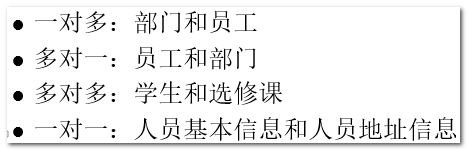
Oracle第二天

# 多表（关联）查询

多表查询也称之为关联查询、多表关联查询等，主要是指通过多个表的关联来获取数据的一种方式。

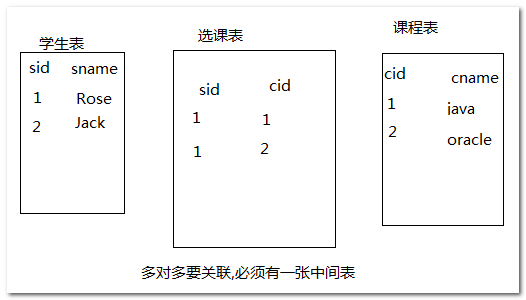
## 多表映射关系



一对多：A表的一行数据，对应B表中的多条。如:一个部门可以对应多个员工.

多对一：B表中的多条对应A表的一行数据.如:多个员工对应一个部门.

多对多：学生和选修课表----学生和课程对应表。



一对一：人员基本信息和人员信息扩展表。



## 笛卡尔集



笛卡尔集对于我们数据库的数据查询结果的影响：

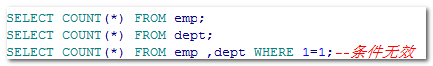
1. 数据冗余。---笛卡尔集并不是我们所需要的数据.
2. 效率问题：导致数量级的增长。100w \*100w，====》1w亿。如果你在查询大量数据的时候，不注意这个笛卡尔集的话，会导致你的查询结果时间非常非常非常长，还会导致数据库故障。

因此，在实际运行环境下，应避免使用全笛卡尔集。

笛卡尔集产生的条件：

* 省略连接条件
* 连接条件无效

如下示例：



如何避免笛卡尔集：

在 WHERE 加入有效的连接条件。

这时候就需要学习表关联的几种方式了。

## 多表连接的类型

根据连接方式的不同，Oracle的多表关联的类型分为：

内连接、外连接、自连接。

内连接分为：等值内连接、不等值内连接

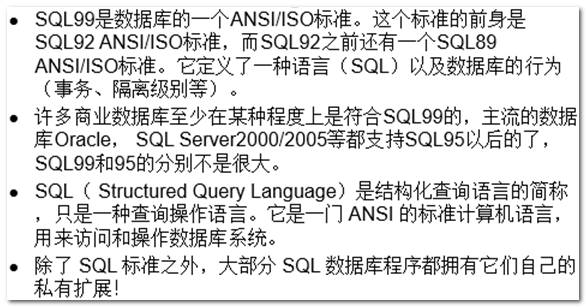
外连接分为：左外连接、右外连接、全外连接

自连接是一种特殊的关联，可以包含内连接和外连接的连接方式。

## 关于sql99-了解

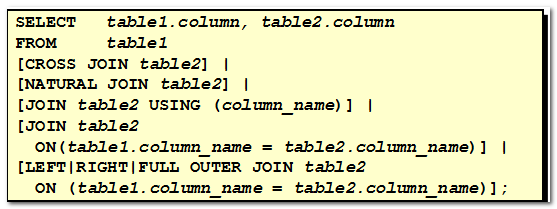
Oracle是关系型数据库，它遵循sql99的规范（sql规范）。

但是，mysql和Oracle有些地方不一样，原因：各个厂商的实现可能会有差别。

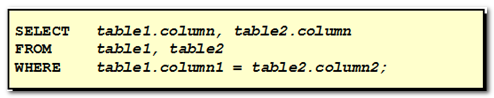


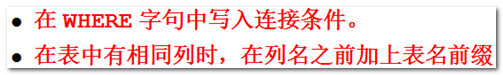
## 多表连接的基本语法

Sql99的语法：



Oracle的语法：





sql语句 优化：

加上前缀：效率高！

## 内连接

### 等值内连接

等值内连接也称之为等值连接。

【示例】

*-需求：查询一下员工信息，并且显示其部门名称*

|  |
| --- |
| *--需求：查询一下员工信息，并且显示其部门名称*  SELECT \* FROM emp t1,dept t2 WHERE t1.deptno=t2.deptno;*--等值内连接，数据库的私有扩展语法：隐式内连接（mysql，oracle都支持）*  SELECT \* FROM emp t1 INNER JOIN dept t2 ON t1.deptno=t2.deptno;*--sql99语法，显示内连接(所有符合sql99规范的都支持)* |
| *扩展：表设计的”范式“，尽量满足前三范式。* |

### 不等值内连接

不等值内连接也称之为不等值连接。

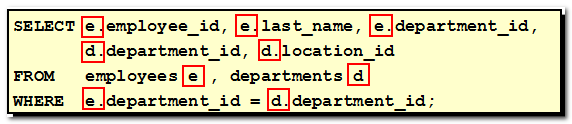
【示例】需求:查询员工信息,要求显示员工的编号、姓名、月薪、工资级别。

|  |
| --- |
| --分析：要完成这个需求，需要使用到下面两张表：  *--需求:查询员工信息,要求显示员工的编号、姓名、月薪、工资级别。*  SELECT \* FROM emp t1,salgrade t2 WHERE t1.sal >=t2.losal AND t1.sal<=t2.hisal;*--隐式语法*  SELECT \* FROM emp t1,salgrade t2 WHERE t1.sal BETWEEN t2.losal AND t2.hisal;*--隐式不等值连接* |

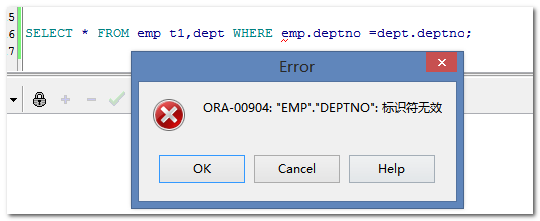
### 表的别名

为什么要使用表的别名？

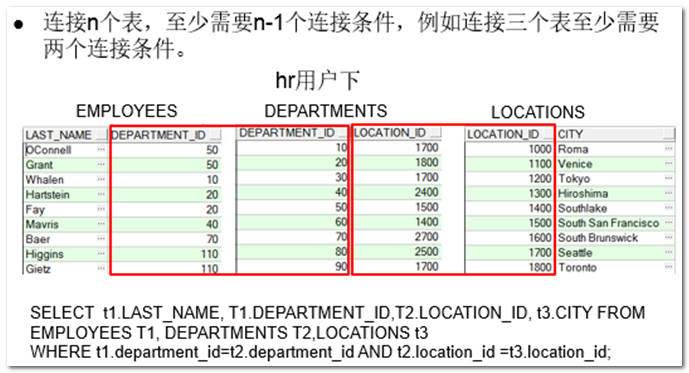
* 使用别名可以简化查询。
* 使用表名前缀可以提高执行效率。--SQL性能优化方案
* 在不同表中具有相同列名的列，可以用表的别名作为前缀来加以区分。



需要注意的是，如果一旦使用了表的别名，则不能在用真实的表名。



### 更多表的连接



注意：这个理论。

## 外连接

分为左外连接，右外连接，全外连接。

### 左外连接

*--查询"所有"员工信息，要求显示员工号，姓名 ,和部门名称--要求使用左外连接*

|  |
| --- |
| *--查询"所有"员工信息，要求显示员工号，姓名 ,和部门名称--要求使用左外连接*  SELECT \* FROM emp t1 LEFT OUTER JOIN dept t2 ON t1.deptno=t2.deptno;*--sql99标准语法*  SELECT \* FROM emp t1,dept t2 WHERE t1.deptno=t2.deptno(+);*--oracle私有语法（mysql不支持），+放到右边是左外，你可以认为(+)是附加补充的意思。*--要求查询所有的信息的表，我们可以称之为主表，而补充信息的表，称之为从表 |

### 右外连接

*----查询“所有”部门及其下属的员工的信息。--右外连接*

|  |
| --- |
| SELECT \* FROM emp t1 RIGHT OUTER JOIN dept t2 ON t1.deptno=t2.deptno;*--sql99--右外连接--右边表（dept）数据全部显示。*  SELECT \* FROM emp t1,dept t2 WHERE t1.deptno(+)=t2.deptno;*--oracle语法，右外连接* |

### 如何选择左外和右外

|  |
| --- |
| SELECT t1.\*,t2.\* FROM dept t1 ,emp t2 WHERE t1.deptno=t2.deptno(+);  *--1.到底是使用左外还是右外,主要是看两张表的在语句中的位置,*  *--两张表是有主从关系,一般把主表放在左边,----一般两张表的情况下,我们都使用左连接.*  *--2.+到底是放在条件哪边?左外连接的+放在右边,右外连接的+放在左边.----记忆的方法:(+)放在从表的一方,起到数据附加的作用.*    简单的说：左外连接就是左边的表的数据全部显示，右外就是右边的表的数据全部显示。 |

这种（+）的写法，只能用在Oracle。不能用于mysql！

一定要有主表和从表这个概念，分清那张是主表，哪张是从表。

把你想查询基础表当成左表。想把谁全部都查询出来就当成主表。

到底哪张是主表哪张是从表？最终还看你的需求。

一般我们把主表放在左边，使用左外连接。

一般情况下，我们就用左连接就行了。

### 全外连接

左表和右表的数据全部都显示，而且不是笛卡尔集。

相当于左外+右外的数据。

【示例】

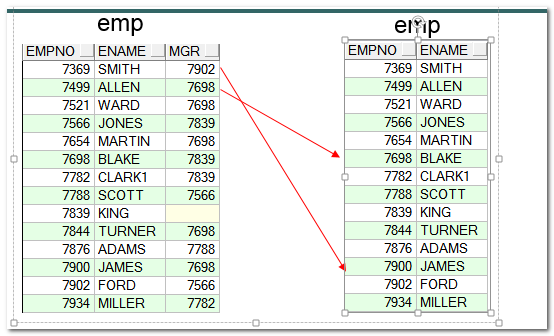
需求：要求将所有员工和所有部门都显示出来

|  |
| --- |
| *--全外连接*  SELECT \* FROM emp t1 LEFT OUTER JOIN dept t2 on t1.deptno=t2.deptno  UNION  SELECT \* FROM emp t1 RIGHT JOIN dept t2 ON t1.deptno=t2.deptno;  SELECT \* FROM emp t1 FULL OUTER JOIN dept t2 ON t1.deptno=t2.deptno;*--sql99语法，Oracle没有私有扩展的语法。而且，mysql没有全外* |

## 自连接

自连接，就是将一张表当成两张表来查询。

### 示例



自连接的查询的原理：就是将一张表当成两张表来使用.

【示例】

1.查询员工信息，要求同时显示员工和员工的领导的姓名

2.查询“所有”员工信息，要求同时显示员工和员工的领导的姓名

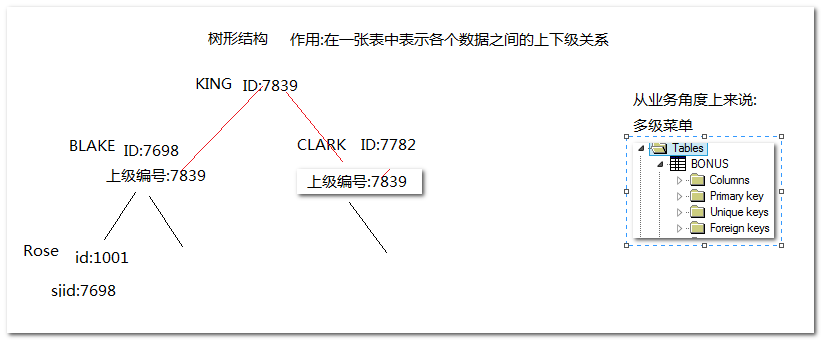
|  |
| --- |
| *--查询员工信息，要求同时显示员工和员工的领导的姓名*  SELECT \* FROM emp t1,emp t2 WHERE t1.mgr=t2.empno;  *--查询“所有”员工信息，要求同时显示员工和员工的领导的姓名*  SELECT \* FROM emp t1,emp t2 WHERE t1.mgr=t2.empno(+); |

自连接是一种特殊的多表连接方式,其实含有内连接和外连接的操作.

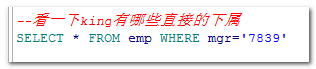
注意问题:你也要注意笛卡尔集的产生.



