracle:thin:@localhost:1521:binbo";
            String user = "scott";
            String password = "tiger";
            Connection conn = DriverManager.getConnection(url,user,password);

            Statement stat = conn.createStatement();
            ResultSet result = stat.executeQuery("SELECT * FROM test");
            result.next();
            System.out.println(result.getString(1));
            System.out.println(result.getString(2));
            System.out.println(result.getString(3));
            result.close();
            stat.close();
            conn.close();
        } catch (ClassNotFoundException en) {
            System.out.println("数据库驱动找不到！");
            en.printStackTrace();
        } catch (SQLException ex) {
            while (ex != null) {
                ex.printStackTrace();
                ex = ex.getNextException();
            }
        } catch (Exception e) {
            System.out.println("其他未知异常！");
            e.printStackTrace();
        }
    }
}

```

```
}
```

=====XML 连接数据库=====

da.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<PEOPLE><!--
  <PERSON PERSONID="E01">
    <className>com.microsoft.jdbc.sqlserver.SQLServerDriver</className>
    <url>jdbc:microsoft.sqlserver://localhost:1433;DatabaseName=binbo</url>
    <user>sa</user>
    <password>sa</password>
  </PERSON>
  --><PERSON PERSONID="E02">
    <className>com.mysql.jdbc.Driver</className>
    <url>jdbc:mysql://localhost:3306/binbo</url>
    <user>root</user>
    <password>binbo</password>
  </PERSON><!--
  <PERSON PERSONID="E03">
    <className>oracle.jdbc.driver.OracleDriver</className>
    <url>jdbc:oracle:thin:@localhost:1521:binbo</url>
    <user>scott</user>
    <password>tiger</password>
  </PERSON>
  --></PEOPLE>
```

-----
content.java

```
package org.binbo.dom;
import java.sql.Connection;
import java.sql.DriverManager;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
public class Content {
    public static Connection getConnection(){
        Connection conn=null;
        try {
            DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
            DocumentBuilder builder=factory.newDocumentBuilder();
            Document doc=builder.parse("da.xml");
            NodeList nl=doc.getElementsByTagName("PERSON");
            Element node=(Element) nl.item(0);
```

```

        String                                     className
=node.getElementsByTagName("className").item(0).getFirstChild().getNodeValue();

        String url =node.getElementsByTagName("url").item(0).getFirstChild().getNodeValue();

        String user=node.getElementsByTagName("user").item(0).getFirstChild().getNodeValue();

        String                                     pwd
=node.getElementsByTagName("password").item(0).getFirstChild().getNodeValue();

        Class.forName(className);
        conn =DriverManager.getConnection(url,user,pwd);

    } catch (Exception e) {
        e.printStackTrace();
    }
    return conn;
}
}

```

-----  
Domtest.java

```

package org.binbo.dom;
import java.io.FileOutputStream;
import java.sql.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class Domtest {
    public static void main(String[] args){
        try{
            Connection conn =Content.getConnection();
            PreparedStatement ps=conn.prepareStatement("select * from testxml");
            ResultSet rs =ps.executeQuery();
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.newDocument();
            Element Stu = doc.createElement("binboxml");

```



```

Element uname = doc.createElement("username");
Element nameid = doc.createElement("userid");
Element pass = doc.createElement("password");
while(rs.next()){
    String id=rs.getString(1);
    String name=rs.getString(2);
    String pwd=rs.getString(3);
    System.out.print(rs.getString(1));
    System.out.print(rs.getString(2));
    System.out.print(rs.getString(3));
    System.out.print("写入成功! ");
    nameid.appendChild(doc.createTextNode(id));
    uname.appendChild(doc.createTextNode(name));
    pass.appendChild(doc.createTextNode(pwd));
}

Stu.appendChild(nameid);
Stu.appendChild(uname);
Stu.appendChild(pass);
doc.appendChild(Stu);
TransformerFactory tf = TransformerFactory.newInstance();
Transformer tr = tf.newTransformer();
tr.transform(new DOMSource(doc), new StreamResult(
    new FileOutputStream("Binbo.xml")));
rs.close();
} catch(Exception e){
    e.printStackTrace();
}

}

}

=====数据库连接池=====

