

Oracle 第二天

一、子查询

● 什么是子查询？

使用子查询解决问题：谁的工资比 SCOTT 高？



● 子查询的语法

```
SELECT      select_list
FROM        table
WHERE       expr operator
            (SELECT      select_list
             FROM       table);
```

- 子查询（内查询）在主查询之前一次执行完成。
- 子查询的结果被主查询使用（外查询）。

- 1、可以在主查询的where select having from 后面使用子查询；
- 2、不可以在group by 使用子查询；
- 3、一般不在子查询中排序，但在top-n分析问题中 必须对子查询排序；
- 4、一般先执行子查询，再执行主查询；但相关子查询例外；

● 子查询的类型



● 单行子查询

- 只返回一条记录
- 单行操作符

操作符	含义
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

- 单行子查询示例 1

```

SELECT ename, job, sal
FROM emp
WHERE job = MANAGER
      (SELECT job
       FROM emp
       WHERE empno = 7566)
AND sal > 2450
      (SELECT sal
       FROM emp
       WHERE empno = 7782);
    
```

- 单行子查询示例 2

```

SELECT ename, job, sal
FROM emp
WHERE sal = 800
      (SELECT MIN(sal)
       FROM emp);
    
```

- 单行子查询示例 3

```

SELECT deptno, MIN(sal)
FROM emp
GROUP BY deptno
HAVING MIN(sal) >
          (SELECT MIN(sal)
           FROM emp
           WHERE deptno = 10);

```

800

● 单行子查询示例 4

范例：查询出比雇员 7654 的工资高，同时从事和 7788 的工作一样的员工

```

select *
  from emp t1
 where t1.sal > (select t1.sal from emp t where t.empno = 7654)
   and t1.job = (select t2.job from emp t2 where t2.empno = 7788)

```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	SCOTT	ANALYST	7566	1987/4/19	3000.00		20
2	FORD	ANALYST	7566	1981/12/3	3000.00		20

范例：要求查询每个部门的最低工资和最低工资的雇员和部门名称

```

select d.dname, a.minsal, e.ename
  from dept d,
       (select deptno, min(sal) minsal from emp group by deptno) a,
       emp e
 where d.deptno = a.deptno
   and e.sal = a.minsal

```

DNAME	MINSAL	ENAME
RESEARCH	800	SMITH
SALES	950	JAMES
ACCOUNTING	1300	MILLER

● 非法使用单行子查询示例

```

SELECT empno, ename
  FROM emp
 WHERE sal =
          (SELECT MIN(sal)
           FROM emp
           GROUP BY deptno);

```

```

ERROR at line 4:
ORA-01427: single-row subquery returns more than
one row

```

● 多行子查询

- 返回了多条记录
- 多行操作符

操作符	含义
IN	等于列表中的任何一个
ANY	和子查询返回的任意一个值比较
ALL	和子查询返回的所有值比较

● 子查询中的 null 值问题

■ 单行子查询中的 null 值问题

```
SELECT ename, job
FROM emp
WHERE job =
    (SELECT job
     FROM emp
     WHERE ename = 'Mike');
```

■ 多行子查询中的 null 值问题

示例：查询不是老板的员工

xxx not in (10,20 ,null) 等价于
xxx != 10 and xxx!=20 and xxx!=null
xxx!=null永远为false

```
SELECT *
FROM emp
WHERE empno not in( SELECT mgr
                      FROM emp);
```

多行子查询中 null 值需要注意的问题：

- ❖ **Returning Nulls in the Resulting Set of a Subquery**
- ❖ The SQL statement on the slide attempts to display all the employees who do not have any subordinates. Logically, this SQL statement should have returned 12 rows. However, the SQL statement does not return any rows. One of the values returned by the inner query is a null value, and hence the entire query returns no rows. The reason is that all conditions that compare a null value result in a null. So whenever null values are likely to be part of the results set of a subquery, do not use the NOT IN operator. The NOT IN operator is equivalent to <> ALL.
- ❖ Notice that the null value as part of the results set of a subquery is not a problem if you use the IN operator. The IN operator is equivalent to =ANY. For example, to display the employees who have subordinates, use the following SQL statement:
 - ❖ SELECT emp.last_name
 - ❖ FROM employees emp
 - ❖ WHERE emp.employee_id IN
 - ❖ (SELECT mgr.manager_id
 - ❖ FROM employees mgr);
 - ❖ Alternatively, a WHERE clause can be included in the subquery to display all employees who do not have any subordinates:
 - ❖ SELECT last_name FROM employees
 - ❖ WHERE employee_id NOT IN
 - ❖ (SELECT manager_id
 - ❖ FROM employees
 - ❖ WHERE manager_id IS NOT NULL);

