# 视图

## 什么是视图

视图是一种数据库对象，是从一个或者多个数据表或视图中导出的虚表，视 图所对应的数据并不真正地存储在视图中，而是存储在所引用的数据表中，视图的结构和数据是对数据表进行查询的结果。

根据创建视图时给定的条件，视图可以是一个数据表的一部分，也可以是多个基表的联合，它存储了要执行检索的查询语句的定义，以便在引用该视图时使用。

使用视图的优点：

1.简化数据操作：视图可以简化用户处理数据的方式。

2.着重于特定数据：不必要的数据或敏感数据可以不出现在视图中。

3.视图提供了一个简单而有效的安全机制，可以定制不同用户对数据的访问权限。

4.提供向后兼容性：视图使用户能够在表的架构更改时为表创建向后兼容接口。

## 创建或修改视图语法

|  |
| --- |
| CREATE [OR REPLACE] [FORCE] VIEW view\_name  AS  subquery  [WITH CHECK OPTION ]  [WITH READ ONLY] |

选项解释：

|  |  |
| --- | --- |
| OR REPLACE | 若所创建的试图已经存在，ORACLE 自动重建该视图 |
| FORCE | 不管基表是否存在 ORACLE 都会自动创建该视图 |
| subquery | 一条完整的 SELECT 语句，可以在该语句中定义别名； |
| WITH CHECK OPTION | 插入或修改的数据行必须满足视图定义的约束； |
| WITH READ ONLY | 该视图上不能进行任何 DML 操作 |

## 删除视图语法

|  |
| --- |
| DROP VIEW view\_name |

## 案例

### 简单视图的创建与使用

什么是简单视图？如果视图中的语句只是单表查询，并且没有聚合函数，我们就称之为简单视图。

需求：创建视图：业主类型为 1 的业主信息

语句：

|  |
| --- |
| create or replace view view\_owners1 as  select \* from t\_owners where ownertypeid=1 |

利用该视图进行查询

|  |
| --- |
| select \* from view\_owners1 where addressid=1; |

就像使用表一样去使用视图就可以了。

对于简单视图，我们不仅可以用查询，还可以增删改记录。

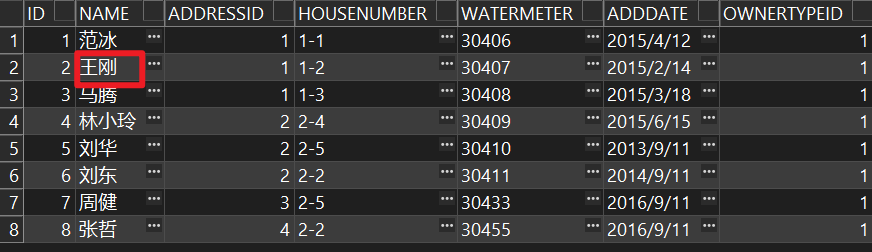
我们下面写一条更新的语句，试一下：

|  |
| --- |
| update view\_owners1 set name='王刚' where id=2; |

再次查询：

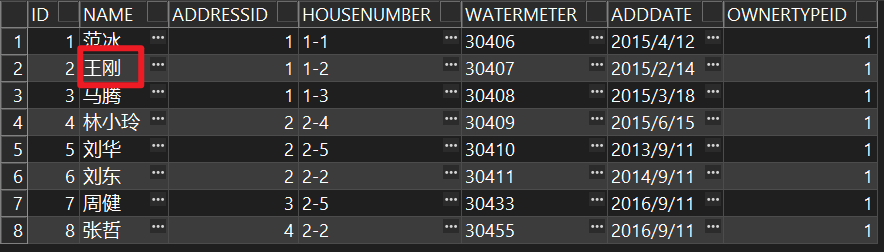
|  |
| --- |
| select \* from view\_owners1; |

查询结果如下：



结果已经更改成功。

我们再次查询表数据



发现表的数据也跟着更改了。由此我们得出结论：视图其实是一个虚拟的表，它的数据其实来自于表。如果更改了视图的数据，表的数据也自然会变化，更改了表的数据，视图也自然会变化。一个视图所存储的并不是数据，而是一条 SQL 语句。

案例

|  |
| --- |
| -- todo 1 创建视图: 查询 区域类型为1的地址信息 -- todo 2 使用视图: 查询 区域类型为1的地址信息 -- todo 3 修改视图的列值 存在 -- todo 4 修改视图的列值 不存在 -- todo 5 验证 |

实现

|  |
| --- |
| -- todo 1 创建视图: 查询 区域类型为 2 的地址信息 create or replace view view\_addr as select \* from T\_ADDRESS where AREAID=2 ;  -- todo 2 使用视图 查询 区域类型为1的地址信息 select \* from view\_addr;  -- todo 3 修改视图的列值 存在 update view\_addr set name='霍营地铁站' where id=5; commit;  -- todo 4 修改视图的列值 不存在 update view\_addr set name='流星花园' where id=1; commit;  -- todo 5 验证 select \* from view\_addr; select \* from T\_ADDRESS; |

修改视图中不存在但原表中存在的值，原表并不会改变

### 带检查约束的视图

需求：根据地址表（ T\_ADDRESS）创建视图 VIEW\_ADDRESS2 , 内容为区域 ID 为 2 的记录。

语句：

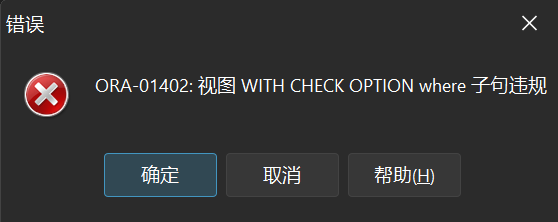
|  |
| --- |
| create or replace view view\_address2 as  select \* from T\_ADDRESS where areaid=2  with check option |

With check option的含义就是你建立的视图中的条件，不可以被改变，就是这个样子

执行下列更新语句：

|  |
| --- |
| update view\_address2 set areaid=1 where id=4 |

系统提示如下错误信息：



案例

|  |
| --- |
| -- todo 1 创建视图: 查询 区域类型为 3 的地址信息, 携带 with check option -- todo 2 使用视图 查询 区域类型为 3 的地址信息 -- todo 3 修改视图的列值 name=新值, 是否报错? -- todo 4 修改视图的列值 areaid=新值, 是否报错? -- todo 5 验证 |

实现

|  |
| --- |
| -- todo 1 创建视图: 查询 区域类型为 3 的地址信息, 携带 with check option create or replace view view\_addr\_2 as select \* from T\_ADDRESS where areaid=3 with check option ;  -- todo 2 使用视图 查询 区域类型为 3 的地址信息 select \* from view\_addr\_2;  -- todo 3 修改视图的列值 name=新值, 是否报错? no update view\_addr\_2 set name='霍营地铁站' where areaid=2; commit;  -- todo 4 修改视图的列值 areaid=新值, 是否报错?  update view\_addr\_2 set areaid=3 where id=3; commit;  -- todo 5 验证 select \* from view\_addr; select \* from T\_ADDRESS; |

### 只读视图的创建与使用

如果我们创建一个视图，并不希望用户能对视图进行修改，那我们就需要创建视

图时指定 WITH READ ONLY 选项，这样创建的视图就是一个只读视图。

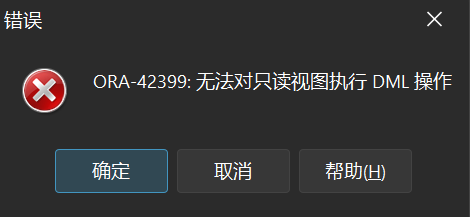
需求：将上边的视图修改为只读视图

语句：

|  |
| --- |
| create or replace view view\_owners1 as  select \* from T\_OWNERS where ownertypeid=1  with read only |

修改后，再次执行 update 语句，会出现如下错误提示

|  |
| --- |
| update view\_owners1 set name='和珅' where id=2; |



案例

|  |
| --- |
| -- todo 1 创建视图: 查询 区域类型为 3 的地址信息, 只读 -- todo 2 使用视图 查询 区域类型为 3 的地址信息 -- todo 3 修改视图的列值 name=新值, 是否报错? |