【扩展知识】:自连接的经典案例就是树形结构的设计。（表述父子关系-家族关系，上下级关系）



实际业务中获取数据的方式一般是采用树形节点的ajax懒加载，当展开节点的时候，查询节点下面的一级子元素：



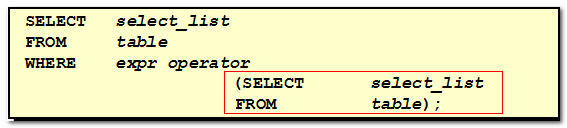
选作作业：

自己使用ajax方式，将人员表的树形结构打印到页面。

# 子查询

子查询也称之为嵌套子句查询。

## 语法



语法上的运行使用规则：

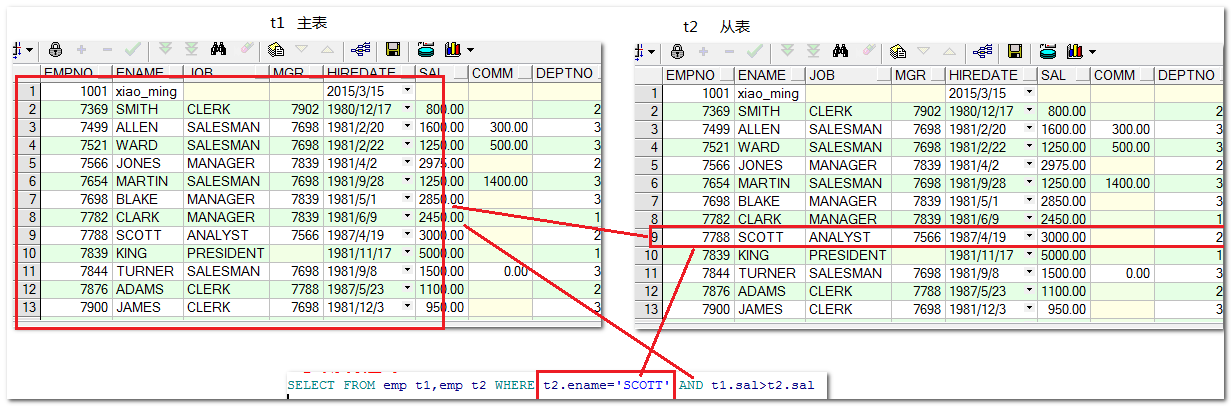
* 子查询 (内查询、嵌套子句) 在主查询之前一次执行完成。（子查询先执行）
* 子查询的结果被主查询使用 (外查询)。
* 子查询要包含在括号内。
* 将子查询放在比较条件的右侧。

## 为什么要使用子查询？

【需求】谁的工资比scott高？

采用连接的方式写（这里是自连接，见下图）：

|  |
| --- |
| --【需求】谁的工资比scott高？  --多表关联查询：*自连接的不等值连接*  SELECT \* FROM emp t1,emp t2 WHERE t2.ename='SCOTT' AND t1.sal>t2.sal  --不等值连接 |

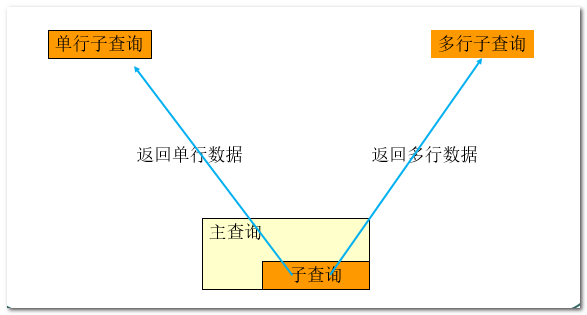


采用子查询的方式写：

|  |
| --- |
| *--子查询*  *--分析一下:谁的工资比scott高？--->1,scott工资是多少2，谁的工资比3000高*  SELECT sal FROM emp WHERE ename='SCOTT';  SELECT \* FROM emp WHERE sal >3000;  SELECT \* FROM emp WHERE sal >(SELECT sal FROM emp WHERE ename='SCOTT'); |

对比可以发现：在某些业务上，子查询比连接查询更容易理解。

## 子查询的分类

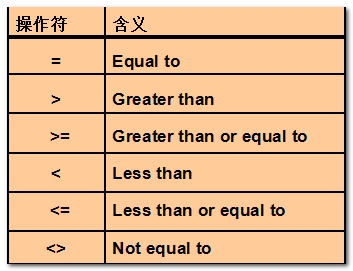


单行操作符（> = <）对应单行子查询，多行操作符(in,not in)对应多行子查询。

## 单行子查询

### 语法要求：

* 只返回一行。
* 使用单行比较操作符。



其中<>也可以可以用!=代替，意思一样。

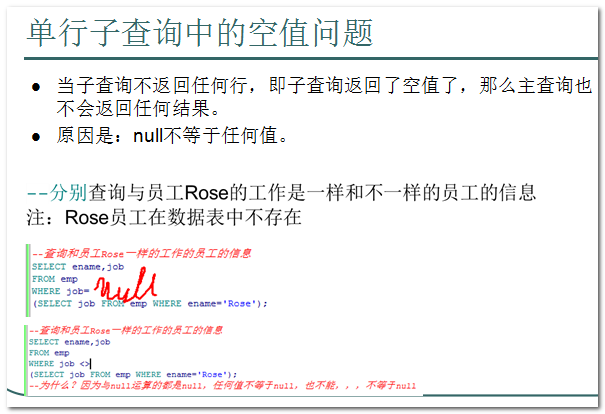
【示例】

*-查询部门名称是saleS的员工信息*

|  |
| --- |
| *--查询部门名称是saleS的员工信息*  SELECT \* FROM emp WHERE deptno=(SELECT deptno FROM DEPT WHERE dname ='SALES') |

了解：子查询可以是一张表的数据，也可以是不同表的数据。

### 空值问题



【代码】

需求:查找工作和'Rose' 这个人的工作一样的员工信息

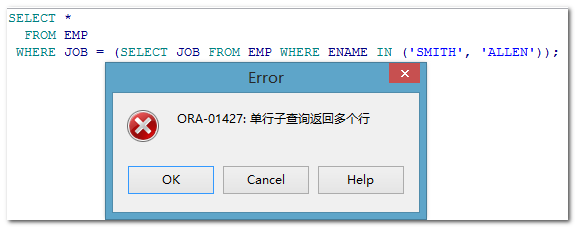
需求:查找工作和'Rose' 这个人的工作不一样的员工信息

|  |
| --- |
| ----需求:查找工作和'Rose' 这个人的工作一样的员工信息  SELECT \* FROM emp WHERE job =(SELECT job FROM emp WHERE ename='ROSE');  --需求:查找工作和'Rose' 这个人的工作不一样的员工信息  SELECT \* FROM emp WHERE job <>(SELECT job FROM emp WHERE ename='ROSE');--与null运算的都是null |

注意：使用子查询的时候，一定要保证子查询不能为空，否则数据就会出现异常。

### 非法使用单行子查询

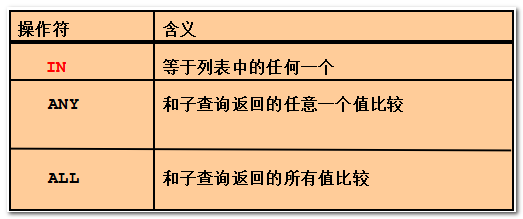
【示例】需求:查找工作和'SMITH' 'ALLEN' 这两个人的工作一样的员工信息



## 多行子查询

### 语法要求：

* 返回多行。
* 使用多行比较操作符。



### In操作符

【示例】

需求:查找工作和'SMITH' 'ALLEN' 这两个人的工作一样的员工信息

--需求:查找工作和'SMITH' 'ALLEN' 这两个人的工作不一样的员工信息

|  |
| --- |
| *--需求:查找工作和'SMITH' 'ALLEN' 这两个人的工作一样的员工信息*  SELECT JOB FROM emp WHERE ename IN('SMITH','ALLEN');  SELECT \* FROM emp WHERE job IN(SELECT JOB FROM emp WHERE ename IN('SMITH','ALLEN'));  *--需求:查找工作和'SMITH' 'ALLEN' 这两个人的工作不一样的员工信息*  SELECT \* FROM emp WHERE job NOT IN(SELECT JOB FROM emp WHERE ename IN('SMITH','ALLEN')); |

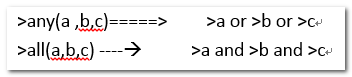
### Any和all操作符

【示例】需求:查询工资比30号部门任意一个员工的工资高的员工信息。--面试题

【示例】需求:查询工资比30号部门 所有 员工的工资高的员工信息。

|  |
| --- |
| *--需求:查询工资比30号部门任意一个员工的工资高的员工信息。--面试题*  SELECT \* FROM emp WHERE deptno =30;  *--任意一个：比最低的那个高就ok*  SELECT \* FROM emp WHERE sal >(SELECT MIN(sal) FROM emp WHERE deptno=30);  *--any(多行函数)*  SELECT \* FROM emp WHERE sal >ANY(SELECT sal FROM emp WHERE deptno=30);  *--【示例】需求:查询工资比30号部门 所有 员工的工资高的员工信息。*  SELECT \* FROM emp WHERE sal>(SELECT MAX (sal) FROM emp WHERE deptno=30);  *--all(多个返回记录)--max(sal)*  SELECT \* FROM emp WHERE sal>ALL(SELECT sal FROM emp WHERE deptno=30); |

分析结果：



## 子查询注意事项

|  |
| --- |
| * 关于格式：子查询要包含在括号内，最好有合理的书写风格。      * 子查询的位置：可以放在主查询的where、select、having、from的后面。不可以放在主查询的group by后面。 * 子查询和主查询可以是同一张表，也可以不是是不同一张表，只要子查询返回的结果在主查询中能使用即可。 * 关于使用操作符：单行操作符对应单行子查询，多行操作符对应多行子查询。 * 执行顺序：一般子查询先执行，再执行主查询； * 关于排序：一般不在子查询中使用order by；但在top-N分析问题中，必须在子查询中使用order by。 * 多行子查询一般用于from后面，作为一张新的虚拟临时表来使用。 |

虚拟临时表是临时表的一种，是运行过程中，内存中虚拟出来的一张临时表，用于sql的操作。

【示例】

|  |
| --- |
| *--虚拟表*  SELECT \* FROM  (  SELECT \* FROM emp WHERE deptno=30 *--虚表:将查询结果再作为一张表来使用。*  ) t  WHERE sal>2000 |

## 子查询和多表关联查询的选择

理论上，在都可以实现需求的情况下尽量选择多表查询。

原因：子查询会操作两次，多表查询只操作一次。多表的效率高。

但要注意的是，多表查询如果产生了笛卡尔集（语句上要注意条件的使用），则会出现严重的效率问题。

一般不在子查询中使用排序（order by），但在top-N分析问题中必须在子查询中使用排序。

# 伪列

## 什么是伪列

* 伪列是在ORACLE中的一个虚拟的列。
* 列的数据是由ORACLE进行维护和管理的，用户不能对这个列修改，只能查看。
* 所有的伪列要得到值必须要显式的指定。

最常用的两个伪列：rownum和rowid。

## ROWNUM

ROWNUM（行号）：是在查询操作时由ORACLE为每一行记录自动生成的一个编号。

每一次查询ROWNUM都会重新生成。（查询的结果中Oracle给你增加的一个编号，根据结果来重新生成）

rownum永远按照默认的顺序生成。（不受orderby的影响）

rownum只能使用< <=，不能使用> >=符号，原因是：Oracle是基于行的数据库，行号永远是从1开始，即必须有第一行，才有第二行。

### 行号的产生

【示例】需求：查询出所有员工信息，并且显示默认的行号列信息。

|  |
| --- |
| *--需求：查询出所有员工信息，并且显示默认的行号列信息。*  SELECT ROWNUM,t.\* FROM emp t;*--\* 和指定的列一起显示的时候，必须加别名* |

提示两点：

* ROWNUM是由数据库自己产生的。
* ROWNUM查询的时候自动产生的。

### 行号的排序

【示例】需求：查询出所有员工信息，按部门号正序排列，并且显示默认的行号列信息。

|  |
| --- |
| *--需求：查询出所有员工信息，按部门号正序排列，并且显示默认的行号列信息。*  SELECT ROWNUM,t.\* FROM emp t ORDER BY deptno;*--order by 的原理：将查询结果（此时行号已经有了，已经和每一行数据绑定了）进行排序。*  *- --order by是查询语句出来的结果之后再排序的，，rownu是在查询出来结果的时候产生。order by不会影响到行号*  *--先排序，再查询*  SELECT ROWNUM,t.\* FROM  (  SELECT \* FROM emp ORDER BY deptno  ) t |

结论：

order by排序，不会影响到rownum的顺序。rownum永远按照默认的顺序生成。

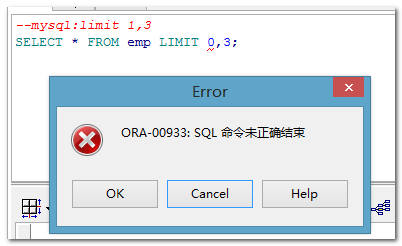
所谓的“默认的顺序”，是指系统按照记录插入时的顺序（其实是rowid）。

### 利用行号进行数据分页

回顾mysql如何排序？

|  |
| --- |
| select \* from table limit m,n  其中m是指记录开始的index，从0开始，表示第一条记录  n是指从第m+1条开始，取n条。  select \* from tablename limit 3,3  即取出第4条至第6条，3条记录 |