二、课堂练习

- 找到员工表中工资最高的前三名，如下格式：

ROWNUM	EMPNO	ENAME	SAL
1	7839	KING	5000
2	7788	SCOTT	--工资前三名
3	7902	FORD	SELECT ROWNUM , e.* FROM (SELECT empno, ename, sal FROM emp ORDER BY sal DESC) e WHERE ROWNUM <=3

- 找到员工表中薪水大于本部门平均薪水的员工。

--薪水大于本部门平均薪水的员工

```
SELECT
    e.*,
    d.avgsal
FROM
    emp e,
    ( SELECT deptno, round( avg( sal ), 4 ) avgSal FROM EMP GROUP BY deptno ) d
WHERE
    e.deptno = d.deptno
    AND e.sal > d.avgsal
```

E	SAL	AUGSAL
N	1600	1566.66667
S	2975	2175
E	2850	1566.66667
7788 SCOTT	3000	2175
7839 KING	5000	2916.66667
7902 FORD	3000	2175

- 统计每年入职的员工个数

Total	1980	1981	1982
14	1	10	

--显示不同年份入职的员工数

```
SELECT
    count( 1 ) total,
    SUM(
        DECODE( to_char( hiredate, 'yyyy' ), 1981, 1, 0 ) ) "1981",
    SUM(
        DECODE( TO_CHAR( hiredate, 'YYYY' ), 1980, 1, 0 ) ) "1980",
    SUM(
        DECODE( TO_CHAR( hiredate, 'yyyy' ), 1982, 1, 0 ) ) "1982",
    SUM(
        DECODE( TO_CHAR( hiredate, 'yyyy' ), 1987, 1, 0 ) ) "1987"
FROM
    emp;
```

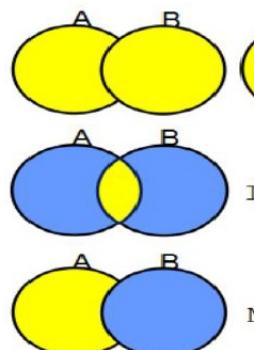
- 补充知识点：Oracle 中的分页查询

ROWNUM:表示行号，实际上此是一个列,但是这个列是一个伪列,此列可以在每张表中出现。

- 1、rownum/level永远按照默认的顺序生成；
 - 2、rownum只能使用<=

三、集合运算

● 什么是集合运算？



集合运算注意事项：

- 1、各个集合的列数必须相同
- 2、采用第一个集合作为最后的表头
- 3、order by永远在最后

UNION/UNION ALL 并集

INTERSECT 交集

MINUS 差集

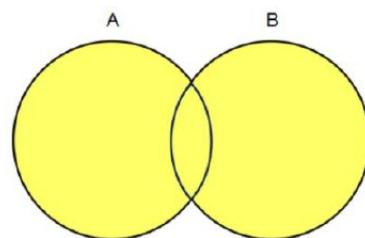
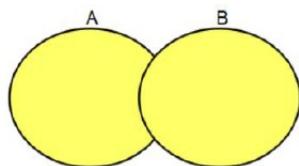
要点：
1.列名不足用to_char/number
(null)补充；
2.rollup关键字

```

1 SELECT deptno,job,sum(sal) from EMP GROUP BY deptno,job
2 UNION
3 SELECT deptno,TO_CHAR(null),SUM(sal) from EMP GROUP BY deptno
4 UNION
5 SELECT to_number(null),to_char(null),sum(sal) from EMP
6
7 ==等价于==
8
9 select deptno,job,sum(sal) from emp group by rollup(deptno,job);

```

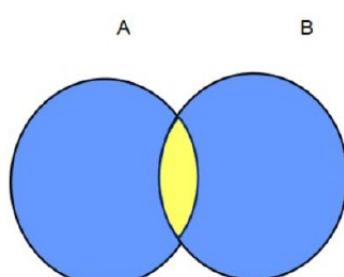
● 并集



UNION运算符返回两个集合去掉重复元素后的所有记录。

UNION ALL 返回两个集合的所有记录，包括重复的

● 交集



INTERSECT 运算符返回同时属于两个集合的记录

使用INTERSECT运算符

❖ 显示薪水同时位于级别**1 (700~1300)** 和级别**2 (1201~1400)** 的员工信息。

```

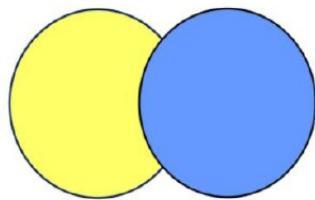
select ename,sal from emp
where sal between 700 and 1300
INTERSECT
select ename,sal from emp
where sal between 1201 and 1400;