```

加到 tomcat 中的 server.xml

```

<Logger className="org.apache.catalina.logger.FileLogger"
        directory="logs" prefix="localhost_log." suffix=".txt"
        timestamp="true"/>

```

这个后面

```

<Context path="/myjsp" docBase="myjsp"
        debug="5" reloadable="true" crossContext="true">

```

```

        <Resource name="jdbc/myjsp"
            auth="Container"
            type="javax.sql.DataSource"/>
    <ResourceParams name="jdbc/myjsp">
        <parameter>
            <name>factory</name>
            <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>100</value>
        </parameter>
        <parameter>
            <name>maxIdle</name>
            <value>30</value>
        </parameter>
        <parameter>
            <name>maxWait</name>
            <value>10000</value>
        </parameter>

        <parameter>
            <name>username</name>
            <value>sa</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>sa</value>
        </parameter>
        <parameter>
            <name>driverClassName</name>
            <value>com.microsoft.jdbc.sqlserver.SQLServerDriver</value>
        </parameter>
        <parameter>
            <name>url</name>
            <value>jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=jsp</value>
        </parameter>
    </ResourceParams>
</Context>

```

---

用 DBPool.java 获得数据库连接池

```

package com.binbo.dbo;
import java.sql.Connection;
import java.sql.DriverManager;

```

```

import javax.naming.Context;
import javax.naming.NamingException;
import javax.sql.DataSource;
public class DBPool {
    public static synchronized Connection getConnection()throws Exception{
        DataSource ds = null;
        try{
            Context ininCtx  = new javax.naming.InitialContext();

            Context envCtx = (Context)ininCtx.lookup("java:comp/env");
            ds = (DataSource)envCtx.lookup("jdbc/myjsp");
        } catch(NamingException e){
            e.printStackTrace();
        }
        Connection conn = ds.getConnection();
        return conn;
    }
}

```

-----

操作数据库

```

package com.binbo.dbo;
import java.sql.*;
import com.binbo.javabean.BreakBean;
import com.binbo.javabean.OpenBean;
public class DataBaseClass {
    private Statement sta = null;
    private ResultSet rs = null;
    Connection conn = null;
    private int count;
    public DataBaseClass() throws Exception {
        // 取得数据库的连接
        conn = DBPool.getConnection();
        sta = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
    }
    public void getExecute(String sql) {
        try {
            System.out.println(sql);
            sta.executeUpdate(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

```

/*
 * sql 语句集的查询
 */
public ResultSet getQuery(String sql) {
    try {
        System.out.println(sql);
        rs = sta.executeQuery(sql);
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return rs;
}

```

\* 取得结果集的行数

```

public int getCount(ResultSet rs2) {
    try {
        rs2.next();
        count = rs2.getRow();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return count;
}

```

//添加!

```

public void getRegister(OpenBean open) {
    String sql = "insert into f_info(nam,email,titl,content,tem,mid)values(?,?,?,?,?,?)";
    try {
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, open.getNam());
        ps.setString(2, open.getEmail());
        ps.setString(3, open.getTitl());
        ps.setString(4, open.getContent());
        ps.setString(5, open.getTem());
        ps.setString(6, open.getDepa());
        ps.execute();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

// 删除

```

public void getRealys(OpenBean real) {
    try {

```

```

        String sql = "delete from departments where id=?";
        PreparedStatement ppt = conn.prepareStatement(sql);
        ppt.setString(1, real.getUnam());
        ppt.execute();

    } catch (SQLException e) {
        e.printStackTrace();
    }

}

// 查询
public void getRealy(OpenBean hg) {

    try {
        String sql = "select * from departments where id=?";
        PreparedStatement ppg = conn.prepareStatement(sql);
        ppg.setString(1, hg.getUnam());
        ppg.execute();
    } catch (SQLException e) {
        e.printStackTrace();
    }

}

}

=====hibernate 操作数据库=====
package com.binbo.hibernate.xml;
import java.util.Iterator;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
public class PersonOperate {
    private Session session = null ;
    public PersonOperate()
    {
        Configuration config = new Configuration().configure();
        SessionFactory factory = config.buildSessionFactory() ;
        this.session = factory.openSession();
    }
    // 增加数据
    public void insert(Person p)
    {