Oracle如何分页呢？



结论：Mysql使用limit的关键字可以实现分页，但Oracle没有该关键字，无法使用该方法进行分页。

【示例】需求：根据行号查询出第四条到第六条的员工信息。

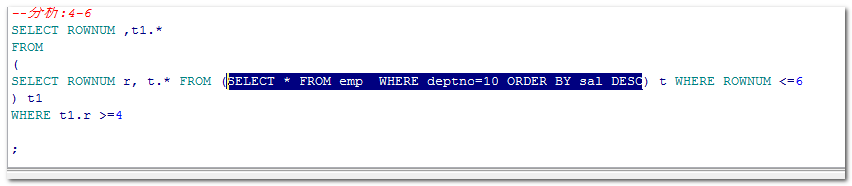
|  |
| --- |
| *--需求：根据行号查询出第四条到第六条的员工信息。*  SELECT ROWNUM,t.\* FROM emp t;  SELECT ROWNUM,t.\* FROM emp t WHERE ROWNUM >=4 AND ROWNUM<=6;  *--rownum只能使用< <=，不能使用> >=符号，原因是：Oracle是基于行的数据库，行号永远是从1开始，即必须有第一行，才有第二行。*  SELECT ROWNUM,t.\* FROM emp t WHERE ROWNUM<=6;  *--方案：可以使用子查询*  SELECT rownum,t2.\* FROM  (  SELECT ROWNUM r,t.\* FROM emp t WHERE ROWNUM<=6*--此时子查询的rownum已经变成了虚表的一个列*  ) t2*--尽量让虚表尽量小*  WHERE t2.r >=4  *--需求：要分页查询，每页3条记录，查询第二页*  */\**  *pageNum=2*  *pageSize=3*  *计算：*  *firstIndex=pageSize\*(pageNum-1);*  *maxCount=pageSize；*  *mysql:*  *limit 起始索引firstIndex,最大记录数maxCount*  *Oracle：*  *//起始行号*  *firstRownum=pageSize\*(pageNum-1)+1*  *//结束行号*  *endRownum=firstRownum+pageSize-1*  *具体计算：*  *firstRownum=3\*(2-1)+1=4;*  *endRownum=4+3-1=6;*  *\*/*  *--写Oracle的分页，从子查询写起，也就是说从小于等于写起，或者说从endRownum写起*  SELECT ROWNUM ,t2.\* FROM  (  SELECT ROWNUM r,t.\* FROM emp t WHERE ROWNUM <=6  ) t2 WHERE t2.r >=4;  *--优化*  SELECT \* FROM  (  SELECT ROWNUM r,t.\* FROM emp t WHERE ROWNUM <=6  ) WHERE r >=4;  SELECT empno,ename,job FROM*--结果指定字段*  (  SELECT ROWNUM r,t.\* FROM emp t WHERE ROWNUM <=6  ) WHERE r >=4;  *--按照薪资的高低排序再分页*  SELECT \* FROM  (  SELECT ROWNUM r,t.\* FROM emp t WHERE ROWNUM <=6 ORDER BY sal DESC  ) WHERE r >=4 ;  SELECT \* FROM emp ORDER BY sal DESC;  *--先排序薪资，再分页*  SELECT \* FROM  (  SELECT ROWNUM r,t.\* FROM  (SELECT \* FROM emp ORDER BY sal DESC) t  WHERE ROWNUM <=6 ORDER BY sal DESC  )  WHERE r >=4 ;*--Hibernate会自动将所有数据封装到实体对象（多余出来的行号那一列不会封装）*  *--如果不需要额外的字段，则只需要指定特定的列名就可以了。*  *--优化：子查询字段尽量少一些。数据量少。比如，表中有100个字段，但你就想显示5个，那么，你就子查询中直接指定5个就ok了。但，使用orm框架的建议都查出来。*  SELECT \* FROM  (  SELECT ROWNUM r,t.\* FROM  (SELECT ename,job,sal FROM emp ORDER BY sal DESC) t  WHERE ROWNUM <=6 ORDER BY sal DESC  )  WHERE r >=4 ;  *--通用*  SELECT \* FROM  (  SELECT ROWNUM r,t.\* FROM  (SELECT ename,job,sal FROM emp ORDER BY sal DESC) t  WHERE ROWNUM <=endRownum ORDER BY sal DESC  )  WHERE r >=firstRownum ;  */\**  *另外一种计算方法（索引算法）*  *firstIndex=pageSize\*(pageNum-1);*  *endRownum=firstIndex+pageSize；*  *\*/*  SELECT \* FROM  (  SELECT ROWNUM r,t.\* FROM  (SELECT ename,job,sal FROM emp ORDER BY sal DESC) t  WHERE ROWNUM <=endRownum ORDER BY sal DESC  )  WHERE r > firstIndex;*--Hibernate的内置算法* |

分析原因：

rownum只能使用< <=，不能使用> >=符号，原因是：Oracle是基于行的数据库，行号永远是从1开始，即必须有第一行，才有第二行。

【提示】：

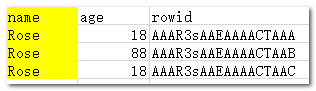
* 如何记忆编写Oracle的分页？建议写的时候从里到外来写，即先写小于的条件的子查询（过滤掉rownum大于指定值的数据），再写大于的条件的查询（过滤掉rownum小于的值）。
* Oracle的分页中如果需要排序显示，要先排序操作,再分页操作。(再嵌套一个子查询)
* 性能优化方面:建议在最里层的子查询中就直接指定字段或者其他的条件，减少数据的处理量。



## ROWID

ROWID（记录编号）：是表的伪列，是用来唯一标识表中的一条记录，并且间接给出了表行的物理位置，定位表行最快的方式。

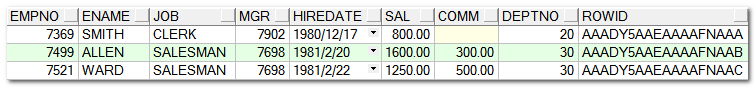
* 主键:标识唯一的一条业务数据的标识。主键是给业务给用户用的。不是给数据库用的。
* 记录编号rowid：标识唯一的一条数据的。主要是给数据库用的。类似UUID。



### ROWID的查看

【示例】

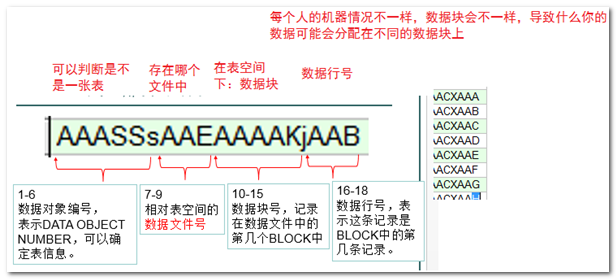
|  |
| --- |
| SELECT t.\*,ROWID FROM emp t; |



### ROWID的产生

使用insert语句插入数据时，oracle会自动生成rowid并将其值与表数据一起存放到表行中。

这与rownum有很大不同，rownum不是表中原本的数据，只是在查询的时候才生成的。



提示：rownum默认的排序就是根据rowid

### ROWID的作用

这里列举两个常见的应用：

* 去除重复数据。--面试题—了解
* 在plsql Developer中，加上rowid可以更改数据。

关于主键和rowid的区别：

相同点：为了标识唯一一条记录的。

不同点：

主键：针对业务数据，用来标识不同的一条业务数据。

rowid：针对具体数据的，用来标识不同的唯一的一条数据，跟业务无关。

【示例】需求：删除表中的重复数据，要求保留重复记录中最早插入的那条。（DBA面试题）

|  |
| --- |
| *--查看rowid*  SELECT t.\*,ROWID FROM emp t;  *--需求：删除表中的重复数据，要求保留重复记录中最早插入的那条。（DBA面试题）*  *--准备测试表和测试数据：*  *--参考建表语句如下：*  *-- Create table*  create table test  (  id number,  name varchar2(50)  )  ;  *--插入测试数据*  INSERT INTO TEST VALUES(1,'xiaoming');  INSERT INTO TEST VALUES(2,'xiaoming');  INSERT INTO TEST VALUES(3,'xiaoming');  COMMIT;  SELECT \* FROM TEST ;  *--通过rowid，剔除重复xiaoming，保留最早插入的xiaoming*  SELECT t.\*,ROWID FROM TEST t;  *--删除的的时候，可以先查询你要删除的东东*  SELECT t.\*,ROWID FROM TEST t WHERE ROWID > (SELECT MIN(ROWID) FROM TEST);  DELETE FROM TEST t WHERE ROWID > (SELECT MIN(ROWID) FROM TEST);  *--语句有缺点：条件不足，会只保留一条数据，误删其他数据*  *--重新插入测试数据*  INSERT INTO TEST VALUES(1,'xiaoming');  INSERT INTO TEST VALUES(2,'xiaoming');  INSERT INTO TEST VALUES(3,'xiaoming');  INSERT INTO TEST VALUES(4,'Rose');  INSERT INTO TEST VALUES(5,'Rose');  COMMIT;  *--剔除重复数据*  SELECT \* FROM TEST WHERE ROWID NOT in(SELECT MIN(ROWID) FROM TEST GROUP BY NAME);  DELETE TEST WHERE ROWID NOT in(SELECT MIN(ROWID) FROM TEST GROUP BY NAME); |

注意:删除重复记录一定要小心,万一你的条件有问题,就会删错数据.建议删除之前，可以先用查询查一下，看是否是目标数据。

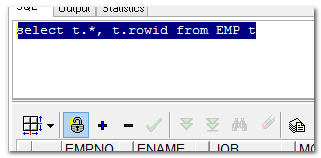
数据一旦删除恢复比较麻烦，但可以恢复，采用日志回滚。一般不要轻易用。

# 数据处理

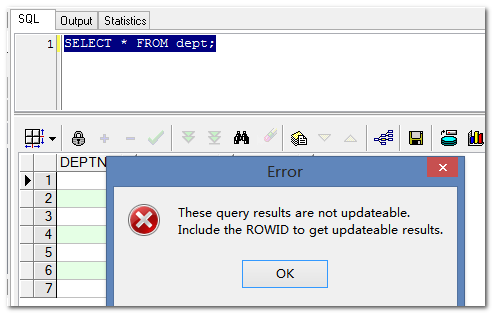
说完了所有的查询，下面说说增、删、改。

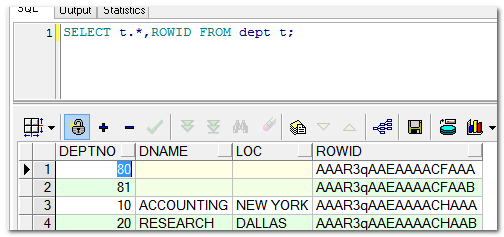
## Update

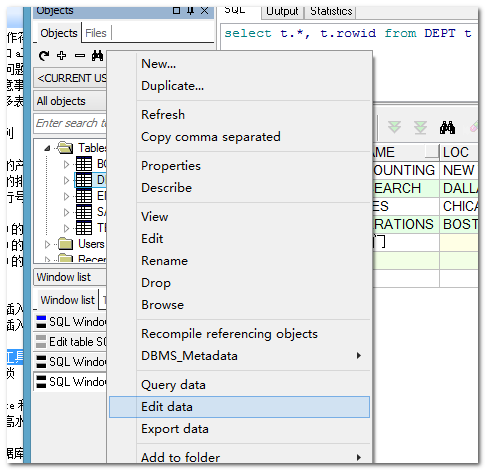
* 使用工具进行更新数据的操作。（通过rowid伪列）



通过工具修改数据







提示：是否能使用工具修改，主要看语句有没有rowid。

## Insert

### 批量插入

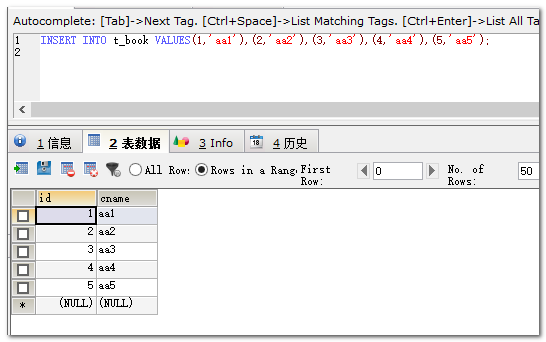
语法：

|  |
| --- |
| INSERT INTO table VALUES*--单条插入语法*  INSERT INTO table SELECT 查询语句*--批量插入语法(主要用于将一张表中的数据批量插入到另外一张表中)* |

【示例】需求：将dept表中部门名称不为空的数据都插入到test表中

|  |
| --- |
| *--需求：将dept表中部门"名称"不为空的数据都插入到test表中*  INSERT INTO TEST(ID,NAME) SELECT deptno,dname FROM dept;*--select的结果必须能插入到目标表中。（字段个数要对应、字段类型要对应）*  INSERT INTO TEST SELECT deptno,dname FROM dept ;*--必须前后字段对应*  *--非法使用批量插入*  INSERT INTO TEST SELECT deptno,dname,loc FROM dept ; |