```

ENAME	SAL
MARTIN	1250
MILLER	1300
WARD	1250

● 差集

A B



MINUS返回属于第一个集合，但不属于第二个集合的记录。

使用 **Minus** 运算符

- 显示薪水同时位于级别**1 (700~1300)**，但不
属于级别**2 (1201~1400)** 的员工信息。

```
select ename,sal from emp
where sal between 700 and 1300
minus
select ename,sal from emp
where sal between 1201 and 1400;
```

ENAME	SAL
ADAMS	1100
JAMES	950
SMITH	800



四、使用 DDL 语句管理表

● 创建表空间

表空间？ ORACLE 数据库的逻辑单元。 数据库---表空间 一个表空间可以与多个数据文件(物理结构)关联一个数据库下可以建立多个表空间,一个表空间可以建立多个用户、一个用户下可以建立多个表。

```
create tablespace itcast001
datafile 'c:\itcast001.dbf'
size 100m
autoextend on
next 10m
```

itcast 为表空间名称

datafile 指定表空间对应的数据文件

size 后定义的是表空间的初始大小

autoextend on 自动增长 , 当表空间存储都占满时, 自动增长

next 后指定的是一次自动增长的大小。

● 用户

1、创建用户

```
create user itcastuser
identified by itcast
default tablespace itcast001
```

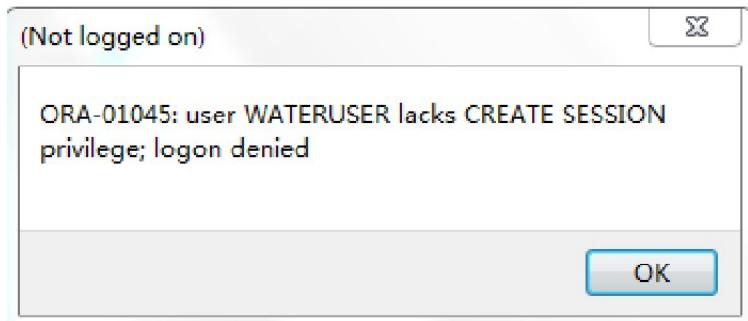
identified by 后边是用户的密码

default tablespace 后边是表空间名称

oracle 数据库与其它数据库产品的区别在于, 表和其它的数据库对象都是存储在用户下的。

2、用户赋权限

新创建的用户没有任何权限, 登陆后会提示



Oracle 中已存在三个重要的角色：connect 角色，resource 角色，dba 角色。

CONNECT 角色：--是授予最终用户的典型权利，最基本的

ALTER SESSION --修改会话
CREATE CLUSTER --建立聚簇
CREATE DATABASE LINK --建立数据库链接
CREATE SEQUENCE --建立序列
CREATE SESSION --建立会话
CREATE SYNONYM --建立同义词
CREATE VIEW --建立视图

RESOURCE 角色：--是授予开发人员的

CREATE CLUSTER --建立聚簇
CREATE PROCEDURE --建立过程
CREATE SEQUENCE --建立序列
CREATE TABLE --建表
CREATE TRIGGER --建立触发器
CREATE TYPE --建立类型

DBA 角色：拥有全部特权，是系统最高权限，只有 DBA 才可以创建数据库结构，并且系统权限也需要 DBA 授出，且 DBA 用户可以操作全体用户的任意基表，包括删除

grant dba to itcastuser

进入 system 用户下给用户赋予 dba 权限，否则无法正常登陆

● 创建表

语法：

```
CREATE TABLE [schema.]table
  (column datatype [DEFAULT expr][, ...]);
```

数据的类型：

数据类型	描述
VARCHAR2 (size)	可变长字符数据
CHAR (size)	定长字符数据
NUMBER (p, s)	可变长数值数据
DATE	日期型数据
LONG	可变长字符数据, 最大可达到2G
CLOB	字符数据, 最大可达到4G
RAW and LONG RAW	原始的二进制数据
BLOB	二进制数据, 最大可达到4G
BFILE	存储外部文件的二进制数据, 最大可达到4G
ROWID	行地址

使用子查询创建表的语法:

```
CREATE TABLE table
[(column, column...)]
AS subquery;
```

创建表范例: 创建 person 表

```
create table person(
    pid      number(10),
    name     varchar2(10),
    gender   number(1) default 1,
    birthday date
);
insert into person(pid, name, gender, birthday)
values(1, '张三', 1, to_date('1999-12-22', 'yyyy-MM-dd'));
```