实现

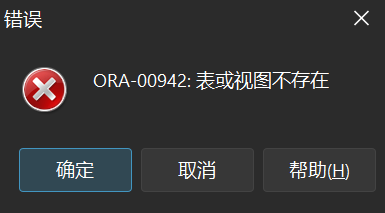
|  |
| --- |
| -- todo 1 创建视图: 查询 区域类型为 3 的地址信息, 只读 create or replace view view\_addr\_3 as select \* from T\_ADDRESS where areaid=3 with read only ;  -- todo 2 使用视图 查询 区域类型为 3 的地址信息 select \* from view\_addr\_3;  -- todo 3 修改视图的列值 name=新值, 是否报错? update view\_addr\_3 set name='西三期' where id=7; commit; |

### 创建带错误的视图

我们创建一个视图，如果视图的 SQL 语句所设计的表并不存在，如下

|  |
| --- |
| create or replace view view\_temp as  select \* from t\_temp |

T\_TEMP 表并不存在，此时系统会给出错误提示



有的时候，我们创建视图时的表可能并不存在，但是以后可能会存在，我们如果

此时需要创建这样的视图，需要添加 FORCE 选项，SQL 语句如下：

|  |
| --- |
| create or replace **force** view view\_temp as  select \* from t\_temp |

此时视图创建成功。

### 复杂视图的创建与使用

所谓复杂视图，就是视图的 SQL 语句中，有聚合函数或多表关联查询。

我们看下面的例子：

#### 多表关联查询的例子

需求：创建视图，查询显示业主编号，业主名称，业主类型名称

语句：

|  |
| --- |
| create or replace view view\_owners as  select o.id 业主编号, o.name 业主名称, ot.name 业主类型  from t\_owners o, t\_ownertype ot  where o.ownertypeid=ot.id |

使用该视图进行查询

|  |
| --- |
| select \* from view\_owners; |

那这个视图能不能去修改数据呢？

我们试一下下面的语句：

|  |
| --- |
| update view\_owners set 业主名称='范小冰' where 业主编号=1 |

可以修改成功。

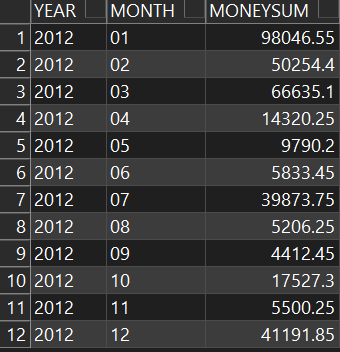
我们再试一下下面的语句：

|  |
| --- |
| update view\_owners set 业主类型='普通居民' where 业主编号=1; |

可以修改成功。

#### 分组聚合统计查询的例子

需求：创建视图，按年月统计水费金额，效果如下



语句：

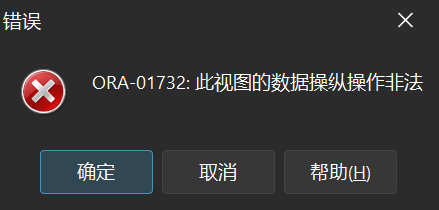
|  |
| --- |
| create view view\_accountsum as  select year,month,sum(money) moneysum  from t\_account  group by year, month  order by year, month |

那这个视图能不能去修改数据呢？

|  |
| --- |
| update view\_accountsum set year='2023' where month='01' |

发现修改失败。

此例用到聚合函数，没有键保留表，所以无法执行 update 。



案例

|  |
| --- |
| -- todo 1 创建视图: 查询 地址id, 地址名称, 区域名称 -- todo 2 查询视图内容 -- todo 3 修改 地址名称 -- todo 4 修改 区域名称 -- todo 5 创建视图2: 在地址表 查询每个区域的地址数量 -- todo 6 修改 视图的区域值 -- todo 7 修改 视图的数量值 |

实现

|  |
| --- |
| -- todo 1 创建视图: 查询 地址id, 地址名称, 区域名称 create or replace view view\_addr\_area as select  t1.id,  t1.name ad\_name,  t2.name ar\_name from T\_ADDRESS t1, T\_AREA t2 where t1.AREAID=t2.ID ;  -- todo 2 查询视图内容 select \* from view\_addr\_area;  -- todo 3 修改 地址名称 update view\_addr\_area set ad\_name='流行花园' where id=1;  -- todo 4 修改 区域名称 update view\_addr\_area set ar\_name='海淀区' where id=1;  -- todo 5 创建视图2: 在地址表 查询每个区域的地址数量 create or replace view view\_addr\_2 as select  t1.areaid,  (select name from t\_area t2 where t1.AREAID=t2.ID) area\_name,  count(1) as cnt from T\_ADDRESS t1 group by t1.areaid ; select \* from view\_addr\_2; -- todo 6 修改 视图的区域值 update view\_addr\_2 set area\_name='海淀' where areaid=1;  -- todo 7 修改 视图的数量值 update view\_addr\_2 set cnt=666 where areaid=1; |

# 物化视图

## 什么是物化视图

视图是一个虚拟表（也可以认为是一条语句），基于它创建时指定的查询语句返回的结果集。每次访问它都会导致这个查询语句被执行一次。为了避免每次访问都执行这个查询，可以将这个查询结果集存储到一个物化视图（也叫实体化视图）。

物化视图与普通的视图相比的区别是物化视图是建立的副本，它类似于一张表，需要占用存储空间。

而对一个物化视图查询的执行效率与查询一个表是一样的。

## 创建物化视图语法

|  |
| --- |
| create materialized view view\_name  [build immediate | build deferred ]  refresh [fast|complete|force]  [  on [commit | demand ] | start with (start\_time) next  (next\_time)  ]  as  subquery |

* BUILD IMMEDIATE 是在创建物化视图的时候就生成数据
* BUILD DEFERRED 则在创建时不生成数据，以后根据需要再生成数据。
  + 默认为 BUILD IMMEDIATE。
* 刷新（ REFRESH ）：指当基表发生了 DML 操作后，物化视图何时采用哪种方式和基表进行同步。
* REFRESH 后跟着指定的刷新方法有三种：FAST、COMPLETE、FORCE。
  + FAST 刷新采用增量刷新，只刷新自上次刷新以后进行的修改。
  + COMPLETE 刷新对整 个物化视图进行完全的刷新。
  + 如果选择 FORCE 方式，则 Oracle 在刷新时会去判断是否可以进行快速刷新，
    - 如果可以则采用 FAST 方式，否则采用 COMPLETE的方式
    - FORCE 是默认的方式。
  + 刷新的模式有两种：ON DEMAND 和 ON COMMIT 。
    - ON DEMAND 指需要 手动刷新物化视图（默认）。
    - ON COMMIT 指在基表发生 COMMIT 操作时自动刷新。

## 案例

### 创建手动刷新的物化视图

需求：查询地址 ID,地址名称和所属区域名称, 结果如下：



语句：

|  |
| --- |
| create materialized view mv\_address  as  select ad.id, ad.name adname, ar.name ar\_name  from t\_address ad,t\_area ar  where ad.areaid=ar.id |

执行上边的语句后查询

|  |
| --- |
| select \* from mv\_address |

查询结果如下：



这时，我们向地址表（ T\_ADDRESS）中插入一条新记录，

|  |
| --- |
| insert into t\_address values(8,'宏福苑小区',1,1); |

再次执行上边的语句进行查询，会发现新插入的语句并没有出现在物化视图中。

我们需要通过下面的语句（PL/SQL），手动刷新物化视图：

|  |
| --- |
| begin  dbms\_mview.refresh('mv\_address','C');  end; |

### 创建自动刷新的物化视图，和上例一样的结果集

需求: 当 T\_ADDRESS 表发生变化时，物化视图 自动跟着改变。

语句如下：

|  |
| --- |
| create materialized view mv\_address2  refresh  on commit  as  select ad.id,ad.name adname,ar.name ar\_name  from t\_address ad, t\_area ar  where ad.areaid=ar.id |

创建此物化视图后，当 T\_ADDRESS 表发生变化时，MV\_ADDRESS2 自动跟着改变。

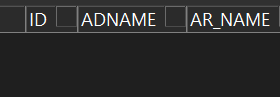
### 创建时不生成数据的物化视图

|  |
| --- |
| create materialized view mv\_address3  build deferred  refresh  on commit  as  select ad.id,ad.name adname,ar.name ar\_name  from t\_address ad,t\_area ar  where ad.areaid=ar.id; |