Mysql的批量（Oracle不支持）：



## Delete

### Delete和truncate区别-面试题

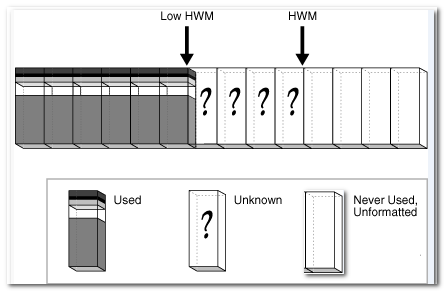
* delete逐条删除，truncate先摧毁表，再重建 。
* 最根本的区别是：delete是DML（可以回滚,还能闪回），truncate是DDL（不可以回滚 ，后面的所事务会讲回滚）
* Delete不会释放空间，truncate会（当确定一张表的数据不再使用，应该使用truncate）
* delete会产生碎片，truncate不会。

友情提示：面试经常会被问道。

### Hwm-高水位

高水位线英文全称为high water mark，简称HWM，那什么是高水位呢 ？

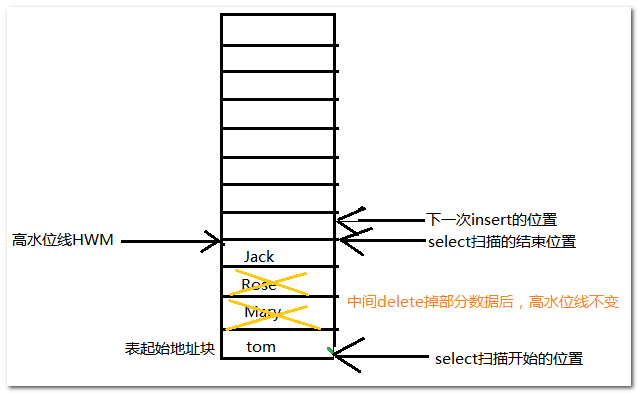
在Oracle数据的存储中，可以把存储空间想象为一个水库，数据想象为水库中的水。水库中的水的位置有一条线叫做水位线，在Oracle中，这条线被称为高水位线（High-warter mark, HWM）。在数据库表刚建立的时候，由于没有任何数据，所以这个时候水位线是空的，也就是说HWM为最低值。当插入了数据以后，高水位线就会上涨，但是这里也有一个特性，就是如果你采用delete语句删除数据的话，数据虽然被删除了，但是高水位线却没有降低，还是你刚才删除数据以前那么高的水位。也就是说，这条高水位线在日常的增删操作中只会上涨，不会下跌。



【高水位对Oracle的应用有什么影响呢？】

高水位对查询有巨大的影响。而且还浪费空间。

【解读Oracle中Select语句的特性】



极端例子：数据库有10w条数据，删掉了前面的99999个，我select查询的时候，还是需要扫描10w次，虽然表中只有一条数据。效率还是非常的低！！！！！

如何解决高水位带来的查询效率问题呢？

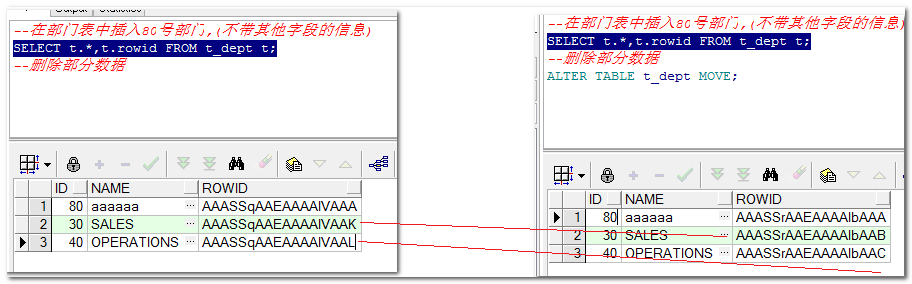
1. 将表数据备份出来，摧毁表再重建（truncate table），然后再将数据导回来。
2. 收缩表，整理碎片，可使用变更表的语句：alter table 表名 move

【示例】查看、测试、消除高水位—了解

|  |
| --- |
| *--之前查看rowid*  SELECT t.\*,ROWID FROM TEST t;    *--对表进行分析，收集统计信息（执行了收集信息的动作，user\_tables表的块字段才有数据）*  analyze table TEST compute statistics;  *--查询表数据的块信息，其中blocks是高水位，empty\_blocks是预申请的块空间。*  select table\_name,blocks,empty\_blocks from user\_tables where table\_name='TEST';  *--收缩表(整理碎片)，降低高水位，消除行移植和行链接，不释放申请的空间*  ALTER TABLE TEST MOVE;  *--对表进行分析，收集统计信息（执行了收集信息的动作，user\_tables表的块字段才有数据）*  analyze table TEST compute statistics;  *--查询表数据的块信息，其中blocks是高水位，empty\_blocks是预申请的块空间。*  select table\_name,blocks,empty\_blocks from user\_tables where table\_name='TEST';  *--之后查看rowid*  SELECT t.\*,ROWID FROM TEST t; |

结论：

* 收缩表之后，高水位线下降了。
* 收缩表之后，rowid发生了变化。



注意：

* move最好是在空闲时做，记得move的是会产生锁的(如果你move的时候需要很长事件，那么别人是不能操作这张表的。排他锁)
* move以后记得重建index（后续讲到索引，你会知道索引存放的其实就是数据的地址信息。当数据的地址变动了，索引也会失效。）语法：ALTER INDEX 索引名字 REBUILD;

# 数据库事务

## 什么是数据库事务？

事务是保持数据的一致性，它由相关的DDL或者DML语句做为载体，这组语句执行的结果要么一起成功，要么一起失败。

## 事务的特性

SQL92标准定义了数据库事务的四个特点（ACID）：

原子性 (Atomicity) :一个事务里面所有包含的SQL语句是一个执行整体，不可分割，要么都做，要么都不做

一致性 (Consistency) :事务开始时，数据库中的数据是一致的，事务结束时，数据库的数据也应该是一致的

隔离性 (Isolation): 多个事务并发的独立运行，而不能互相干扰，一个事务修改，新增，删除数据在根据当前事务的事务隔离级别基础上，其余事务能看到相应的结果（这里为什么这么说，下面我会给我具体的例子进行分析）

持久性 (Durability) : 事务被提交后，数据会被永久保存（具有数据恢复的功能）

## 事务的开始和结束

Oracle的默认事务开启和结束是跟mysql不一样的。

回顾：mysql的事务是如何开启的？

MySQL默认采用autocommit模式运行。这意味着，当您执行一个用于更新（修改）表的语句之后，MySQL立刻把更新存储到磁盘中，不需要手动提交。

如果需要手动管理事务，需要显式的关闭自动事务：Set autocommit false，然后显式的手动开启事务：START TRANSACTION，直到手动COMMIT或ROLLBACK结束事务为止。

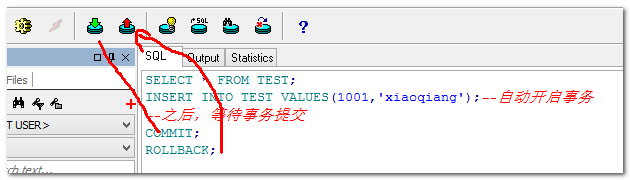
那么，Oracle的事务是如何开启的？

Oracle的事务默认是手动管理事务(不支持自动事务)，事务是自动开启（不需要显式的开启，隐式开启），但一般需要手动提交关闭。

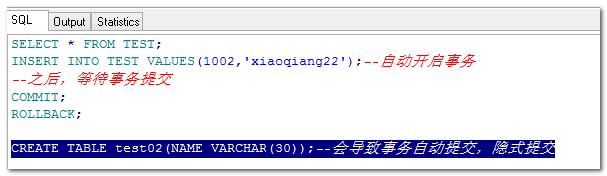
Oracle事务的开始和结束的触发条件：

* 事务的开始：以第一个DML语句(insert update delete)的执行作为开始，即是自动开启的事务。
* 事务的结束（以下条件之一）：
  + 显式结束：commit， rollback（还是隐式commit）
  + 隐式结束（自动提交）：DDL（create table…）和DCL（所以不能回滚），exit(事务正常退出)
  + 隐式回滚（系统异常终止）：关闭窗口，死机，掉电。

工具上的事务按钮：



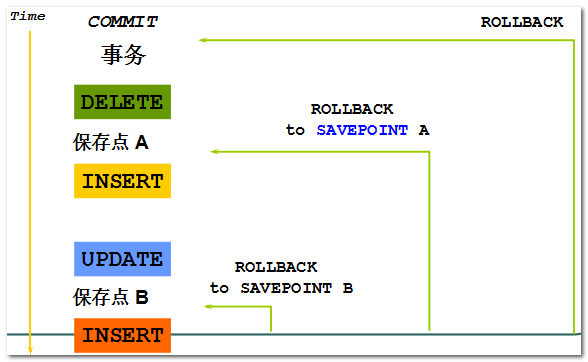
隐式提交：



提示：一般情况下，我们尽量使用手动提交事务。

## 控制事务-保留点SAVEPOINT—了解

事务过程中是可以控制的，通过SAVEPOINT语句。



SAVEPOINT的作用：

* 使用 SAVEPOINT 语句在当前事务中创建保存点，语法：SAVEPOINT 保留点名称。
* 使用 ROLLBACK 语句回滚到创建的保存点。语法：ROLLBACK TO 保留点名称。

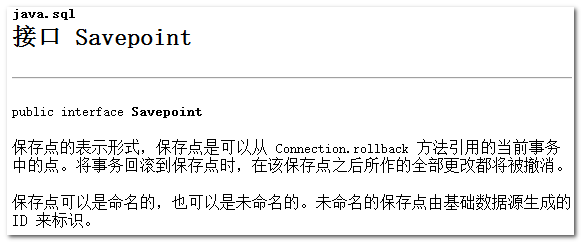
【示例演示】

|  |
| --- |
| SELECT \* FROM TEST;  INSERT INTO TEST VALUES(85,NULL);  SELECT \* FROM TEST;  SAVEPOINT aa;*--保留点*  INSERT INTO TEST VALUES(86,NULL);*--后悔了，不插入了*  SELECT \* FROM TEST;  *--回滚*  ROLLBACK TO aa;  SELECT \* FROM TEST;  INSERT INTO TEST VALUES(87,NULL);  SELECT \* FROM TEST;  *--提交*  COMMIT;  SELECT \* FROM TEST; |

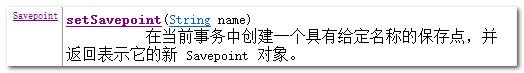
注：

当前事务提交后，事务中所有的保存点将被释放。

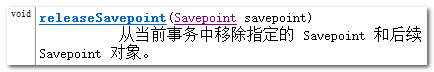
JAVA中也有关于保留点的API，



具体调用采用Connection对象来操作它，相关方法如下：



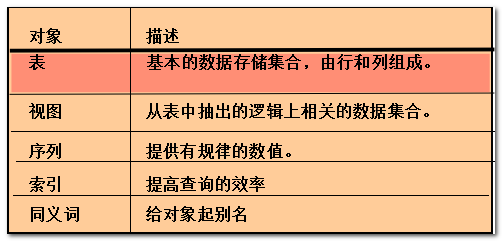




# 数据库对象-表（TABLE）

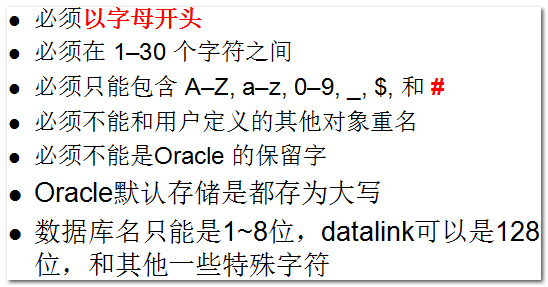
## 什么是数据库对象？

数据库对象，是数据库的组成部分，有表（Table ）、索引（Index）、视图（View）、用户（User）、触发器（Trigger）、存储过程（Stored Procedure）、图表（Diagram）、缺省值（Default）、规则（Rule）等。



## 表的命名规则和命名规范

表名和列名的基本规范如下：



【扩展】

另外，每个公司都有自己特有的命名规范，比如，要求所有的数据库对象都要加上一个前缀，用于快速识别对象的类别。

比如表的命名：

t\_person :存放个人信息的表。

t\_crm\_person:存放客服子系统模块的人员信息表。

视图的命名：

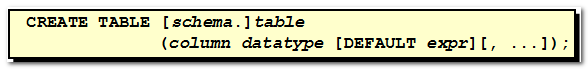
v\_person：用来查询人员信息的视图。

命名规范的作用：

* 良好的命名规范便于识别和管理，对于系统开发和维护都有很大的帮助。
* 使用工具的提示功能也更容易快速定位到所需要的对象。

## 创建表CREATE TABLE

### 基本语法

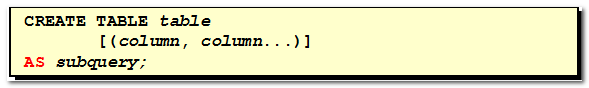


创建表的要求条件：

* 必须具备CREATE TABLE权限、存储空间。
* 必须指定表名、列名、数据类型、数据类型的大小

### 复制表

语法：



Create table 新表 as select from 旧表 条件(如果条件不成立,则只复制结构,如果条件成立,复制结构和值)