● 修改表

在 sql 中使用 alter 可以修改表

- 添加语法: ALTER TABLE 表名称 ADD(列名 1 类型 [DEFAULT 默认值], 列名 1 类型 [DEFAULT 默认值]...)
- 修改语法: ALTER TABLE 表名称 MODIFY(列名 1 类型 [DEFAULT 默认值], 列名 1 类型 [DEFAULT 默认值]...)
- 修改列名: ALTER TABLE 表名称 RENAME COLUMN 列名 1 TO 列名 2

范例: 在 person 表中增加列 address

```
alter table person add(address varchar2(10));
```

范例: 把 person 表的 address 列的长度修改成 20 长度

```
alter table person modify(address varchar2(20));
```



● 删除表

语法：DROP TABLE 表名

drop并非真正完全删除，而是放入回收站。

--查看回收站
show recyclebin;

--清空回收站
purge recyclebin;

--闪回删除 ---> 回收站
flashback table TESTSAVEPOINT to before drop;

--注意：管理员没有回收站

● 约束

在数据库开发中，约束是必不可少，使用约束可以更好的保证数据的完整性。在 Oracle 数据库中，约束的类型包括：

- ✓ 主键约束 (Primary Key)
- ✓ 非空约束 (Not Null)
- ✓ 唯一约束 (Unique)
- ✓ 外键约束 (Foreign Key)
- ✓ 检查性约束 (Check)

1. 主键约束

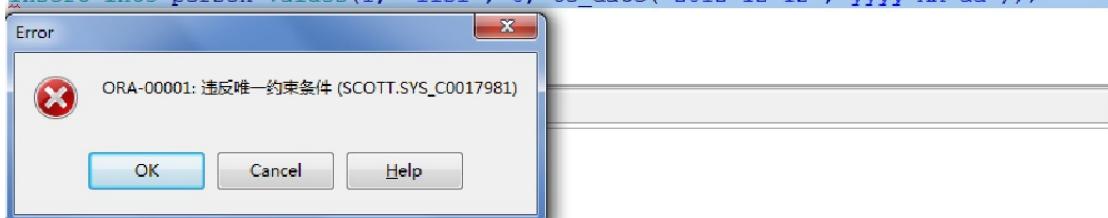
主键约束都是在 id 上使用，而且本身已经默认了内容不能为空，可以在建表的时候指定。

创建一张表，把 pid 作为主键

```
create table person(
    pid      number(10) primary key,
    name     varchar2(10),
    gender   number(1) default 1,
    birthday date
);
```

主键不可重复， SCOTT.SYS_C0017981 是系统自动分配的约束的名字

```
insert into person values(1, 'zhangsan', 1, to_date('2012-12-12','yyyy-MM-dd'));
insert into person values(1, 'lisi', 0, to_date('2012-12-12','yyyy-MM-dd'));
```



主键不可为空

```
insert into person values(null, 'zhangsan', 1, to_date('2012-12-12','yyyy-MM-dd'));

```

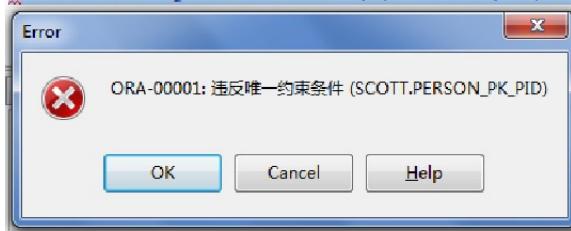
The screenshot shows another 'Error' dialog box from Oracle. It contains the error message 'ORA-01400:无法将NULL插入 (SCOTT.PERSON.PID)' (ORA-01400: Cannot insert NULL into (SCOTT.PERSON.PID)). There are three buttons at the bottom: 'OK', 'Cancel', and 'Help'.

我们可以自己来指定主键约束的名字

```
create table person(
    pid      number(10),
```



```
name      varchar2(10),
gender    number(1) default 1,
birthday  date,
constraint person_pk_pid primary key(pid)
);
insert into person values(1, 'zhangsan', 1, to_date('2012-12-12','yyyy-MM-dd'));
insert into person values(1, 'lisi', 0, to_date('2012-12-12','yyyy-MM-dd'));
```