```

```

        Transaction tran = this.session.beginTransaction();
        this.session.save(p);
        tran.commit();
        this.session.close();
    }

```

-----

增加数据调用

```

        LinOperate po = new LinOperate();
        Lin p = new Lin();
        p.setLname(lname);
        p.setLmonery(lmonery);
        p.setLmain(lmain);
        p.setLtime(ltime);
        LinOperate po = new LinOperate();
        po.insert(p);
        errors.add("success", new ActionMessage("xiangxi"));
        request.setAttribute("org.apache.struts.action.ERROR", errors);
        return mapping.findForward("jinru");

```

-----

// 修改

```

public void update(Person p)
{
    Transaction tran = this.session.beginTransaction();
    this.session.update(p);
    tran.commit();
    this.session.close();
}

```

// 用户登录

```

public boolean queryById(Person person)
{
    boolean flag = false;
    String hql = "from Person as p where p.id=? and p.password=?";
    Query q = this.session.createQuery(hql);
    q.setString(0, person.getId());
    q.setString(1, person.getPassword());
    Iterator iter = q.list().iterator();
    if (iter.hasNext()) {
        flag = true;
        person.setName(((Person) iter.next()).getName());
    }
    this.session.close();
    return flag;
}

```

```

}
//验证用户存不存在
public boolean queryC(Person person)
{
    boolean flag = false;
    String hql = "from Person as p where p.name=?" ;
    Query q = this.session.createQuery(hql) ;
    q.setString(0,person.getName()) ;
    Iterator iter = q.list().iterator();
    if (iter.hasNext()) {
        flag = true;
    }
    this.session.close() ;
    return flag;
}
//查看个人资料
public Person queryZliao(String id)
{
    Person p = null ;
    String hql = "from Person as p where p.id=?" ;
    Query q = this.session.createQuery(hql) ;
    q.setString(0,id) ;
    List l = q.list() ;
    Iterator iter = l.iterator() ;
    if(iter.hasNext())
    {
        p = (Person)iter.next();
    }
    return p ;
}
// 删除数据
public void delete(Person p)
{
    Transaction tran = this.session.beginTransaction() ;
    this.session.delete(p) ;
    tran.commit() ;
}
// 修改
public void delete(String name)
{
    String hql = "delete Person where name=?" ;
    Query q = this.session.createQuery(hql) ;
    q.setString(0,name) ;
    q.executeUpdate() ;
}

```

```

        this.session.beginTransaction().commit() ;
    }

    // 查询全部数据
    public List queryAll()
    {
        List l = null ;
        String hql = "from Person as p" ;
        Query q = this.session.createQuery(hql) ;
        l = q.list() ;
        return l ;
    }
}

-----

(
    查询全部数据调用
        LinOperate po = new LinOperate();
        List l = po.queryAll();
        Iterator iter = l.iterator();
        ArrayList lus = new ArrayList();
        while (iter.hasNext()) {
            Lin p = (Lin) iter.next();
            Luser lu = new Luser();
            lu.setName(p.getLname());
            lu.setChange(p.getLmonery());
            lu.setMainn(p.getLmain());
            lu.setLtime(p.getLtime());
            lus.add(lu);
            System.out.print(p.getLname() + " ");
            request.setAttribute("lus", lus);
        }
)

-----

// 模糊查询
public List queryByLike(String cond)
{
    List l = null ;
    String hql = "from Person as p where p.name like ?" ;
    Query q = this.session.createQuery(hql) ;
    q.setString(0, "%" + cond + "%") ;
    l = q.list() ;
    return l ;
}
}

```