使用子查询创建表，而且还可以将创建表和插入数据结合起来。

【示例】

*--复制一张和原来一模一样的新表，包含数据*

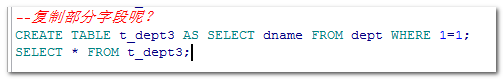
|  |
| --- |
| *--复制一张和原来一模一样的新表，包含数据*  CREATE TABLE t\_dept AS SELECT \* FROM dept;  SELECT \* FROM t\_dept  *--复制一张和原来一模一样的新表，不要数据，只要结构*  CREATE TABLE t\_dept\_empty AS SELECT \* FROM dept WHERE 1<>1;  SELECT \* FROM t\_dept\_empty;  *---t\_dept,在现网(正式环境)确实要 测试一下数据.一般我们可以在建立一张和这个一模一样的表.c*  *--能不能只复制部分字段建立新表？可以！*  *--复制表的部分字段*  CREATE TABLE t\_dept\_part AS SELECT deptno,dname FROM dept;  SELECT \* FROM t\_dept\_part; |

提示：

复制表有没有数据的区别，是select子句结果有没有数据。如果子句结果没有数据，则仅创建表，如果有数据，则创建表的同时也插入符合条件的数据。

注意：

* 指定的列和子查询中的列要一一对应
* 通过列名和默认值定义列



关于where 1=1的写法，一般我们用来拼凑条件的。

1<>1是为了营造一个永远不成立的条件。

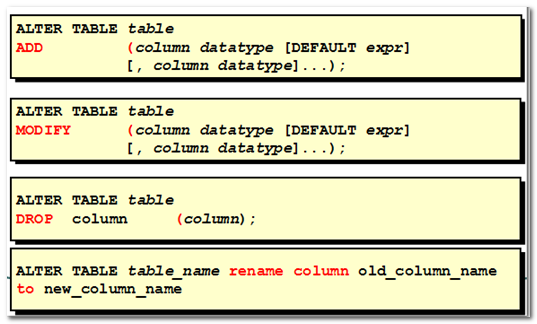
### Oracle的数据类型



## 修改表ALTER TABLE

### 基本语法

修改表的列：



修改表的列的能力：

* 追加新的列
* 修改现有的列
* 删除一个列

修改对象的名称：



作用：

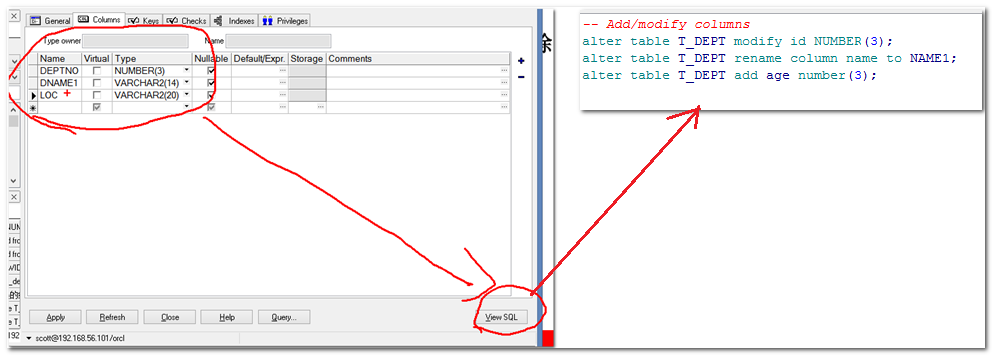
* 执行RENAME语句改变表, 视图, 序列, 或同义词的名称。
* 要求必须是对象的拥有者

【示例】

|  |
| --- |
| RENAME t\_dept TO t\_dept\_new;  Table renamed. |

### 通过工具来修改表

操作方式：



## 删除表DROP TABLE

### 基本语法



注意：

* 数据和结构都被删除。
* 所有正在运行的相关事物被提交。（ddl语句）
* 所有相关索引被删除。（表附属对象会被删除）
* DROP TABLE 语句不能回滚，但是可以闪回。

完整的oracle数据库的版本的情况下，普通用户删除的表，会自动放入回收站



你可以从回收站还原（闪回）。

友情提示：

日常操作中，删除操作一定要小心，一旦删除了且没有放入回收站，则意味着数据的丢失！

记住一句话：数据无价！！！

# 约束

## 约束的概念作用

* 约束是可以更好的保证数据库数据的完整性和一致性的一套机制。
* 约束可以限制加入表的数据的类型。
* 如果存在依赖关系，约束可以防止错误的删除数据，也可以级联删除数据。

数据库的约束可以认为是对表的数据的一种规则。

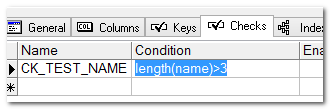
## 约束创建的时机

* 创建表的时候，同时创建约束。
* 表结构创建完成后，可以再添加约束。

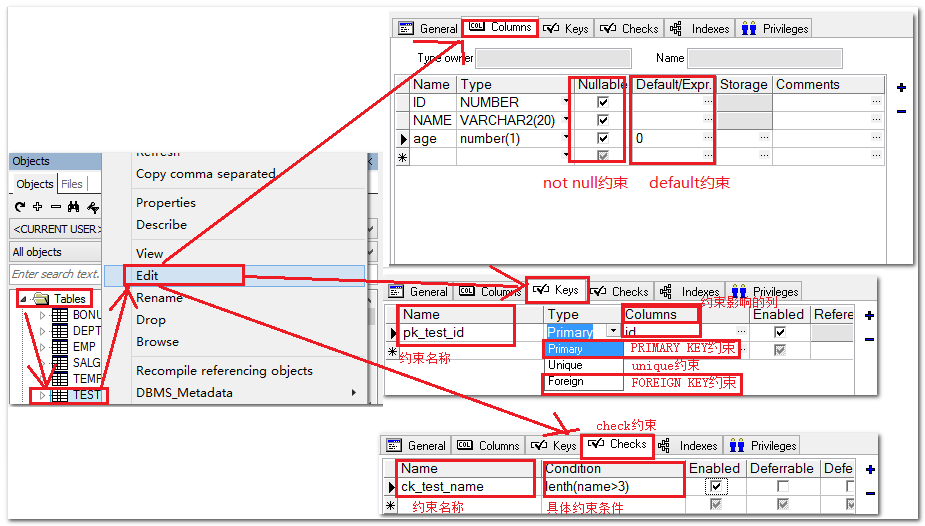
## 常见的约束类型

* NOT NULL
* UNIQUE
* PRIMARY KEY
* FOREIGN KEY
* DEFAULT
* CHECK—用来检查一个字段的值是否符合某表达式,表达式的结果必须是布尔值。

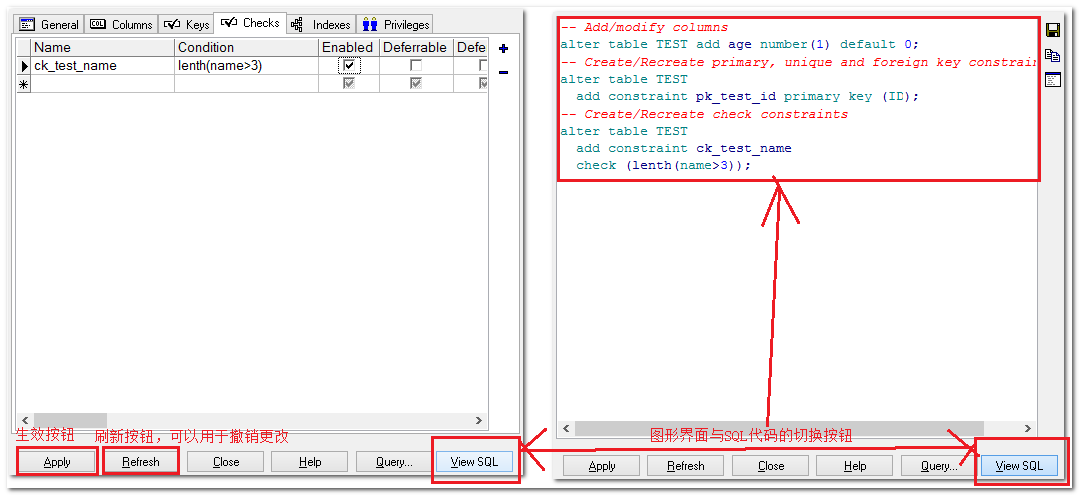
其中：check约束是Oracle特有的约束。



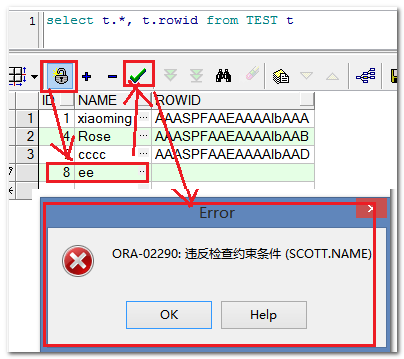
## 通过工具快速添加约束



通过工具快速得到SQL的代码：



插入数据测试Check约束



## 约束的应用选择

在应用开发中，主键约束一般要设置，其他如非空、唯一、默认值、检查等约束，可以根据实际情况来添加。而外键约束是否要设置，是存在一点争议的。（争议在性能上）

一般建议：

* 在大型系统中（性能要求不高，安全要求高），可以使用外键；在大型系统中（性能要求高，安全自己控制），不用外键；小系统随便。
* 不用外键的话，可以用程序控制数据一致性和完整性，可以在代码的数据层通过代码来保证一致性和完整性。
* 用外键要适当，不能过分追求。

从JAVA开发的角度上说,一般不建议使用外键，除了性能外，使用程序控制业务更灵活。

比如客户和订单,这两个之间的关联虽然可以建立外键关系,实现级联效果（如级联删除）。

* 如果有外键约束，则删除客户的时候，必须先删除客户下的订单，否则，不允许删除。
* 从数据完整一致性的角度上说，如果客户被删除了，订单也无意义了，这是合理的。
* 但从业务角度上说，客户被删除了，是否意味这订单也必须删除呢？单纯保留订单的行为也是合理的。

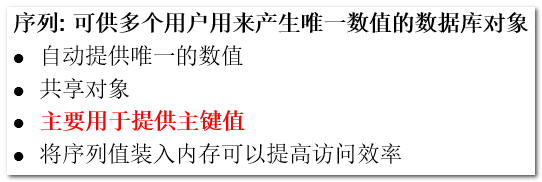
# 序列-sequence

需求:

Mysql中主键有自增长的特性.

Oracle中,主键没有自增长这个特性.那么如何解决这个问题.使用序列可以解决.

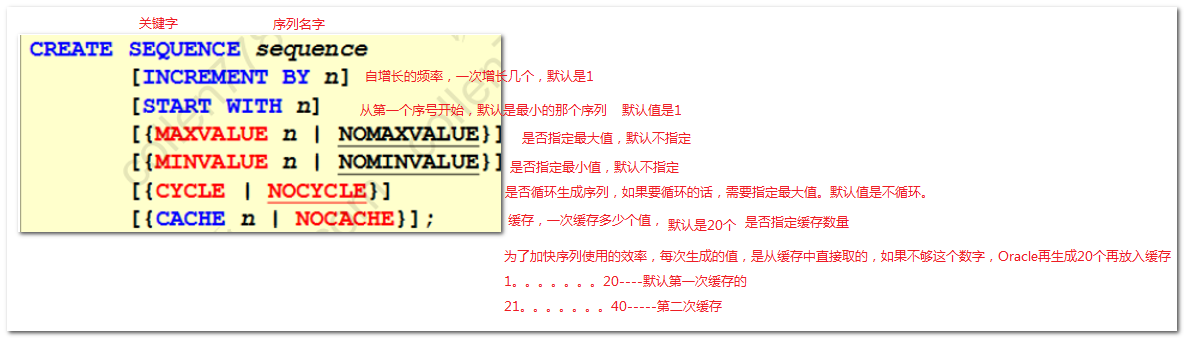
## 概念和作用



这个是Oracle特色的。Mysql是没有的.