创建后执行下列语句查询物化视图

|  |
| --- |
| select \* from mv\_address3 |

查询结果：



执行下列语句生成数据

|  |
| --- |
| begin  dbms\_mview.refresh('MV\_ADDRESS3','C');  end; |

再次查询，得到结果：



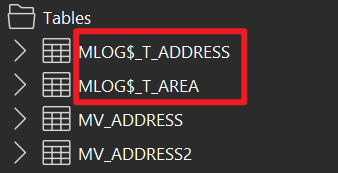
由于我们创建时指定的 on commit ,所以在修改数据后能立刻看到最新数据，无须再次执行 refresh

### 创建增量刷新的物化视图

如果创建增量刷新的物化视图，必须首先创建物化视图日志

|  |
| --- |
| create materialized view log on t\_address with rowid；  create materialized view log on t\_area with rowid; |

创建的物化视图日志名称为 MLOG$\_表名称



创建物化视图

|  |
| --- |
| create materialized view mv\_address4  refresh fast  as  select ad.rowid adrowid, ar.rowid arrowid, ad.id,ad.name adname,ar.name ar\_name  from t\_address ad, t\_area ar  where ad.areaid=ar.id; |

注意：创建增量刷新的物化视图，必须：

1. 创建物化视图中涉及表的物化视图日志。

2. 在查询语句中，必须包含所有表的 rowid ( 以 rowid 方式建立物化视图日志 )

当我们向地址表插入数据后，物化视图日志的内容：

|  |
| --- |
| insert into t\_address values(11, '天通苑北4区', 4, 5) |



SNAPTIME$$：用于表示刷新时间。

DMLTYPE$$：用于表示 DML 操作类型，I 表示 INSERT ，D 表示 DELETE ，U 表示 UPDATE。

OLD\_NEW$$：用于表示这个值是新值还是旧值。 N（EW）表示新值，O（ LD ） 表示旧值，U 表示 UPDATE 操作。

CHANGE\_VECTOR$$：表示修改矢量，用来表示被修改的是哪个或哪几个字段。

此列是 RAW 类型，其实 Oracle 采用的方式就是用每个 BIT 位去映射一个列。

插入操作显示为：FE, 删除显示为：OO, 更新操作则根据更新字段的位置而显示不同的值。

当我们手动刷新物化视图后，物化视图日志被清空，物化视图更新。

|  |
| --- |
| begin  DBMS\_MVIEW.refresh('MV\_ADDRESS4','C');  end; |

案例1

|  |
| --- |
| -- todo 1 增加物化视图日志 -- todo 2 创建增量物化视图: 统计 每个区域的地址数量 -- todo 3 添加一个新的地址 -- todo 4 查看日志 -- todo 5 刷新 begin  DBMS\_MVIEW.refresh('mv\_addr\_5','C'); end; -- todo 6 验证 |

实现

|  |
| --- |
| -- todo 1 增加物化视图日志 drop materialized view log on T\_ADDRESS; create materialized view log on T\_ADDRESS with rowid; create materialized view log on T\_AREA with rowid;  -- todo 2 创建增量物化视图: 地址id 地址名称 区域名称 create materialized view mv\_addr\_5  refresh fast as select  t1.ROWID as addr\_rowid,  t2.ROWID as area\_rowid,  t1.id,  t1.name addr\_name,  t2.name area\_name from T\_ADDRESS t1, T\_AREA t2 where t1.AREAID=t2.id ;  -- todo 3 添加一个新的地址 insert into T\_ADDRESS values(10, '天通苑4区', 2, 1); commit;  delete from T\_ADDRESS where id=8; delete from T\_ADDRESS where id=9; commit;  -- todo 4 查看日志 -- todo 5 刷新 begin  DBMS\_MVIEW.refresh('mv\_addr\_5','C'); end;  -- todo 6 验证 select \* from mv\_addr\_5; |

### 创建全量刷新的物化视图

案例

|  |
| --- |
| -- todo 1 创建全量物化视图: 统计每个区域的地址数量 -- todo 2 添加数据 测试  -- todo 3 验证 |

实现

|  |
| --- |
| -- todo 1 创建全量物化视图: 统计每个区域的地址数量 create materialized view mv\_addr\_7  refresh  on commit as select  areaid,  count(1) as cnt from t\_address group by areaid ;  -- todo 2 添加数据 测试 insert into T\_ADDRESS values(11, '天通苑5区', 2, 1); commit;   -- 3 验证 select \* from mv\_addr\_6; |

# 序列

## 什么是序列

序列是 ORACLE 提供的用于产生一系列唯一[数字的数据库对](http://lib.csdn.net/base/mysql)象。

## 创建与使用简单序列

创建序列语法：

|  |
| --- |
| create sequence 序列名称 |

通过序列的伪列来访问序列的值

**NEXTVAL** 返回序列的下一个值

**CURRVAL** 返回序列的当前值

注意：我们在刚建立序列后，无法提取当前值，只有先提取下一个值时才能再次提取当前值。

提取下一个值

|  |
| --- |
| select 序列名称.nextval from dual |

提取当前值

|  |
| --- |
| select 序列名称.currval from dual |

## 创建复杂序列

语法：

|  |
| --- |
| CREATE SEQUENCE sequence //创建序列名称  [INCREMENT BY n] //递增的序列值是 n 如果 n 是正数就递增,如果是负数就递减 默认是 1  [START WITH n] //开始的值,递增默认是 minvalue 递减是 maxvalue  [{MAXVALUE n | NOMAXVALUE}] //最大值  [{MINVALUE n | NOMINVALUE}] //最小值  [{CYCLE | NOCYCLE}] //循环/不循环  [{CACHE n | NOCACHE}];//分配并存入到内存中  但是你要记住cache的数量必须要小于你每一个周期所能得到的数据的数量 |

## 案例

### 有最大值的非循环序列

创建序列的语句：

|  |
| --- |
| create sequence seq\_test1  increment by 10  start with 10  maxvalue 300  minvalue 20 没有cycle不循环的话就会出现这种情况 |



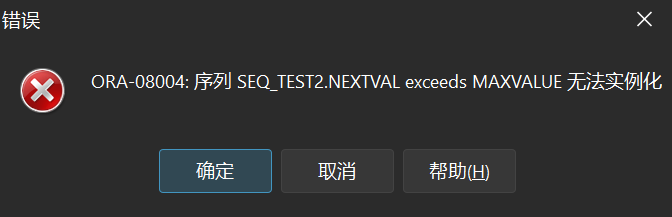
以上的错误，是由于我们的开始值小于最小值 。开始值不能小于最小值，修改

以上语句：

|  |
| --- |
| create sequence seq\_test1  increment by 10  start with 10  maxvalue 300  minvalue 5 没有cycle，不循环的话就会出现这种情况 |

我们执行下列语句提取序列值，当序列值为 300（最大值）的时候再次提取值，

系统会报异常信息。



需求

|  |
| --- |
| -- todo 需求1: 创建序列名为 seq\_a，它以5递增，从10开始，最大值为30，最小值为2。 |

实现

|  |
| --- |
| -- todo 创建序列名为 seq\_a，它以5递增，从10开始，最大值为30，最小值为2。 create sequence seq\_a increment by 5 start with 10 maxvalue 30 minvalue 2 ;  select seq\_a.nextval from dual; |

### 有最大值的循环序列

|  |
| --- |
| create sequence seq\_test2  increment by 10  start with 10  maxvalue 300  minvalue 5  cycle |

当序列当前值为 300（最大值），再次提取序列的值

|  |
| --- |
| select seq\_test2.nextval from dual;  select seq\_test2.currval from dual; |

提取的值为：

由此我们得出结论，循环的序列，第一次循环是从开始值开始循环，而第二次循环是从最小值开始循环。

思考问题：

下列语句是否会报错？为什么？

|  |
| --- |
| create sequence seq\_test3  increment by 10  start with 10  minvalue 5  cycle |

答：此为错误的语句。因为你创建的是一个循环的序列，所以必须指定最大值，否则会报错。

|  |
| --- |
| -- todo 需求2: 创建序列名为 seq\_b，它以5递增，从10开始，最大值为30，最小值为2, 带循环 |

实现

|  |
| --- |
| -- todo 需求2: 创建序列名为 seq\_b，它以5递增，从10开始，最大值为30，最小值为2, 带循环 create sequence seq\_b increment by 5 start with 10 maxvalue 30 minvalue 2 cycle cache 3 ;  select seq\_b.nextval from dual; |