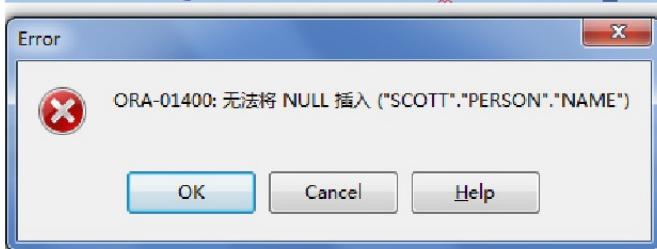


2. 非空约束

使用非空约束，可以使指定的字段不可以为空。

范例：建立一张 pid 和 name 不可以为空的表

```
create table person(
    pid      number(10) not null,
    name     varchar2(10) not null,
    gender   number(1) ,
    birthday date,
);
insert into person values(1, null, 1, to_date('2012-12-12','yyyy-MM-dd'));
```



3. 唯一约束 (unique)

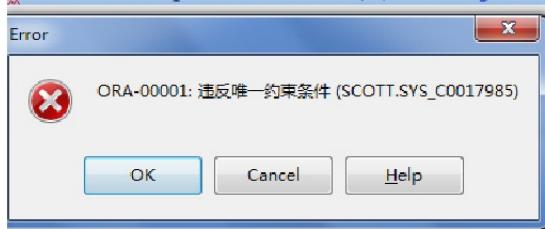
表中的一个字段的内容是唯一的

范例：建表一个 name 是唯一的表

```
create table person(
    pid      number(10) ,
    name     varchar2(10) unique,
    gender   number(1) ,
    birthday date
);
```



```
insert into person values(1, 'zhangsan', 1, to_date('2012-12-12','yyyy-MM-dd'));
insert into person values(2, 'zhangsan', 0, to_date('2012-12-12','yyyy-MM-dd'));
```



唯一约束的名字也可以自定义

```
create table person(
    pid      number(10) ,
    name     varchar2(10),
    gender   number(1) ,
    birthday date,
    constraint person_name_uk unique(name)
);
insert into person values(1, 'zhangsan', 1, to_date('2012-12-12','yyyy-MM-dd'));
insert into person values(2, 'zhangsan', 0, to_date('2012-12-12','yyyy-MM-dd'))
```



4. 检查约束

使用检查约束可以来约束字段值的合法范围。

范例：创建一张表性别只能是 1 或 2

```
create table person(
    pid      number(10) ,
    name     varchar2(10),
    gender   number(1) check(gender in (1, 2)),
    birthday date
);

insert into person values(1, 'zhangsan', 3, to_date('2012-12-12','yyyy-MM-dd'));
```



检查约束也可以自定义

```
create table person(
    pid      number(10) ,
    name     varchar2(10),
```



```
gender number(1),  
birthday date,  
constraint person_gender_ck check(gender in (1,2))  
);  
  
insert into person values(1, 'zhangsan', 3, to_date('2012-12-12','yyyy-MM-dd'));
```

5.外键约束

之前所讲的都是单表的约束，外键是两张表的约束，可以保证关联数据的完整性。

范例：创建两张表，一张订单表，一张是订单明细表，订单和明细是一对多的关系

```
create table orders(  
    order_id      number(10) ,  
    total_price   number(10,2) ,  
    order_time    date,  
    constraint orders_order_id_pk primary key(order_id)  
);  
  
create table order_detail(  
    detail_id     number(10) ,  
    order_id      number(10) ,  
    item_name     varchar2(10) ,  
    quantity      number(10) ,  
    constraint     order_detail_detail_id_pk      primary  
key(detail_id)  
);  
  
insert      into      orders      values(1,           200,  
to_date('2015-12-12','yyyy-MM-dd'));  
insert into order_detail values(1, 2, 'java',1);
```

我们在两张表中插入如上两条数据，我们发现在 `order_detail` 表中插入的 `order_id` 在 `order` 表中并不存在，这样在数据库中就产生了脏数据。此时需要外键来约束它。

我们再次建表

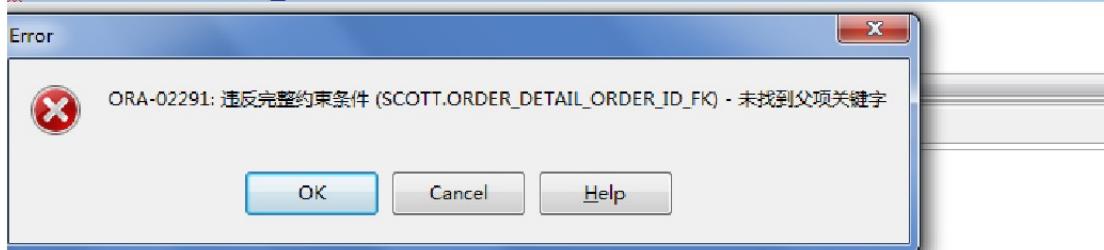
```
create table orders(  
    order_id      number(10) ,  
    total_price   number(10,2) ,  
    order_time    date,  
    constraint orders_order_id_pk primary key(order_id)
```



```
);

create table order_detail(
    detail_id      number(10) ,
    order_id       number(10),
    item_name     varchar2(10),
    quantity       number(10),
    constraint      order_detail_detail_id_pk      primary
key(detail_id),
    constraint      order_detail_order_id_fk      foreign
key(order_id) references orders(order_id)
);
```

```
insert into orders values(1, 200, to_date('2015-12-12','yyyy-MM-dd'));
insert into order_detail values(1, 2, 'java',1);
```