简单的说，他可以用来高效的生成主键值。

## 语法



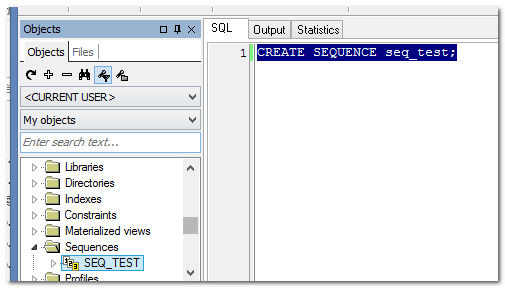
将序列提前装入内存，可以提高效率。

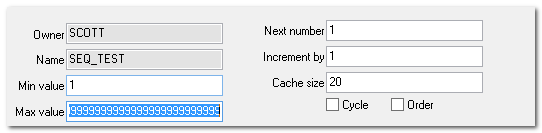
## 创建序列

【示例】

创建一个简单的序列

|  |
| --- |
| CREATE SEQUENCE seq\_test; |





## 序列的使用

在ORACLE中为序列提供了两个伪列：

1. NEXTVAL 获取序列对象的下一个值（指针向前移动一个，并且获取到当前的值。）
2. CURRVAL 获取序列对象当前的值

【示例】

|  |
| --- |
| 为什么？  原因是：序列初始化之后指针在第一个数之前。必须先向前移动才可以查询的到。  数组的指针默认在1之前，并没有指向第一个值，要想使用必须向前移动一下。（指针只能向前不能向后）  [1,2,3….20][  \*  操作指针：  [1,2,3….20][  \*  SELECT seq\_test.nextval FROM dual;    移动一位并且取值。 |

## 序列的应用

Oracle建表的时候是否能像mysql那样设定一个自增长的列吗？

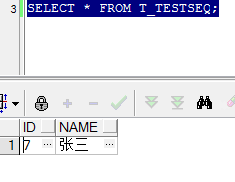
不行！

那如何解决呢？使用序列！

【示例】在插入数据的时候插入序列主键.

|  |
| --- |
| *--在插入数据的时候插入序列主键.*  INSERT INTO TEST VALUES(seq\_test.nextval,'Jack'); |

问题：为什么这个值不是从1开始？



原因:是个独立对象.谁都能用,谁都能共享它.

## 序列的裂缝

1. 序列是一个共有对象，多个表都可以调用。但实际开发中，可以避免多个表用一个序列（创建多个序列）。序列是独立的对象。任意表都可以使用，但是编号就不能保证有序。

2，当插入记录时报错，序列对象值也被使用，下一次再使用时，序列的值就会+1

【示例】序列的裂缝

|  |
| --- |
| INSERT INTO T\_TESTSEQ VALUES(seq\_test.nextval,'张三1');  ROLLBACK;  INSERT INTO T\_TESTSEQ VALUES(seq\_test.nextval,'张三2');  COMMIT;  SELECT \* FROM T\_TESTSEQ; |

也就是说，用序列插入数据库的值不一定是连续的。

补充：

Mysql的自增长列也可以是不连续的.

序列出现裂缝的条件：

* 事务回滚。
* 系统异常。
* 多个表同时使用同一个序列。

这个序列是公用的对象。如果你很在意的话，就一个表用一个序列，但大多数情况下，这个主键值（代理主键）没有什么意义的话，可以多个表公用一个序列。

# Oracle的体系结构-了解

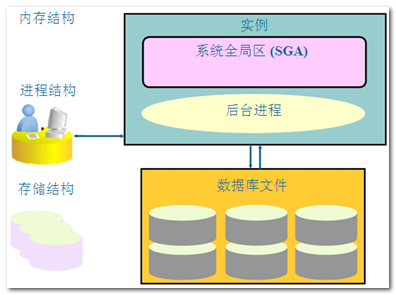
## Oracle数据库和Oracle实例

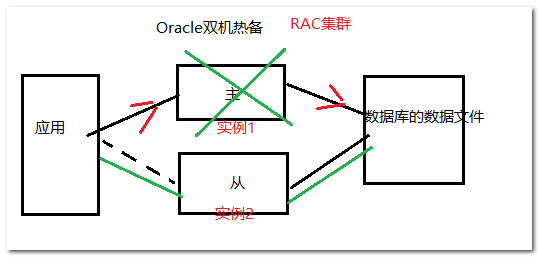
Oracle 服务器软件部分由两大部分组成, Oracle 数据库 和 Oracle 实例。

两者的解释如下：

* Oracle 数据库(物理概念): 位于硬盘上实际存放数据的文件和相应的程序文件, 这些文件组织在一起, 成为一个逻辑整体, 即为 Oracle 数据库. 因此在 Oracle 看来, “数据库” 是指硬盘上文件的逻辑集合, 必须要与内存里实例合作, 才能对外提供数据管理服务。
* Oracle 实例（逻辑概念）: 位于物理内存里的数据结构. 它由一个共享的内存池和多个后台进程所组成, 共享的内存池可以被所有进程访问. 用户如果要存取数据库(也就是硬盘上的文件) 里的数据, 必须通过实例才能实现, 不能直接读取硬盘上的文件。实例的唯一标识也称之为SID（service id）（OSID）。

一个实例只能对应一个数据库，一个数据库可以有多个实例（RAC集群），但大多数情况下, 一个数据库上只有一个实例对其进行操作。我们就是通过连接到实例来操作数据库的。





## Oracle常见的存储文件

常见的存储文件主要为三类：

* 数据文件。存储数据用的。

例：表

* 控制文件。记录数据文件存放的位置。例：数据库名称、数据文件名称及位置。



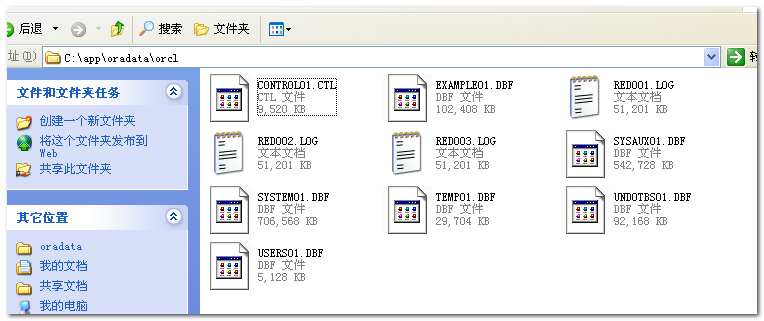
* 日志文件。记录数据信息变化的。例：因故障问题造成一些数据没有及时写入到数据文件，可以使用日志文件恢复(Oracle日志回滚:如果你的数据被delete掉并且提交了,数据还是可以恢复的,可以通过日志来恢复的)。



# 表空间(Tablespace)的管理

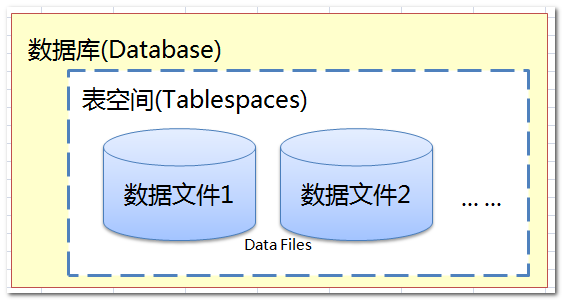
## 表空间的概念

ORACLE是属于文件存储。ORACLE中的数据是存放在一个个数据文件中，数据文件存放在磁盘中。



如果说数据文件是物理概念，那么表空间就是逻辑概念，Oracle通过表空间来对数据文件中的数据进行CRUD。

表空间是一种逻辑结构，是Oracle最大的逻辑单元，可以理解为：`。

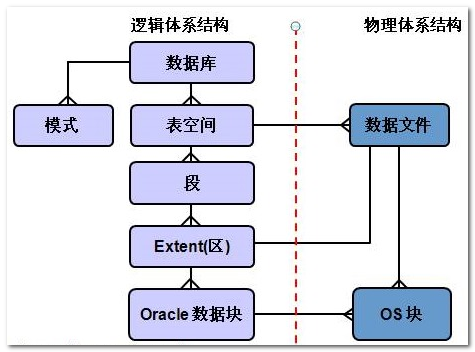


表空间的属性特点：

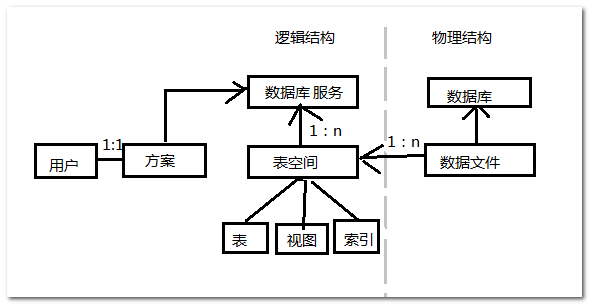
* 一个数据库可以包含多个表空间，一个表空间只能属于一个数据库。
* 一个表空间可以由一个或多个数据文件组成，一个数据文件只能属于一个表空间。
* 表空间可以划分成更细的逻辑存储单元。（了解）

## 数据库的存储结构--了解

官方数据存储结构图：



各对象之间的存储对应关系图：

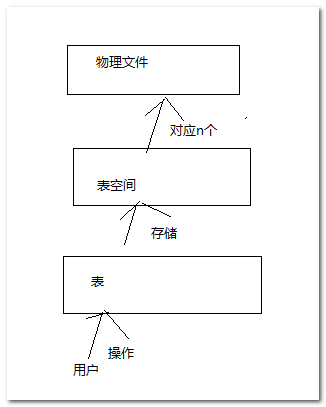


* 所有的数据库对象都存储在表空间中，而表空间被数据库服务管理。
* 一个表空间可以对应N个数据文件，表空间是逻辑概念，而数据文件是物理概念。
* 方案（SCHEMA模式）是表、视图、索引等数据库对象的逻辑集合，它通过数据库服务来间接管理这些对象。
* 一个用户（user）创建时会同时创建一个同名的方案（schema），即，你甚至可以认为用户和方案是同一个东西（事实上不是，用户主要是做权限等相关管理的）。当用户登录后，就立刻拥有了该同名方案下所有对象。
* 方案（用户）和表空间没有什么必然关系，一个方案拥有一个默认的表空间，但同时可以使用多个表空间来存储它的对象。一个表空间可以为不同的方案来存储其所属对象

【补充阅读】下面有个很形象的比喻，是从网上摘的，不妨一看：

我们可以把database看做是一个大仓库，仓库分了很多很多的房间，schema就是其中的房间，一个schema代表一个房间，table可以看做是每个schema中的床，table被放入每个房间中，不能放置在房间之外，那岂不是晚上睡觉无家可归了，然后床上可以放置很多物品，就好比table上可以放置很多列和行一样，数据库中存储数据的基本单元是table，显示中每个仓库放置物品的基本单位就是床，user就是每个schema的主人，（所以schema包含的是object，而不是user），user和schema是一一对应的，每个user在没有特别指定下只能使用自己schema的东西，如果一个user想使用其他schema的东西，那就要看那个schema的user有没有给你这个权限了，或者看这个仓库的老大（DBA）有没有给你这个权限了。换句话说，如果你是某个仓库的主人，那么这个仓库的使用权和仓库中的所有东西都是你的，你有完全的操作权，可以扔掉不用东西从每个房间，也可以防止一些有用的东西到某个房间，你还可以给每个user分配具体的权限，也就是他到某一个房间能做些什么，是只能看（read-only),还是可以像主人一样有所有控制权（R/W），这个就要看这个user所对应的角色Role了。

数据操作的过程：



【小结】

表空间:属于一种逻辑结构。记录物理文件的逻辑单位。是Oracle最大的逻辑单位.

也就是说,我们所有的数据都存储在表空间中.