### 带缓存的序列

我们执行下列语句：

|  |
| --- |
| create sequence seq\_test3  increment by 10  start with 10  maxvalue 300  minvalue 5  cycle  cache 50 |

我们执行上边语句的意思是每次取出 50 个缓存值，但是执行会提示错误



上边错误提示的意思是：缓存设置的数必须小于每次循环的数。

我们缓存设定的值是 50，而最大值是 300，那么为什么还会提示这样的信息呢？

其实我们的 cache 虽然是 50，但是我们每次增长值是 10。这样 50 次缓存提取出的数是 500 （50\*10）

我们更改为下列的语句：

|  |
| --- |
| create sequence seq\_test4  increment by 10  start with 10  maxvalue 500  minvalue 10  cycle  cache 50; |

下列语句依然会提示上边的错误，这是因为还存在一个 minvalue，minvalue 和 maxvalue 之间是 49 个数，也就是一次循环可以提取 490，但是我们的缓存是 500。

我们再次修改语句：

|  |
| --- |
| create sequence seq\_test5  increment by 10  start with 10  maxvalue 500  minvalue 9  cycle  cache 50 |

把最小值减 1，或把最大值加 1 ，都可以通过。

## 修改和删除序列

修改序列：使用 ALTER SEQUENCE 语句修改序列，不能更改序列的 START

WITH 参数

|  |
| --- |
| alter sequence 序列名称 maxvalue 5000 cycle; |

删除序列：

|  |
| --- |
| DROP SEQUENCE 序列名称; |

综合案例

|  |
| --- |
| -- 1 创建序列 seq\_student, 每次递增2, 从1000开始  -- 2 创建学生表 tb\_student(sid, sname)  -- 3 插入数据, sid 使用序列生成的值  -- 4 验证 |

实现

|  |
| --- |
| -- 1 创建序列 seq\_student, 每次递增2, 从1000开始, create sequence seq\_student2 increment by 2 start with 1000 ;  -- 2 创建学生表 tb\_student(sid, sname) create table tb\_student(  sid int primary key,  s\_name varchar2(32) );  -- 3 插入数据, sid 使用序列生成的值 insert into tb\_student values(seq\_student.nextval, '张三'); insert into tb\_student values(seq\_student.nextval, '李四'); insert into tb\_student values(seq\_student.nextval, '王五'); insert into tb\_student values(seq\_student.nextval, '赵六'); insert into tb\_student values(seq\_student.nextval, '刘备');  -- 4 验证 select \* from tb\_student; |

# 同义词

## 什么是同义词

同义词实质上是指定方案对象的一个别名。通过屏蔽对象的名称和所有者以 及对分布式数据库的远程对象提供位置透明性，同义词可以提供一定程度的安全性。同时，同义词的易用性较好，降低了数据库用户的 SQL 语句复杂度。

同义词允许基对象重命名或者移动，这时，只需对同义词进行重定义，基于同义词的应用程序可以继续运行而无需修改。

你可以创建公共同义词和私有同义词。其中，公共同义词属于 PUBLIC 特殊 用户组，数据库的所有用户都能访问；而私有同义词包含在特定用户的方案中，

只允许特定用户或者有基对象访问权限的用户进行访问。

同义词本身不涉及安全，当你赋予一个同义词对象权限时，你实质上是在给同义词的基对象赋予权限，同义词只是基对象的一个别名。

## 创建与使用同义词

创建同义词的具体语法是：

|  |
| --- |
| create [public] SYNONYM synooym for object; |

其中 synonym 表示要创建的同义词的名称，object 表示表，[视图](http://www.chinabyte.com/keyword/%E8%A7%86%E5%9B%BE/)，序列等我们要创建同义词的对象的名称。

## 案例

### 私有同义词

需求：为表 T\_OWNERS 创建( 私有 )同义词 名称为 OWNERS

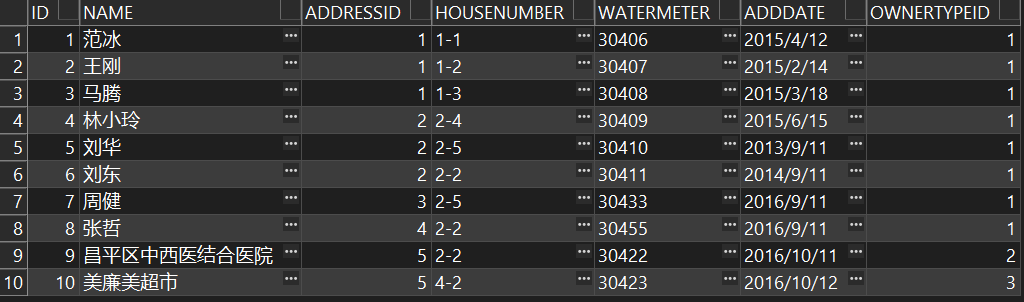
语句：

|  |
| --- |
| create synonym owners for t\_owners; |

使用同义词：

|  |
| --- |
| select \* from owners; |

查询结果如下：



### 公有同义词

需求：为表 T\_OWNERS 创建( 公有 )同义词 名称为 OWNERS2：

|  |
| --- |
| create public synonym owners2 for t\_owners; |

以另外的用户登陆，也可以使用公有同义词：

|  |
| --- |
| select \* from owners; |

案例

|  |
| --- |
| -- 1 需求：为表 T\_AREA 创建( 私有 )同义词 名称为 AREA  -- 2 需求：为表 T\_AREA 创建( 共有 )同义词 名称为 AREA2  -- 3 验证 其他用户 |

实现

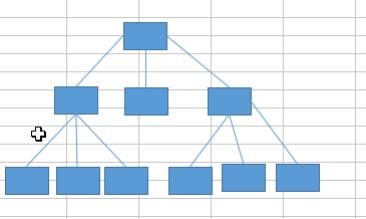
|  |
| --- |
| -- 1 需求：为表 T\_AREA 创建( 私有 )同义词 名称为 AREA create synonym area for t\_area; select \* from area;  -- 2 需求：为表 T\_AREA 创建( 共有 )同义词 名称为 AREA2 create public synonym area2 for t\_area;  -- 3 验证 select \* from area2; |

# 索引

## 什么是索引

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。

索引是需要占据存储空间的，也可以理解为是一种特殊的数据。形式类似于 下图的一棵“树”，而树的节点存储的就是每条记录的物理地址，也就是我们提到的伪列（ROWID）



## 普通索引

语法：

|  |
| --- |
| create index 索引名称 on 表名(列名); |

需求：我们经常要根据业主名称搜索业主信息，所以我们基于业主表的 name 字

段来建立索引。语句如下：

|  |
| --- |
| create index index\_owners\_name on t\_owners(name) |

索引性能测试：

创建一个两个字段的表

|  |
| --- |
| create table t\_indextest (  id number,  name varchar2(30)  ); |

编写 PL/SQL 插入 100 万条记录（关于 PL/SQL 我们在第四章会学到）

|  |
| --- |
| begin  for i in 1 .. 1000000 loop  insert into t\_indextest values (i, 'dev' || i);  end loop;  commit;  end; |

创建完数据后，根据 name 列创建索引

|  |
| --- |
| create index index\_testindex on t\_indextest(name); |

执行下面两句 SQL 执行

|  |
| --- |
| SELECT \* from T\_INDEXTEST where ID=765432;  SELECT \* from T\_INDEXTEST where NAME= 'dev765432'; |

我们会发现根据 name 查询所用的时间会比根据 id 查询所用的时间要短

## 唯一索引

如果我们需要在某个表某个列创建索引，而这列的值是不会重复的。这是我们可以创建唯一索引。

语法：

|  |
| --- |
| create unique index 索引名称 on 表名(列名); |

需求：在业主表的水表编号一列创建唯一索引

语句：

|  |
| --- |
| create unique index index\_owners\_watermeter on t\_owners(watermeter); |

## 复合索引

我们经常要对某几列进行查询，比如，我们经常要根据学历和性别对学员进行搜 索，如果我们对这两列建立两个索引，因为要查两棵树，查询性能不一定高。那 如何建立索引呢？我们可以建立复合索引，也就是基于两个以上的列建立一个索引 。

语法：

|  |
| --- |
| create index 索引名称 on 表名(列名,列名.....); |

根据地址和门牌号对学员表创建索引，语句如下：

|  |
| --- |
| create index owners\_index\_ah on t\_owners(addressid, housenumber); |

# 权限 角色 用户 表空间

这一部分我们主要看看oracle种如何管理权限和角色，权限和角色的区别在那里。

当刚刚建立用户时，用户没有任何权限，也不能执行任何操作。如果要执行某种特定的数据库操作，则必须为其授予系统的权限;如果用户要访问其它方案的对象，则必须为其授予对象的权限.为了简化权限的管理，可以使用角色。这里我们会详细的介绍

## 用户

### 创建用户

|  |
| --- |
| create user c##zhangsan identified by itcast; --12C以后  -------------------------------------------------  create user 用户名 identified by 密码 --密码  default tablespace 表空间 --指定表空间  profile default |

### 修改用户密码

|  |
| --- |
| alter user 用户 identified by 新密码; |

### 删除用户

|  |
| --- |
| drop user test1; -- 普通账号  drop user test1 cascade; -- 带数据权限的 |

## 权限

### 系统权限介绍

系统权限是指执行特定类型sql命令的权利.它用于控制用户可以执行的一个或是一组数据库操作.比如当用户具有create table权限时，可以在其方案中建表,当用户具有create any table权限时,可以在任何方案中建表.