外键关联一定注意：

外键一定是主表的主键

删表时一定先删子表再删主表，如果直接删主表会出现由于约束存在无法删除的问题

```
SQL> drop table orders;
drop table orders
ORA-02449: 表中的唯一/主键被外键引用
```

```
--on delete set null
--on delete cascade
```

但是可以强制删除 drop table orders cascade constraint; (不建议)

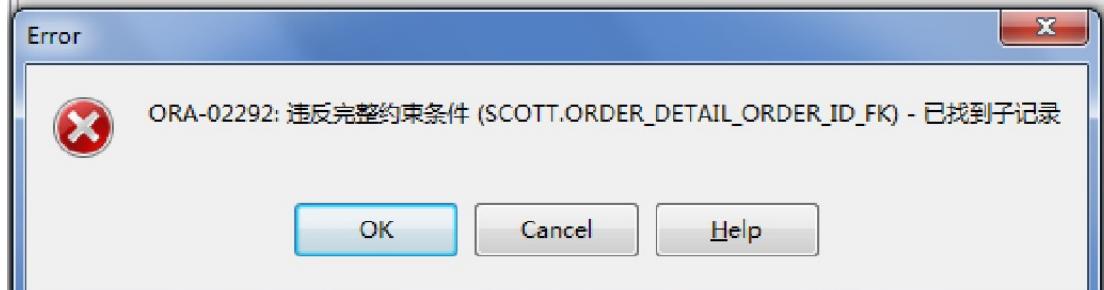
删除主表的数据可以先删除子表的关联数据，再删主表，也可以使用级联删除。

级联删除在外键约束上要加上 on delete cascade 如

```
constraint order_detail_order_id_fk foreign key(order_id)
    references orders(order_id) on delete cascade
```

这样删除主表数据的时候会把子表的关联数据一同删除

```
delete from orders
```





黑马程序员
www.itheima.com

| 传智播客旗下
高端IT教育品牌

改变中国IT教育，我们正在行动



五、使用 DML 语句处理数据

● 插入数据

语法：INSERT INTO 表名[(列名 1, 列名 2, ...)]VALUES(值 1, 值 2, ...)

标准写法

```
insert into person(pid, name, gender, birthday, address)
values(1, '张三', 1, '9-5月-1981', '北京北七家');
```

简单写法（不建议）

```
INSERT INTO 表名 VALUES(值 1, 值 2, ...)
```

```
insert into person
```

```
values(1, '张三', 1, '9-5月-1981', '北京北七家');
```

注意：使用简单的写法必须按照表中的字段的顺序来插入值，而且如果有为空的字段使用 null

```
insert into person values(2, '李四', 1, null, '北京育新');
```

海量插入数据可以使用以下方式：
 1、数据泵dbms_datapump (PLSQL程序包)
 2、SQL*Loader
 3、外部表

● 更新数据

全部修改：UPDATE 表名 SET 列名 1=值 1, 列名 2=值 2, ...

局部修改：UPDATE 表名 SET 列名 1=值 1, 列名 2=值 2, ... WHERE 修改条件；

在 update 中使用子查询：

例如：给 NEW YORK 地区的所有员工涨 100 员工资

```
update emp set sal=sal+100 where deptno
in (select deptno from dept where loc='NEW YORK')
```

● 删除数据

语法：DELETE FROM 表名 WHERE 删除条件；

在删除语句中如果不指定删除条件的话就会删除所有的数据

Truncate table 实现数据删除

“逐D释碎闪快”
猪的事最爽快

delete和truncate的区别
 1、delete逐条删除；truncate先摧毁表 再重建
 2、(*) delete是DML truncate是DDL
 (可以回滚) (不可以回滚)
 3、delete不会释放空间 truncate会
 4、delete会产生碎片 truncate不会
 5、delete可以闪回 (flashback) truncate不可以
 6、delete的删除速度快 (mysql相反)