## 常见的表空间分类—了解

* （永久）数据表空间,主要用来永久存储正式的数据文件。
* 临时数据表空间,主要用来存储临时数据的，比如数据的排序、分组等产生的临时数据，不能存放永久性对象。
* UNDO表空间，保存数据修改前的镜象。

临时表空间和UNDO表空间的异同：（了解）

* 相同之处：两者都不会永久保存数据。
* 不同之处：UNDO表空间用于存放UNDO数据,当执行DML操作时,oracle会将这些操作的旧数据写入到UNDO段,以保证可以回滚和事务隔离读取等，主要用于数据的修改等；而临时表空间主要用来做查询和存放一些缓冲区数据。

## Oracle对表空间的管理方式—了解

字典管理：全库所有的空间分配都放在数据字典中。容易引起字典争用，而导致性能问题。

本地管理：空间分配不放在数据字典，而在每个数据文件头部的第3到第8个块的位图块，来管理空间分配。oracle公司推荐使用本地管理表空间。

## 表空间的创建

注：表空间的创建一般是由DBA来操作完成的，而且需要管理员权限（我们一般用sys）。

三种表空间中，UNDO表空间通常是由Oracle自动化管理的，而另外两种表空间则一般需要手动创建。

【常用参数语法】：

|  |
| --- |
| --创建永久数据表空间  CREATE TABLESPACE TABLESPACE\_NAME  [DATAFILE DATAFILE1，[DATAFILE 2]…]  [LOGGING | NOLOGGING]  [ONLINE|OFFLINE]  [EXTENT\_MANAGEMENT\_CLAUSE]  参数说明：   * TABLESPACE\_NAME，表空间名称随意，但最好遵循一定的规范，如tbl\_baby18\_dat、tbl\_baby18\_tmp等。 * DATAFILE，表空间的类型 * DATAFILE1 数据文件需要有如下格式：文件名 SIZE 初始文件大小 [AUTOEXTEND OFF| ON] [MAXSIZE|NEXT SIZE MAXSIZE SIZE]   + 文件名是数据文件的路径名，可以是绝对路径，也可以是相对路径，如“路径\xxx.dbf”，注意路径必须先建立好。   + 初始化文件大小，是数据文件刚建立起来的时候所占物理磁盘空间的大小；   + AUTOEXTEND，是否自动扩展数据文件的大小，OFF表示关闭自动扩展，数据文件只能是初始大小，ON表示开启自动扩展，当数据文件超过初始大小的时候，会自动增大。默认值为OFF。   + 如果设置自动扩展，则需要设置最大值MAXSIZE，如设置2000m，当然也可以设置为UNLIMITED，表示无限表空间。如果要指定每次扩展的大小，可以使用NEXT SIZE MAXSIZE SIZE语法，表示每次扩展多少尺寸，最大能扩展到多大（大小上限）。 * [LOGGING | NOLOGGING]该子句用来声明这个表空间上所有的用户对象的日志属性，即当操作包括表，索引，分区，物化视图，物化视图上的索引，分区等是否记录日志。缺省值为LOGGING。 * [ONLINE|OFFLINE]表空间的状态，ONLINE表示表空间创建后立即有效，OFFLINE表示表空间创建后暂时无效，即不能使用，只有设置为ONLINE后才有效，默认值为ONLINE。 * EXTENT\_MANAGEMENT\_CLAUSE表空间如何管理范围，推荐设置为本地管理，值为EXTENT MANAGEMENT LOCAL，如果不指定该值，则ORACLE会根据初始化时内部的其他参数进行自动设置一个默认值，生产环境下建议指定该值为本地管理。   --创建临时数据表空间  CREATE TEMPORARY TABLESPACE TABLESPACE\_NAME  TEMPFILE DATAFILE1，[DATAFILE 2]…  EXTENT\_MANAGEMENT\_CLAUSE  参数说明：   * DATAFILE1数据文件的格式语法：文件名 SIZE 初始文件大小，注意临时数据表空间的数据文件一般不需要指定最大值，Oracle对其采用了贪吃算法策略，因此，该表空间会自动逐渐增大。当然你也可以手动指定。 * 其他参数见永久数据表空间的。 * 临时表空间默认是不记日志的。 |

【最简语法】

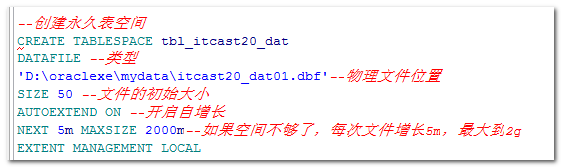
|  |
| --- |
| Create tablespace 表空间名称  表空间类型 ‘物理文件全路径’  Size 初始文件大小 |

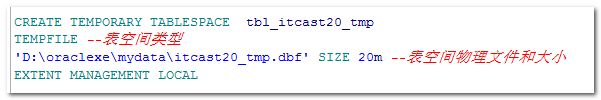
【示例】

永久数据表空间和临时数据表空间的建立。

|  |
| --- |
| *--建立一个数据表空间。*  CREATE TABLESPACE tbl\_baby19\_dat  DATAFILE 'D:\oraclexe\mydata\baby19\_dat01.dbf' SIZE 50m  AUTOEXTEND ON NEXT 5m MAXSIZE 2000m  EXTENT MANAGEMENT LOCAL  *--创建临时数据表空间。*  CREATE TEMPORARY TABLESPACE tbl\_baby19\_tmp  TEMPFILE 'D:\oraclexe\mydata\baby19\_tmp.dbf' SIZE 20m  EXTENT MANAGEMENT LOCAL |

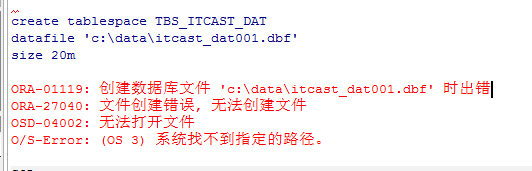
解释：





【提示】：

文件路径（data目录）必须提前存在，否则：



注意：

实际企业开发中，不要用最简化的方式来进行表空间的创建。

【参考示例1】

|  |
| --- |
| --创建数据表空间  CREATE TABLESPACE TBS\_CSP\_BS\_DAT  DATAFILE '/dev/rlv\_dat001' SIZE 2000M REUSE AUTOEXTEND OFF,  '/dev/rlv\_dat002' SIZE 2000M REUSE AUTOEXTEND OFF  LOGGING  ONLINE  PERMANENT  EXTENT MANAGEMENT LOCAL;  --创建临时数据表空间  CREATE TEMPORARY TABLESPACE TBS\_CSP\_BS\_TMP  TEMPFILE '/dev/rlv\_dat009' SIZE 2000M REUSE AUTOEXTEND OFF  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 10M; |

注：

PERMANENT是显式的指定创建的是永久的表空间，用来存放永久对象。默认值。

【参考示例2】

|  |
| --- |
| --创建数据表空间  create tablespace tbs\_user\_data  logging  datafile 'D:\oracle\oradata\Oracle9i\user\_data.dbf'  size 50m  autoextend on  next 50m maxsize 20480m  extent management local;  --创建临时数据表空间  create temporary tbs\_user\_temp  tempfile 'D:\oracle\oradata\Oracle9i\user\_temp.dbf'  size 50m  autoextend on  next 50m maxsize 20480m  extent management local; |

## 删除表空间

语法：



|  |
| --- |
| DROP TABLESPACE |

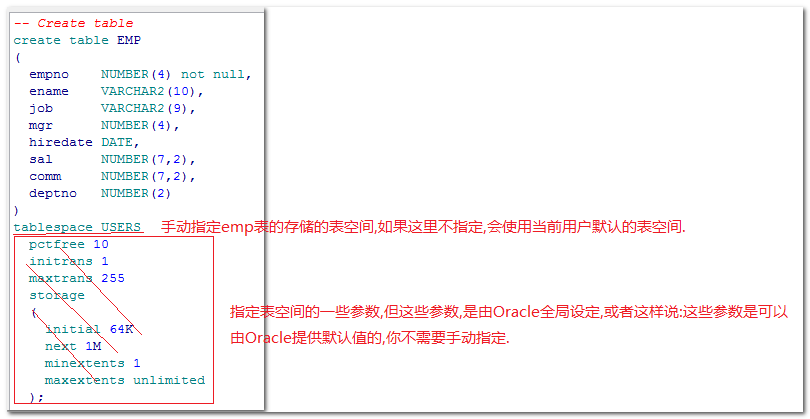
【示例】

|  |
| --- |
| --删除表空间以及下面所有数据和数据文件（全删，寸草不生）  DROP TABLESPACE tbs\_baby14\_dat INCLUDING CONTENTS AND DATAFILES; |
| 提示：如果不加后面的一堆，则，只是将表空间进行了逻辑删除（Oracle无法管理使用这个表空间了，但数据文件还存在）。 |

## 表空间的一个应用

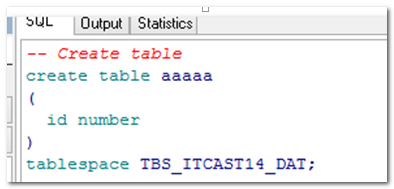
【示例】

建立表的时候指定表空间。



企业开发中，一定不要用默认的表空间，一定使用要指定表空间。

最简的一个建表脚本：



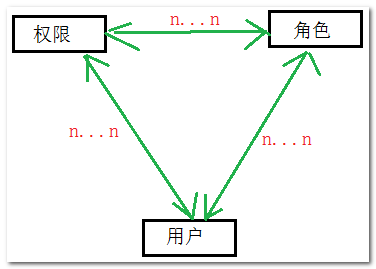
注意:

写建表的语句的时候,可以指定存储的表空间,但不建议指定表空间的参数。

# 用户和权限

## 用户角色权限的关系

预备知识：



## 预定义用户（账户）

Oracle预定义有很多用户，用于不同的用途。这些用户大都默认是禁用的（如scott，hr等），但有两个最重要的用户是默认开启的，这两个用户就是SYS和SYSTEM。

* SYS 帐户(数据库拥有者):
  + 拥有 DBA 角色权限
  + 拥有 ADMIN OPTION 的所有权限
  + 拥有 startup, shutdown, 以及若干维护命令
  + 拥有数据字典
* SYSTEM 帐户
  + 拥有 DBA 角色权限.

注意：这些帐户通常不用于常规操作。

Sys和system账户的区别：

* sys用户是数据库的拥有者，是系统内置的、权限最大的超级管理员帐号。
* system用户只是拥有DBA角色权限的一个管理员帐号，其实它还是归属于普通用户。

## 操作用户

* 创建用户的语句

create user 用户名

identified by 密码(不要加引号)

default tablespace 默认表空间名 quota 5M on 默认表空间名

[temporary tablespace 临时表空间名]

[profile 配置文件名] //配置文件

[default role 角色名] //默认角色

[password expire] //密码失效

//如果设置失效，那么第一次登录的时候，会提醒你更改密码。

[account lock] //账号锁定（停用）

* 修改用户

alter user 用户名 identified by 密码 quota 10M on 表空间名

alter user 用户名 account lock/unlock

* 删除用户

drop user 用户名 [cascade].如果要删除的用户中有模式对象，必须使用cascade.

【示例】最简方式创建一个用户

|  |
| --- |
| 切换到sys用户下： |

注：未指定的参数都采用默认值。

【示例】借助工具创建一个用户

|  |
| --- |
| 创建用户的时候指定的表空间，会成为以后在该用户下建立对象（表）的默认存储表空间。    *-- 语句：Create the user*  */\*创建用户并指定表空间 \*/*  create user baby19  identified by baby19  default tablespace TBL\_BABY19\_DAT  temporary tablespace TBL\_BABY19\_TMP; --上锁解锁改密码等 |

注意：

一般企业开发中，建表要手动指定表空间，可以让不同模块、不同功能的对象存储在不同的数据文件中，可以提高性能。

【示例】删除用户

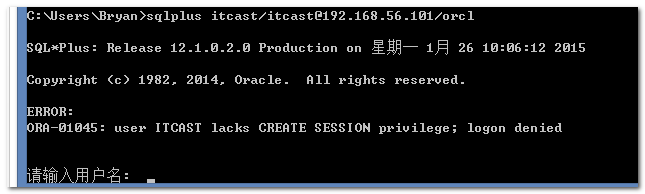
|  |
| --- |
| --删除用户及其下面所有的对象  drop user babytest cascade; |

提示，每个数据库用户帐户具备：

* 一个唯一的用户名
* 一个验证方法
* 一个默认的表空间
* 一个临时表空间
* 权限和角色
* ~~每个表空间的配额.~~

## 配置角色和权限

使用上面创建的用户登录测试：

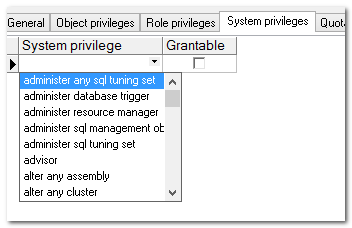


结果报错。

提示说：该用户没有创建会话的权限，登录被拒绝。

那该如何赋权呢？赋什么权限呢？

Oracle内置有大量的权限属性：

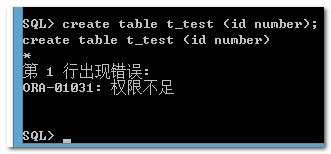


常见权限：



我们可以将create session权限赋权给新建的用户.新建的用户就可以登录了.

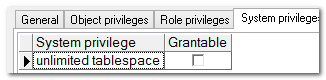
我们再建立一张表看看:



结果又提示是权限不足。= =...