常用的有:

|  |  |
| --- | --- |
| 功能 | 权限 |
| 连接数据库 | create session |
| 建表 | create table |
| 建视图 | create view |
| 建同义词 | create public synonym |
| 建过程、函数、包 | create procedure |
| 建触发器 | create trigger |

### 显示系统权限

oracel 21c 提供了 261 系统权限，而且oracle的版本越高，提供的系统权限就越多,我们可以查询数据字典视图system\_privilege\_map，可以显示所有系统权限

|  |
| --- |
| select \* from system\_privilege\_map order by name; |

### 授予系统权限

语法:

|  |
| --- |
| grant 杈限名称 to 用户名 [with admin option] |

一般情况，授予系统权限是有DBA完成的，如果用其它用户来授予系统权限，则要求该用户必须具有grant any privilege的系统权限。

在授予系统权限时，可以带有with admin option选项，这样，被授子权限的用户或是角色还可以将该系统权限授予其它的用户或是角色。

需求:

|  |
| --- |
| 1.创建两个用户 caocao, liubei. 初始阶段他们没有任何权限，如果登陆就会给出错误的信息  1.1创建两个用户，并指定密码  2.给用户caocao授权:  2.1:授子create session 和 create table权限时 带with admin option  2.2 授予create view 时不带with admin option  3.给用户liubei授权  我们可以通过caocao 给liubei授权，因为with admino ption是加上的。  当然也可以通过dba给liubei授权,我们就用caocao给liubei授权。 |

实现

|  |
| --- |
| -- todo 需求:  -- todo 1.创建两个用户 caocao,liubei. 初始阶段他们没有任何权限，如果登陆就会给出错误的信息  -- todo 1.1创建两个用户，并指定密码  create user c##caocao identified by itcast;  create user c##liubei identified by itcast;  -- todo 2.给用户caocao授权:  -- todo 2.1:授子create session 和 create table权限时 带 with admin option  grant create session to c##caocao with admin option;  grant create table to c##caocao with admin option;  -- todo 2.2 授予create view 时不带with admin option  grant create view to c##caocao;  -- todo 3.给用户 liubei授权  -- todo 我们可以通过caocao 给liubei授权，因为with admin option是加上的。  -- todo 当然也可以通过dba给liubei授权,我们就用caocao给liubei授权。 **可以吗?**  grant create session,create table to c##liubei;  grant create view to liubei; |

### 回收系统权限

语法

|  |
| --- |
| revoke 权限名称 from 用户名 |

一般情况下，回收系统权限是dba来完成的，如果其它的用户来回收系统权限，要求该用户必须具有相应系统权限及转授系统权限的选项(with admin option)。回收系统权限使用revoke来完成,当回收了系统权限用户就不能执行相应的操作了，但是请注意，系统权限级联收回问题? **不会级联回收**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| system | ------------> | caocao | ------------> | liubei |
| create session |  | create session |  | create session |

用system执行如下操作:

|  |
| --- |
| revoke create session from c##caocao; |

**请思考** liubei还能登录? 可以，系统权限并不会被级联

### 对象权限介绍

指访问其它方案对象的权利,用户可以直接访问自己方案的对象，但是如果要访问别的方案的对象，则必须具有对象的权限.比如smith用户要访问wateruser.t\_owners表(scott:方案,emp:表)则必须在wateruser.t\_owners表上具有对象的权限。

常用的有

|  |  |  |  |
| --- | --- | --- | --- |
| alter 修改 | delete 删除 | select查询 | insert添加 |
| update 修改 | index 索引 | references引用 | execute执行 |

查看oracle提供的所有的对象权限(dba用户可以查看):

|  |
| --- |
| select distinct privilege from dba\_tab\_privs; |

### 授予对象权限

在oracle9i前，授予对象权限是由对象的所有者来完成的，如果用其它的用户来操作，则需要用户具有相应的(with grant option )权限.

从oracle9i开始,sys,system 可以将任何对象上的对象权限授予其它用户.授予对象权限是用grant命令来完成的.

语法:

|  |
| --- |
| grant 对象权限 on 数据库对象  to 用户名[,角色名] [with grant option] |

案例:

|  |
| --- |
| 1. zhaoyun用户要操作wateruser.t\_owners表,则必须授子相应的对象权限   1.1 希望zhaoyun可以查询wateruser.t\_owners的表数据，怎样操作?  1.2 希望zhaoyun可以修改wateruser.t\_owners的表数据、怎样操作?  1.3 希望zhaoyun可以删除wateruser.t\_owners的表数据，怎样操作?  1.4 有没有更加简单的方法,一次把所有权限赋给zhaoyun? |

实现:

|  |
| --- |
| create user c##zhaoyun identified by itcast; grant create session to c##zhaoyun; -- 1.zhaoyun用户要操作wateruser.t\_owners表,则必须授子相应的对象权限 -- 1.1 希望zhaoyun可以查询wateruser.t\_owners的表数据，怎样操作? grant select on C##WATERUSER.t\_owners to c##zhaoyun with grant option; -- 1.2 希望zhaoyun可以修改wateruser.t\_owners的表数据、怎样操作? grant update on C##WATERUSER.t\_owners to c##zhaoyun with grant option; -- 1.3 希望zhaoyun可以删除wateruser.t\_owners的表数据，怎样操作? grant delete on C##WATERUSER.t\_owners to c##zhaoyun with grant option; -- 1.4 有没有更加简单的方法,一次把所有权限赋给zhaoyun? grant all on C##WATERUSER.t\_owners to c##zhaoyun with grant option; |

2.授子alter权限

如果zhaoyun用户要修改wateruser.t\_owners表的结构,则必须授予alter对象权限

|  |
| --- |
| grant alter on wateruser.t\_owners to c##zhaoyun; |

3.授子execute权限

如果用户想要执行其它方案的包/过程/函数,则须有execute权限.比如为了让caocao可以执行包dbms\_transaction,可以授execute权限

4.授子index权限

如果想在别的方案的表上建立索引,则必须具有index对象权限,如为了让black可以在 scett.emp上建立索引，就给其index的对象权限

|  |
| --- |
| grant index on wateruser.t\_owners to yangjian |

5.使用 with grant option 选项

该选项用于转授对象权限.但是该选项只能被授子用户,而不能授子角色

|  |
| --- |
| grant select on emp to yangjian with grant option |

### 回收对象权限

语法

|  |
| --- |
| revoke 对象权限 on 数据库对象 from 用户名[,角色名] |

收回对象的权限可以由对象的所有者来完成，也可以用dba用户(sys,system)来完成

这里要说明的时:收回对象权限后，用户就不能执行相应的sql命令, 但是要注意的是对象的权限是否会被级联收回?