注意：插入、更新和删除会引起数据的变化。我们就必须考虑数据的完整性。

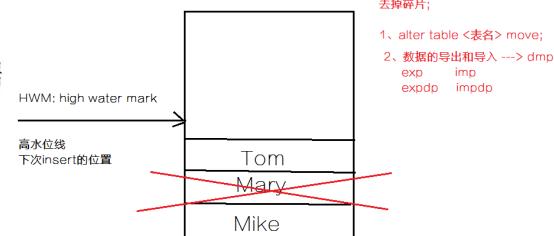
这里有个错误，应该是truncate删除
比较快。

Oracle 中的事务

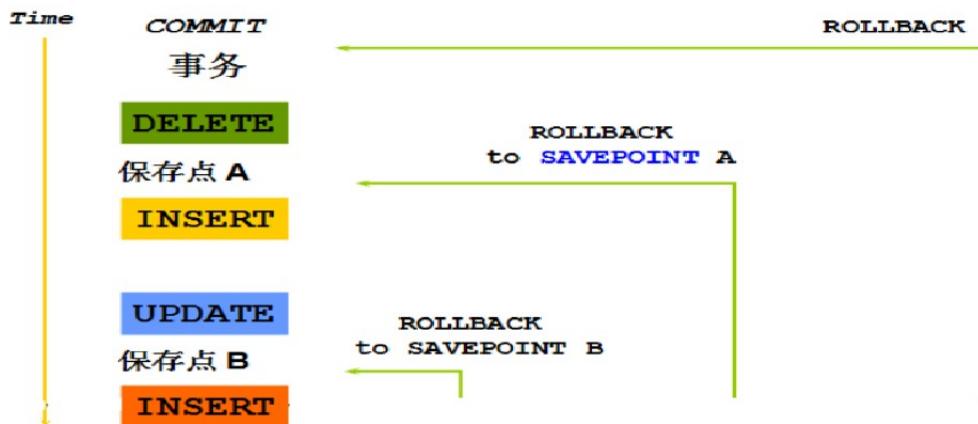
这是因为 oracle 的事务对数据库的变更的处理，我们必须做提交事务才能让数据真正的插入到数据库中，在同样在执行完数据库变更的操作后还可以把事务进行回滚，这样就不会插入到数据库。如果事务提交后则不可以再回滚。

Oracle中的事务

- 1、起始标志：事务中的第一条DML语句
- 2、结束标志：提交：显式 commit 隐式：正常退出 DDL DCL
回滚：显式 rollback 隐式：非正常退出



Oracle 中事务的保存点：



事务的隔离级别和隔离性：

隔离级别	描述
READ UNCOMMITTED (读未提交数据)	允许事务读取未被其他事务提交的变更。脏读、不可重复读和幻读的问题都会出现。
READ COMMITTED (读已提交数据)	只允许事务读取已经被其它事务提交的变更。可以避免脏读，但不可重复读和幻读问题仍然可能出现。
REPEATABLE READ (可重复读)	确保事务可以多次从一个字段中读取相同的值。在这个事务持续期间，禁止其他事务对这个字段进行更新。可以避免脏读和不可重复读，但幻读的问题仍然存在。
SERIALIZABLE(串行化)	确保事务可以从一个表中读取相同的行。在这个事务持续期间，禁止其他事务对该表执行插入、更新和删除操作。所有并发问题都可以避免，但性能十分低下。

Oracle 支持的 3 种事务隔离级别：READ COMMITTED, SERIALIZABLE, READ ONLY.

Oracle 默认的事务隔离级别为：READ COMMITTED

这四个都是SQL99中的标准，
Mysql四种都支持，默认：可重复读
Oracle只支持其中两种+自身定义的Read Only，默认：读已提交

<https://www.cnblogs.com/huanongying/p/7021555.html>

设置事务隔离级别：set transaction xxxx;

六、管理其他数据库对象

● 视图

➤ 什么是视图：

视图就是封装了一条复杂查询的语句。

视图是一个虚表。

最大的优点就是简化复杂的查询。

➤ 创建视图的语法

```
CREATE [OR REPLACE] [(force|not force)] VIEW view
[(alias[, alias]...)]
AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

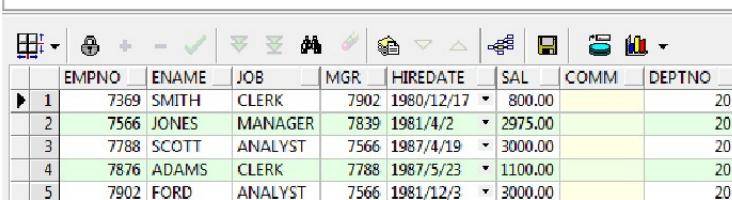
➤ 创建视图示例

范例：建立一个视图，此视图包括了 20 部门的全部员工信息