再添加建表的权限：





再次测试建表：



再添加一条数据看看：



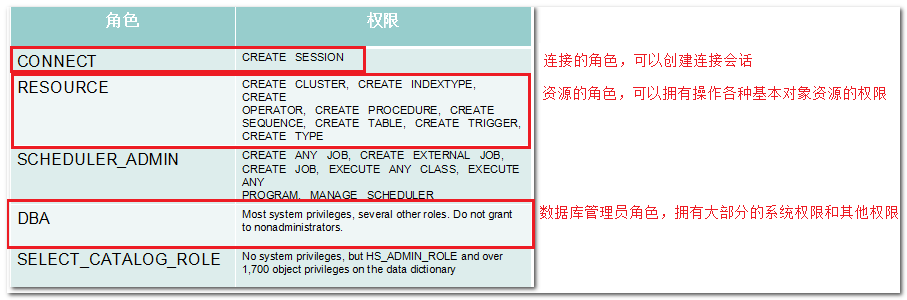
竟然又没有权限！

。。。

结论：这样一个个添加权限非常的麻烦！

是否可以使用比较简单的方式将普通用户的权限赋予给一个用户呢？

可以！通过预定义内置角色就可以实现。



如何选择预定义的角色呢？

普通用户就选择：connect和Resource角色即可。

管理员用户选择：connect和Resource、dba角色。

|  |
| --- |
| */\*给用户授予权限 \*/*  grant connect,resource to username; |

再次登录、各种操作测试，均正常了！

【提示】

如果遇到这个错误：



说明当前用户没有操作该表空间的权限，需要手动加入这个权限：





梳理回顾建立一个普通用户的过程：

1.创建用户—2.赋权限（connect和resourece角色）

## Oracle用户(user)和方案(schema)

几个概念：

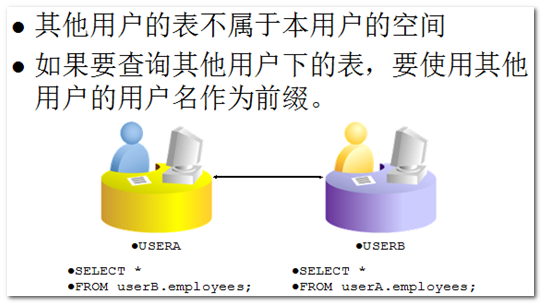
* 方案就是属于某一用户的所有对象（表、视图等）的集合.
* 用户名和方案名往往是通用的.
* 一个用户只能关联一个方案.
* 创建用户时系统会自动创建一个同名方案（schema）

提示：

Scott用户的方案名也是scott，因此，后面我们将这两个概念放在一起用，即我们可以说，某表是scott用户下的对象，也可以说是scott方案下的对象。

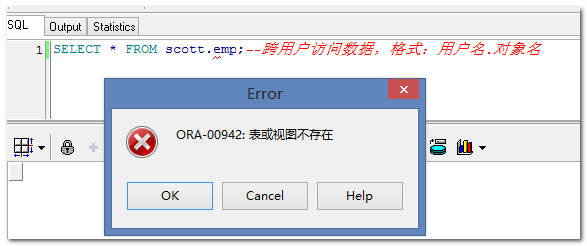
## 跨域访问对象

跨域访问也称之为跨用户访问、跨方案访问，访问的方式为：用户名.对象名，



如在baby用户下访问scott用户下的emp表的数据：

Select \* from scott.emp;



原因：没有对象访问权限。

Oracle用户的权限分为两种:

* 系统权限(System Privilege): 允许用户执行对于数据库的特定行为，例如：创建表、创建用户等
* 对象权限(Object Privilege): 允许用户访问和操作一个特定的对象，例如：对其他方案下的表的查询

【示例】需求：baby20用户要读取scott用户中emp表的数据。

|  |
| --- |
| --scott用户登录赋权：    --Sql语句：    --baby用户登录测试： |

注意：

赋权的时候，只能是自己拥有的权限、或者该权限是可以传递的，才可以将其赋予别人。

# 视图VIEW

问题：

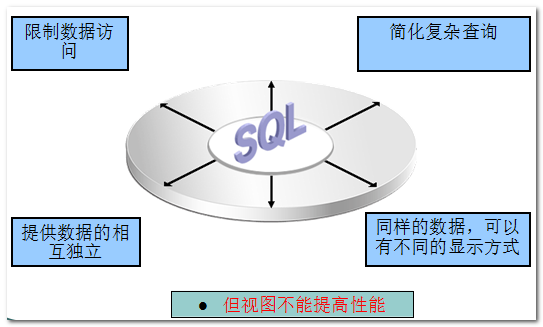
baby24用户现在只需要查询10部门的员工数据就行了，scott也不想将所有数据都开放给babyt用户。

## 视图的概念和作用

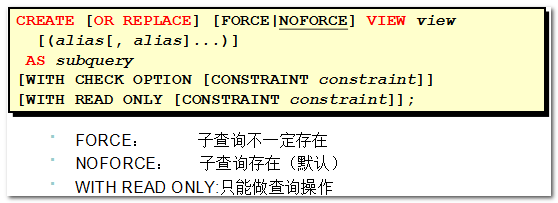
概念：

* 视图是一种虚表.
* 视图建立在已有表的基础上, 视图赖以建立的这些表称为基表。
* 向视图提供数据内容的语句为 SELECT 语句, 可以将视图理解为存储起来的 SELECT 语句.
* 视图向用户提供基表数据的另一种表现形式

作用：



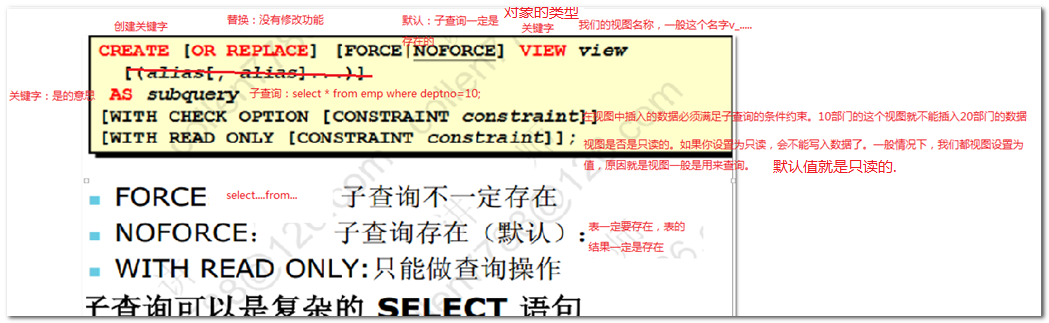
## 语法



提示：

子查询可以是任意复杂的 SELECT 语句。

语法详细分析：



纠正：默认值不是只读。

## 操作视图

视图只能创建、替换和删除，不能修改。

### 创建视图

【示例】创建10号部门的员工信息视图

|  |
| --- |
| --sql语句建立视图  CREATE VIEW v\_emp10  AS  SELECT \* FROM emp WHERE deptno=10  ; |

提示：如果提示权限不足而导致无法添加视图，则需要添加权限，一般为学习方便，我们会直接添加dba角色权限。

|  |
| --- |
| --切换到sys用户下，为scott添加dba权限： |

### 查询视图

【示例】查询视图

|  |
| --- |
|  |

【示例】视图的真实内容查看

|  |
| --- |
|  |

结论：

可以看出，视图的本质就是sql语句。

### 替换视图

视图没有修改功能。

【示例】要将视图改为可以查询10号部门的员工信息且工资要大于2000：

|  |
| --- |
| CREATE OR REPLACE VIEW v\_emp10  AS  SELECT \* FROM emp WHERE deptno=10 AND sal >2000; |

提示：

平时我们在编写建立视图的语句时候，一般直接把replace加上，即直接CREATE OR REPLACE。

### 删除视图

【示例】删除10好部门的这个视图

|  |
| --- |
| DROP VIEW v\_emp10;  SELECT \* FROM v\_emp10; |

## 几个参数说明

|  |
| --- |
| *--先创建视图再创建表:一般用的不多，一般我们都是先有表再创建视图。*  CREATE OR REPLACE FORCE VIEW v\_test2015  AS  SELECT \* FROM test2015;  SELECT \* FROM v\_test2015  *--视图默认情况下和表一样，拥有表类似的功能，可以crud*  SELECT t.\*,ROWID FROM v\_emp10 t;  SELECT \* FROM emp;  CREATE OR REPLACE VIEW v\_emp10  AS  SELECT \* FROM emp WHERE deptno=10  WITH CHECK OPTION;*--数据的增加和修改，必须满足子查询的条件*  *--一般视图，我们主要用来查询的，一般不维护它。*  CREATE OR REPLACE VIEW v\_emp10  AS  SELECT \* FROM emp WHERE deptno=10  WITH READ ONLY; |

## 只读视图

一般情况下，视图主要用来提供查询的，并不希望用户去修改它，因此，我们可以创建只读视图。创建只读视图只需要添加with read only 选项即可，这样就可以屏蔽对视图的DML操作。

【示例】将已有的视图修改为只读视图

|  |
| --- |
| CREATE OR REPLACE VIEW v\_emp\_dept10  AS  SELECT \* FROM emp WHERE deptno=10 AND sal >2000  WITH READ ONLY ; |

友情提示：

其实，很多大的系统中，比如银行，某些客户会告诉你，这个表存这个数据，那个表存哪个数据，但实际上，可能不是真正的表，而是视图，而且还是只读的。

为什么给视图？原因是：

如果是存钱的表，那么放开给你，是不是非常危险。如果业务需要确实是需要更改这个表的数据呢?一般来调用存储过程(一般有提供,有一定特定功能,还能记录日志)来改表，为了安全！不能直接改表。

## 跨域访问视图

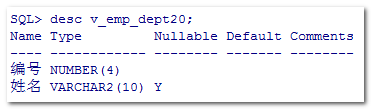
【示例】只放开scott下的emp表的部分数据给babyt14用户查询，开放的数据要求为：20号部门的员工，字段只显示员工号和姓名，且要求这两个字段的标题显示为中文。（要求本例使用工具来操作）

|  |
| --- |
| --在scott下创建视图（视图名称参考为：v\_emp\_dept20）    --将生成的脚本如下：  create or replace view v\_emp\_dept20 as  select empno "编号",ename "姓名"  from emp  where deptno=20  WITH READ ONLY;  --scott下查询验证一下：  SELECT \* FROM v\_emp\_dept20;    --将该视图赋予baby用户：在scott用户下操作：  grant select on v\_emp\_dept20 to baby;  --切换到babyt用户下进行查询验证：  Select \* from scott.v\_emp\_dept20; |

另外补充：

视图可以屏蔽筛选不同字段、字段名称等，因此，你看到的时候的字段也未必是真实表中存在的！

|  |
| --- |
| CREATE OR REPLACE VIEW v\_emp10  AS  SELECT empno 编号,ename empname FROM emp WHERE deptno=10  WITH READ ONLY; |



## 视图小结

视图和表的区别：

视图是实体表的映射，视图和实体表区别就是于视图中没有真实的数据存在。

什么时候使用视图：

1. 在开发中，有一些表结构是不希望过多的人去接触，就把实体表映射为一个视图。
2. 在项目过程中，程序人员主要关注编码的性能、业务分析这方面。对于一些复杂的SQL语句，设计人员会提前把这些语句封装到一个视图中，供程序人员去调用

注意:在企业中，你查询的对象（表）他可能不是一张的表，可能是视图；你看到的视图的字段可能也不是真实的字段。

# 同义词SYNONYM

问题：我们想伪装一下这个视图的名字，或者是嫌调用的这个对象名字太长，怎么办？

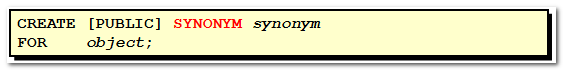
## 同义词的概念和作用

同义词就是（对象的）别名，可以对表、视图等对象起个别名，然后通过别名就可以访问原来的对象了。

作用：

* 方便访问其它用户的对象
* 缩短对象名字的长度

## 语法



## 操作同义词

同义词只有创建和删除操作。

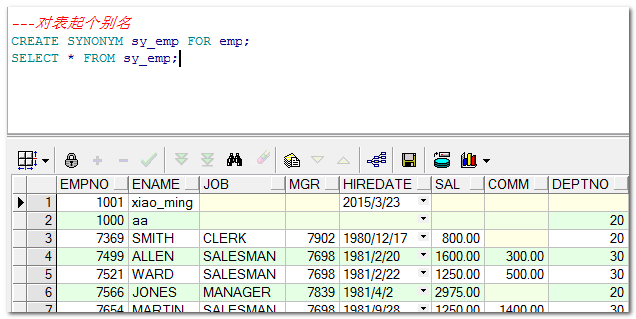
【需求】在baby用户下为视图scott.v\_emp\_dept20创建一个同义词emp20;

|  |
| --- |
| --查询验证： |

友情提示：

如果工作中,你遇到一张”表”来查询数据,那么它一定是表么?不一定,可能是视图,也可能是同义词.

另外，任何对象都能起别名。下面的例子对emp表起个别名：



重点：

1. 多表关联查询（oracle的语法，左外，右外 自连接）
2. 子查询：any和all的面试题，子查询和多表查询的选择（面试）
3. 分页：rownum+子查询！！！
4. rowid：删除重复数据（面试）
5. 两个新语法：批量插入(insert into table select ...) 复制表（create table tablename as select ....）
6. delete和truncate的区别（面试），高水位，如何消除高水位（truncate，move）
7. Oracle的事务和mysql的事务的不同（oracle在dml时隐式开启，必须手动提交（不建议隐式提交））
8. 约束的使用（外键是否要增加）（面试）
9. 序列：创建(create sequence 序列名字)和插入数据的使用。
10. 表空间-了解
11. 创建用户：创建用户+赋予角色（connect,resource，注意：unlimited tablespace权限如果没有加上）
12. 用户和方案（schema）的关系
13. 跨域访问
14. 视图，
15. 同义词