文件对象是会被联级收回的

**案例:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| wateruser | ---------> | yangjian | ---------> | nazha |
| select on t\_owners |  | select on t\_owners |  | select on t\_owners |

|  |
| --- |
| -- todo 1 system操作 创建用户 yangjian 和 nazha  -- todo 2 system操作: 给用户 yangjian 和 nazha 授权 create session  -- todo 3 wateruser 登录，完成如下操作: 把对 t\_owners 表的操作权利给 yangjian:  -- todo 4 使用 yangjian 把权限给 nazha;  -- todo 5 使用 wateruser 将查询权限从yangjian 回收  -- todo 6 验证: yangjian 和 nazha 还能对 wateruser.t\_owners 进行查询吗? |

实现

|  |
| --- |
| -- todo 1 system操作 创建用户 yangjian 和 nazha  create user c##yangjian identified by itcast;  create user c##nazha identified by itcast;  -- todo 2 system操作: 给用户 yangjian 和 nazha 授权 create session  grant create session to c##yangjian;  grant create session to c##nazha;  -- todo 3 wateruser 登录，完成如下操作: 把对 t\_owners 表的操作权利给 yangjian:  grant select on t\_owners to c##yangjian with grant option;  -- todo 4 使用 yangjian 把权限给 nazha;  grant select on c##wateruser.t\_owners to c##nazha;  -- todo 5 使用 wateruser 将查询权限从yangjian 回收  revoke select on t\_owners from c##yangjian;  -- todo 6 验证: yangjian 和 nazha 还能对 wateruser.t\_owners 进行查询吗?  select \* from c##wateruser.T\_OWNERS;  -- todo 此时yangjian和nazha都不能再对 wateruser.t\_owners 进行查询, 说明 对象权限是级联回收的. |

## 角色管理

### 介绍

角色就是相关权限的命令集合,使用角色的主要目的就是为了简化权限的管理.

请看一个问题:假定有用户1,2,3 为了让他们都拥有权限

* 连接数据库 create session
* 在wateruser.t\_owners表上select,insert,update... ... 如果采用直接授权操作，则需要进行12次授权。

因为要进行12次授权操作, 所以比较麻烦. 怎么办?

### 预定义角色

预定义角色是指oracle所提供的角色,每种角色都用于执行一些特定的管理任务,下面我们介绍常用的预定义角色connect, resource, dba.

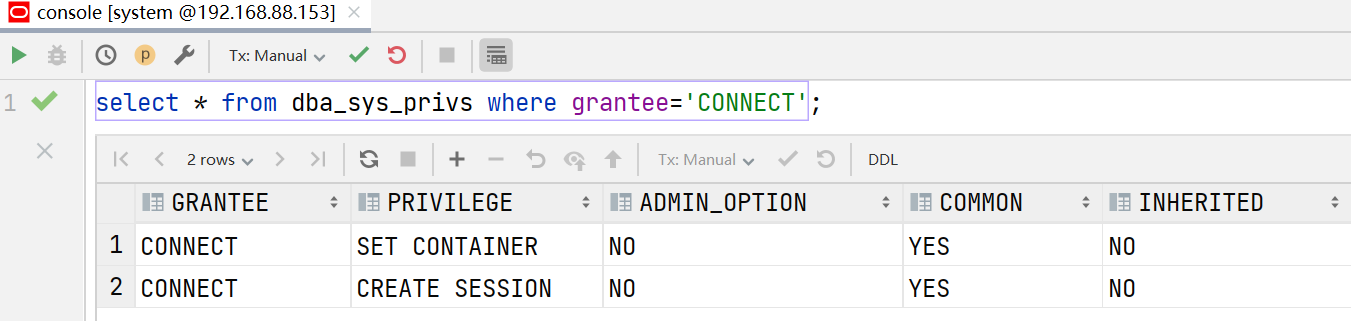
一般应用开发人员需要的大部分权限,只要给用户授子connect和resource角色就够了.

#### connect角色

connect角色具有哪些系统权限呢?

create session

|  |
| --- |
| 了解 查询connect角色具有哪些系统权限呢? 注意必须大写  select \* from dba\_sys\_privs where grantee='CONNECT'; |



#### resource角色

resource角色具有应用开发人员所需要的其它权限, 比如建立存储过程触发器等。

这里需要注意的是resource角色隐含了unlimited tablespace系统权限。

**resource角色包含以下系统权限:**

create cluster

create indextype

create table

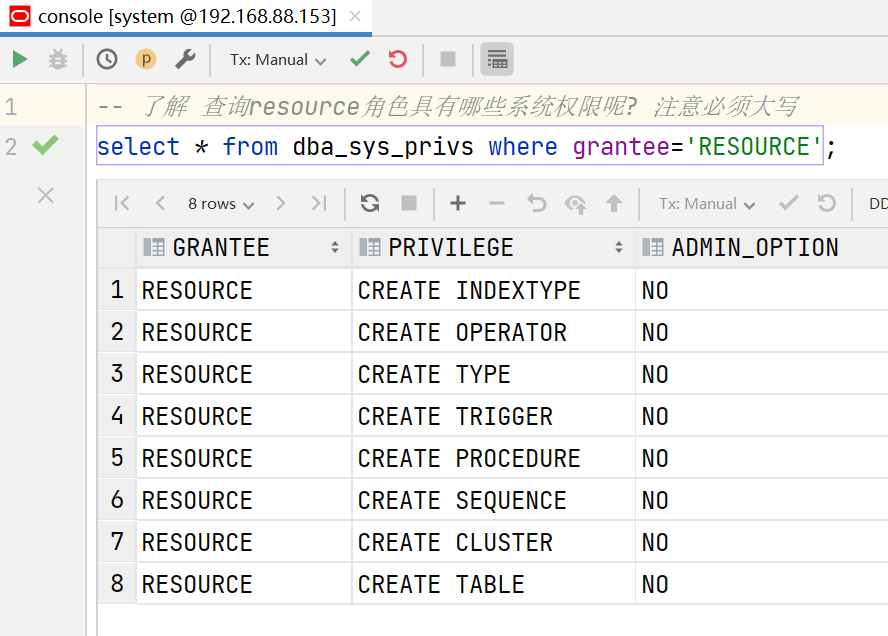
create sequence

create type

create procedure

create trigger

|  |
| --- |
| -- 了解 查询resource角色具有哪些系统权限呢? 注意必须大写 select \* from dba\_sys\_privs where grantee='RESOURCE'; |

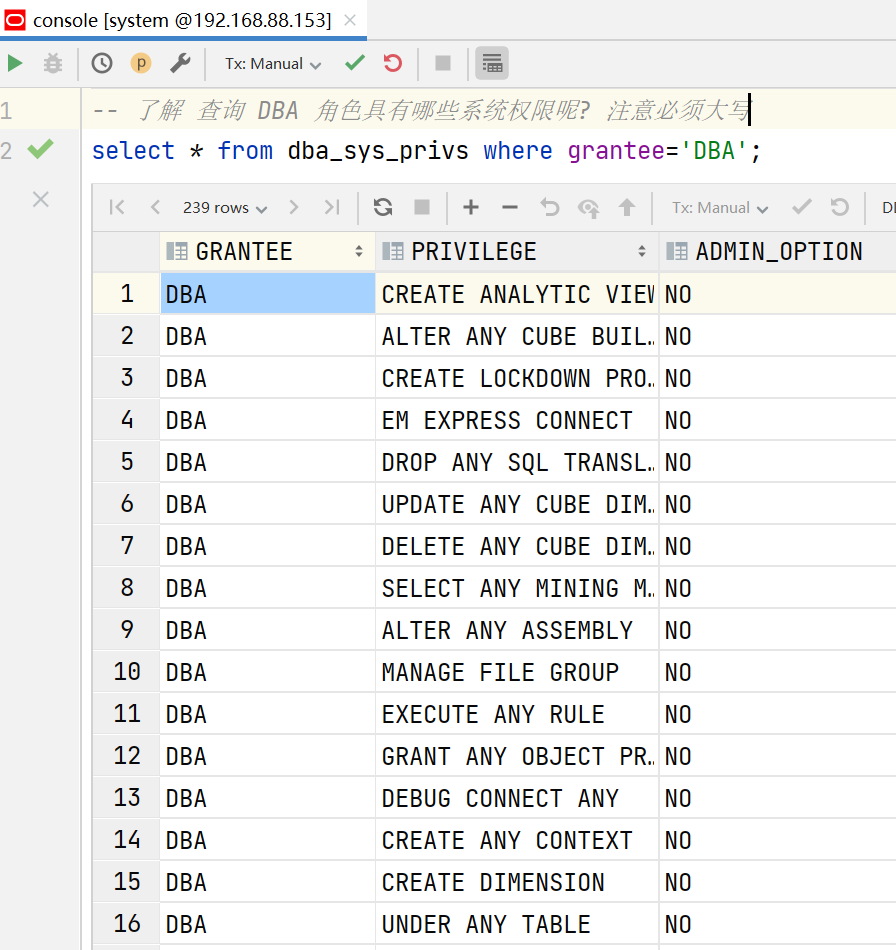


#### dba角色

dba角色具有所有的系统权限，及 with admin option选项, 默认的dba用户为sys和system

他们可以将任何系统权限授予其它用户但是要注意的是dba角色不具备(启动和关闭数据库)

|  |
| --- |
| -- 了解 查询 DBA 角色具有哪些系统权限呢? 注意必须大写  select \* from dba\_sys\_privs where grantee='DBA'; |