```
create view empvd20 as select * from emp t where t.deptno = 20
```

视图创建完毕就可以使用视图来查询，查询出来的都是 20 部门的员工

```
select * from EMPVD20 t
```



	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980/12/17	800.00		20
2	7566	JONES	MANAGER	7839	1981/4/2	2975.00		20
3	7788	SCOTT	ANALYST	7566	1987/4/19	3000.00		20
4	7876	ADAMS	CLERK	7788	1987/5/23	1100.00		20
5	7902	FORD	ANALYST	7566	1981/12/3	3000.00		20

语法 2：CREATE OR REPLACE VIEW 视图名称 AS 子查询

如果视图已经存在我们可以使用语法 2 来创建视图，这样已有的视图会被覆盖。

```
create or replace view empvd20 as select * from emp t where t.deptno = 20
```

➤ 不建议通过视图对表中的数据进行修改，因为会受到很多的限制。

● 序列

在很多数据库中都存在一个自动增长的列, 如果现在要想在 oracle 中完成自动增长的功能, 则只能依靠序列完成, 所有的自动增长操作, 需要用户手工完成处理。并且 Oracle 将序列值装入内存可以提高访问效率。

语法:

```
CREATE SEQUENCE sequence
  [INCREMENT BY n]
  [START WITH n]
  [{MAXVALUE n | NOMAXVALUE}]
  [{MINVALUE n | NOMINVALUE}]
  [{CYCLE | NOCYCLE}]
  [{CACHE n | NOCACHE}];
```

范例:

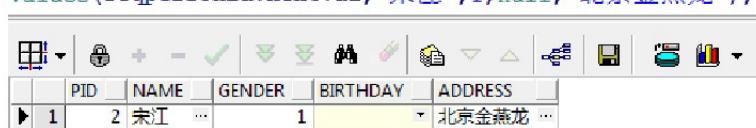
```
CREATE SEQUENCE dept_deptid_seq
  INCREMENT BY 10
  START WITH 120
  MAXVALUE 9999
  NOCACHE
  NOCYCLE;
Sequence created.
```

序列创建完成之后, 所有的自动增长应该由用户自己处理, 所以在序列中提供了以下的两种操作:

- nextval : 取得序列的下一个内容
- currval : 取得序列的当前内容

在插入数据时需要自增的主键中可以这样使用

```
insert into person
values (seqpersonid.nextval, '宋江', 1, null, '北京金燕龙');
```



	PID	NAME	GENDER	BIRTHDAY	ADDRESS
▶	1	宋江	...	1	北京金燕龙

序列可能产生裂缝的原因:

- 回滚
- 系统异常
- 多个表共用一个序列

● 索引

索引即目录

索引由oracle自身维护

索引是用于加速数据存取的数据对象。合理的使用索引可以大大降低 i/o 次数, 从而提高数据访问性能。

oracle的索引类型 :

- 1. B树索引(默认)
- 2. 位图索引

1. 单列索引

在一个列上建索引

单列索引是基于单个列所建立的索引, 比如:

```
CREATE index 索引名 on 表名(列名)
```

2. 复合索引

在多个列上建索引

复合索引是基于两个列或多个列的索引。在同一张表中, 要求列的组合必须不同, 比如:

```
Create index emp_idx1 on emp(ename, job);
```

```
Create index emp_idx1 on emp(job, ename);
```

范例: 给 person 表的 name 建立索引

```
create index pname_index on person(name);
```

范例: 给 person 表创建一个 name 和 gender 的索引

```
create index pname_gender_index on person(name, gender);
```

以下情况可以创建索引:

- 列中数据值分布范围很广
- 列经常在 WHERE 子句或连接条件中出现
- 表经常被访问而且数据量很大, 访问的数据大概占数据总量的2%到4%

下列情况不要创建索引:

- 表很小
- 列不经常作为连接条件或出现在WHERE子句中
- 查询的数据大于2%到4%
- 表经常更新

● 同义词

就是别名

```
CREATE [PUBLIC] SYNONYM synonym  
FOR object;
```

例如:

```
create public synonym emp for scott.emp;
```

```
select * from emp;
```

```
drop public synonym emp;
```

使用同义词的作用?

1. 可以很方便的访问其它用户的数据库对象
2. 缩短了对象名字的长度