语法

|  |
| --- |
| grant 角色名 to 用户名 [with admin option]; |

案例

|  |
| --- |
| 1 创建一个用户 jack  2 将 jack 设为具有 DBA 角色的用户 |

实现

|  |
| --- |
| -- todo 1 创建一个用户 jack create user c##jack identified by itcast;  -- todo 2 将 jack 设为具有 DBA 角色的用户 grant DBA to c##jack with admin option; |

扩展: 如何知道某个用户具有什么样的角色, 注意查询时 角色名称必须大写

|  |
| --- |
| select \* from dba\_role\_privs where grantee='C##JACK'; |

## 自定义角色

oracle 设计者，认为，33 中预定义角色可能不能满足所有的需求，所以，可以使用自定义角色来解决.

顾名思义就是自己定义的角色根据自己的需要来定义.一般是dba来建立.

### 创建角色

如果用别的用户来建立，则需要具有create role的系统权限.在建立角色时可以指定验证方式(不验证，数据库验证等)

建立角色(不验证)

方式一 如果角色是公用的角色，可以采用不验证的方式建立角色

|  |
| --- |
| create role 角色名 not identified: |

方式二 建立角色(数据库验证)

采用这样的方式时,角色名、口令存放在数据库中。当激活该角色时，必须提供口令.在建立这种角色时，需要为其提供口令

|  |
| --- |
| create role 角色名 identified by xxx |

### 给角色授权

语法

|  |
| --- |
| grant 系统权限 to 角色;  grant 对象权限 on 对象 to 角色; |

例子

|  |
| --- |
| grant create session to c##dev;  grant select, insert, update, delete on C##WATERUSER.T\_OWNERS to c##dev; |

### 分配角色给某个用户

一般分配角色是由dba(即sys或system)来完成的，如果要以其它用户身份分配角色,则要求用户必须具有grant any role的系统权限。

|  |
| --- |
| grant 角色名 to 用户名 [with admin option] |

### 删除角色

使用drop role,一般是dba来执行,如用其它用户则要求该用户具有drop anyrole 系统权限

|  |
| --- |
| drop role 角色名 |

### 显示角色信息

|  |
| --- |
| 1.显示所有角色 select \* from dba\_roles;  2.显示用户具有的角色，及默认角 select granted\_role from dba\_role\_privs where grantee='C##JACK'; 注意:用户名大写 |

案例

|  |
| --- |
| -- todo 1.创建用户 sunquan, 并把某个角色 dev (具有create session,select on wateruser.t\_owners杈限)赋给该用户. -- todo 2.创建用户 zhouyu -- todo 3.用sunquan给zhouyu赋角色dev -- todo 4.测试 -- todo 5 删除角色 -- todo 6.测试 |

实现

|  |
| --- |
| -- todo 1.创建用户 sunquan, 并把某个角色dev(具有create session,select on wateruser.t\_owners杈限)赋给该用户. create user c##sunquan identified by itcast;  create role c##dev not identified;  grant create session to c##dev; grant select, insert, update, delete on C##WATERUSER.T\_OWNERS to c##dev;  grant c##dev to c##sunquan with admin option;  -- todo 2.创建用户 zhouyu create user c##zhouyu identified by itcast;  -- todo 3.用sunquan给zhouyu赋角色dev grant c##dev to c##zhouyu;  -- todo 4.测试 select \* from C##WATERUSER.T\_OWNERS;  -- todo 5 删除角色 drop role c##dev; |

# 分区表

## 为什么

分区表通过将大表分割成更小的逻辑单元，提高了数据库的性能、管理效率和数据存储的灵活性。

* 优点：
  + 增强可用性：如果表的某个分区出现故障，表在其他分区的数据仍然可用；
  + 维护方便：如果表的某个分区出现故障，需要修复数据，只修复该分区即可；
  + 均衡 I/O：可以把不同的分区映射到磁盘以平衡 I/O，改善整个系统性能；
  + 改善查询性能：对分区对象的查询可以**仅搜索**自己关心的分区，**提高检索速**

## 分类

* 范围分区
* HASH 分区（散列分区）
* 列表分区
* 复合分区（范围＋HASH）

## 准备工作

|  |
| --- |
| CREATE TABLESPACE part1\_ts DATAFILE 'C:\data\part1\_ts\_01.dbf' SIZE 10M,  'C:\data\part1\_ts\_02.dbf' SIZE 10M,  'C:\data\part1\_ts\_03.dbf' SIZE 10M;  CREATE TABLESPACE part2\_ts DATAFILE 'C:\data\part2\_ts\_01.dbf' SIZE 10M,  'C:\data\part2\_ts\_02.dbf' SIZE 10M,  'C:\data\part2\_ts\_03.dbf' SIZE 10M;  CREATE TABLESPACE part3\_ts DATAFILE 'C:\data\part3\_ts\_01.dbf' SIZE 10M,  'C:\data\part3\_ts\_02.dbf' SIZE 10M,  'C:\data\part3\_ts\_03.dbf' SIZE 10M; |

## 范围分区

是对数据表中的某个值的范围进行分区，根据某个值的范围，决定将该数据存储在哪个分区上。

可根据序号分区，根据业务记录的创建日期进行分区等。

如：

订单表，表名：tb\_order,该表可能有千万条数据记录，

要求使用分区表

* 按序号分区

|  |
| --- |
| CREATE TABLE tb\_order (  order\_id NUMBER PRIMARY KEY,  customer\_id NUMBER,  order\_date DATE,  total\_amount NUMBER(10, 2),  status VARCHAR2(20) ) PARTITION BY RANGE(order\_id) (  PARTITION orders\_1 VALUES LESS THAN (1000) TABLESPACE part1\_ts,  PARTITION orders\_2 VALUES LESS THAN (2000) TABLESPACE part2\_ts,  PARTITION orders\_3 VALUES LESS THAN (MAXVALUE) TABLESPACE part3\_ts );  -- 插入数据到orders\_1分区 INSERT INTO tb\_order PARTITION (orders\_1) VALUES (1, 1001, TO\_DATE('2023-01-01', 'YYYY-MM-DD'), 100.50, 'Pending');  -- 插入数据到orders\_2分区 INSERT INTO tb\_order PARTITION (orders\_2) VALUES (1001, 2001, TO\_DATE('2023-02-01', 'YYYY-MM-DD'), 200.75, 'Processing');  -- 插入数据到orders\_3分区 INSERT INTO tb\_order PARTITION (orders\_3) VALUES (2001, 3001, TO\_DATE('2023-03-01', 'YYYY-MM-DD'), 300.25, 'Completed');  -- 从orders\_1分区取数据 SELECT \* FROM tb\_order PARTITION (orders\_1);  -- 从orders\_2分区取数据 SELECT \* FROM tb\_order PARTITION (orders\_2);  -- 从orders\_3分区取数据 SELECT \* FROM tb\_order PARTITION (orders\_3);  -- 查不到, 为什么? SELECT \* FROM tb\_order PARTITION (orders\_2)  where order\_id=3001; |

* 按日期范围

|  |
| --- |
| CREATE TABLE tb\_order\_2 (  order\_id NUMBER PRIMARY KEY,  customer\_id NUMBER,  order\_date DATE,  total\_amount NUMBER(10, 2),  status VARCHAR2(20) ) PARTITION BY RANGE(order\_date) (  PARTITION orders\_2022 VALUES LESS THAN (TO\_DATE('2023-01-01', 'YYYY-MM-DD')) TABLESPACE part1\_ts,  PARTITION orders\_2023 VALUES LESS THAN (TO\_DATE('2024-01-01', 'YYYY-MM-DD')) TABLESPACE part2\_ts,  PARTITION orders\_future VALUES LESS THAN (MAXVALUE) TABLESPACE part3\_ts );  -- 插入数据到orders\_2022分区 INSERT INTO tb\_order\_2 PARTITION (orders\_2022) VALUES (1, 1001, TO\_DATE('2022-12-31', 'YYYY-MM-DD'), 100.50, 'Pending');  -- 插入数据到orders\_2023分区 INSERT INTO tb\_order\_2 PARTITION (orders\_2023) VALUES (1001, 2001, TO\_DATE('2023-12-31', 'YYYY-MM-DD'), 200.75, 'Processing');  -- 插入数据到orders\_future分区 INSERT INTO tb\_order\_2 PARTITION (orders\_future) VALUES (2001, 3001, TO\_DATE('2024-12-31', 'YYYY-MM-DD'), 300.25, 'Completed');  -- 从orders\_2022分区取数据 SELECT \* FROM tb\_order\_2 PARTITION (orders\_2022);  -- 从orders\_2023分区取数据 SELECT \* FROM tb\_order\_2 PARTITION (orders\_2023);  -- 从orders\_future分区取数据 SELECT \* FROM tb\_order\_2 PARTITION (orders\_future); |

## HASH 分区（散列分区）

在列的取值不容易确定时可以采用此方法。

HASH 实际上是一种算法，当向表插入数据时，系统会**自动**根据当前分区

列的值**计算出 HASH** 值之后确定应该将该行存放在哪个表空间。

|  |
| --- |
| -- 创建表 tb\_order\_3，并使用哈希分区 CREATE TABLE tb\_order\_3 (  order\_id NUMBER PRIMARY KEY,  customer\_id NUMBER,  order\_date DATE,  total\_amount NUMBER(10, 2),  status VARCHAR2(20) ) PARTITION BY HASH(order\_id) (  PARTITION part1 TABLESPACE part1\_ts,  PARTITION part2 TABLESPACE part2\_ts,  PARTITION part3 TABLESPACE part3\_ts );  -- 向 tb\_order\_3 表插入数据 INSERT INTO tb\_order\_3 (order\_id, customer\_id, order\_date, total\_amount, status) VALUES (1, 1001, TO\_DATE('2022-01-01', 'YYYY-MM-DD'), 100.50, 'Pending');  INSERT INTO tb\_order\_3 (order\_id, customer\_id, order\_date, total\_amount, status) VALUES (2, 2001, TO\_DATE('2022-01-01', 'YYYY-MM-DD'), 200.75, 'Processing');  INSERT INTO tb\_order\_3 (order\_id, customer\_id, order\_date, total\_amount, status) VALUES (3, 3001, TO\_DATE('2022-01-01', 'YYYY-MM-DD'), 300.25, 'Completed');  -- 验证数据插入 SELECT \* FROM tb\_order\_3; |

## 列表分区

对表的某个列的可列举的值进行分区

如果分区的字段的值并不能划分范围，同时分区的值的取值有一个范围，则

在分区条件中可以只用**枚举的方式**列出分区字段的所有选项，从而达到分区的目的

|  |
| --- |
| drop table tb\_order\_4;  -- 创建列表分区 tb\_order\_4，并添加城市列作为分区键 CREATE TABLE tb\_order\_4 (  order\_id NUMBER PRIMARY KEY,  customer\_id NUMBER,  order\_date DATE,  total\_amount NUMBER(10, 2),  status VARCHAR2(20),  city VARCHAR2(50) -- 添加城市列 ) PARTITION BY LIST (city) (  PARTITION part\_beijing VALUES ('北京') TABLESPACE part1\_ts,  PARTITION part\_shanghai VALUES ('上海') TABLESPACE part2\_ts,  PARTITION part\_guangdong VALUES ('广州', '深圳') TABLESPACE part3\_ts,  PARTITION qita VALUES (default) TABLESPACE part3\_ts ); -- 向 tb\_order\_4 表插入数据 INSERT INTO tb\_order\_4 (order\_id, customer\_id, order\_date, total\_amount, status, city) VALUES (1, 1001, TO\_DATE('2023-01-01', 'YYYY-MM-DD'), 100.50, 'Pending', '北京');  INSERT INTO tb\_order\_4 (order\_id, customer\_id, order\_date, total\_amount, status, city) VALUES (2, 2001, TO\_DATE('2023-02-01', 'YYYY-MM-DD'), 200.75, 'Processing', '上海');  INSERT INTO tb\_order\_4 (order\_id, customer\_id, order\_date, total\_amount, status, city) VALUES (3, 3001, TO\_DATE('2023-03-01', 'YYYY-MM-DD'), 300.25, 'Completed', '广州');  ALTER TABLESPACE PART3\_TS ADD DATAFILE 'C:\data\part3\_ts\_04.dbf' SIZE 100M;  INSERT INTO tb\_order\_4 (order\_id, customer\_id, order\_date, total\_amount, status, city) VALUES (4, 4001, TO\_DATE('2024-03-01', 'YYYY-MM-DD'), 300.25, 'Completed', '石家庄');  -- 验证数据插入 SELECT \* FROM tb\_order\_4;  SELECT \* FROM tb\_order\_4 PARTITION (part\_beijing);  -- 从广州分区取数据 SELECT \* FROM tb\_order\_4 PARTITION (part\_guangdong);  -- 从非一线城市取数据 SELECT \* FROM tb\_order\_4 PARTITION (qita); |

## 复合分区（范围＋HASH）（范围＋列表）

|  |
| --- |
| -- 给表空间增加文件 ALTER TABLESPACE part1\_ts ADD DATAFILE 'C:\data\part1\_ts\_05.dbf' SIZE 10M; ALTER TABLESPACE part2\_ts ADD DATAFILE 'C:\data\part2\_ts\_05.dbf' SIZE 10M; ALTER TABLESPACE part3\_ts ADD DATAFILE 'C:\data\part3\_ts\_05.dbf' SIZE 10M;   drop table tb\_order\_5;  -- 创建表 tb\_order\_5，并使用复合分区（范围+哈希）策略 CREATE TABLE tb\_order\_5 (  order\_id NUMBER PRIMARY KEY,  customer\_id NUMBER,  order\_date DATE,  total\_amount NUMBER(10, 2),  status VARCHAR2(20),  city VARCHAR2(50) ) PARTITION BY RANGE (order\_date) SUBPARTITION BY HASH (order\_id) SUBPARTITIONS 3 (  PARTITION PART\_BEFORE\_2023 VALUES LESS THAN (TO\_DATE('2023-01-01', 'YYYY-MM-DD'))  (  SUBPARTITION part1 TABLESPACE part1\_ts,  SUBPARTITION part2 TABLESPACE part2\_ts,  SUBPARTITION part3 TABLESPACE part3\_ts  ),  PARTITION PART\_2023 VALUES LESS THAN (TO\_DATE('2024-01-01', 'YYYY-MM-DD'))  (  SUBPARTITION part4 TABLESPACE part1\_ts,  SUBPARTITION part5 TABLESPACE part2\_ts,  SUBPARTITION part6 TABLESPACE part3\_ts  ),  PARTITION PART\_AFTER\_2024 VALUES LESS THAN (MAXVALUE)  (  SUBPARTITION part7 TABLESPACE part1\_ts,  SUBPARTITION part8 TABLESPACE part2\_ts,  SUBPARTITION part9 TABLESPACE part3\_ts  ) );  INSERT INTO tb\_order\_5 (order\_id, customer\_id, order\_date, total\_amount, status, city) VALUES (1, 1001, TO\_DATE('2022-12-31', 'YYYY-MM-DD'), 100.50, 'Pending', '北京');  INSERT INTO tb\_order\_5 (order\_id, customer\_id, order\_date, total\_amount, status, city) VALUES (2, 2001, TO\_DATE('2023-01-01', 'YYYY-MM-DD'), 200.75, 'Processing', '上海');  -- 检索 PART\_BEFORE\_2023 分区中的子分区 part1 中的数据 SELECT \* FROM tb\_order\_5 PARTITION (PART\_BEFORE\_2023); |

## 分区操作

删除失败:

|  |
| --- |
| delete from tb\_order PARTITION (orders\_2) t where t.order\_id=1; commit; select \* from tb\_order; |

删除成功:

|  |
| --- |
| delete from tb\_order PARTITION (orders\_1) t where t.order\_id=1; commit; select \* from tb\_order; |

为什么?

如果指定了分区, 而条件中的数据右不在该分区中, 将不会删除数据.