ORACLE 数据库

第 3 章

ORACLE 对象

传智播客.黑马程序员

一、视图

（一）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。

（二）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。

（三）删除视图语法

```
DROP  VIEW view_name
```

（四）案例

1. **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：

	ID	NAME	ADDRESSID	HOUSENUMBER	WATERMETER	ADDDATE	OWNERTYPEID
1	1	范冰	1	1-1	30406	2015/9/10 星期四	1
2	2	王刚	1	1-2	30407	2015/9/11 星期五	1
3	3	马腾	1	1-3	30408	2015/9/11 星期五	1
4	4	林小玲	2	2-4	30409	2015/9/11 星期五	1
5	5	刘华	2	2-5	30410	2015/9/11 星期五	1
6	6	刘东	2	2-6	30411	2015/9/11 星期五	1
7	7	周健	3	2-6	30411	2016/9/11 星期日	1
8	8	张哲	4	2-2	30411	2016/9/11 星期日	1

结果已经更改成功。

我们再次查询表数据

	ID	NAME	ADDRESSID	HOUSENUMBER	WATERMETER	ADDDATE	OWNERTYPEID
1	1	范冰	1	1-1	30406	2015/9/10 星期四	1
2	2	王刚	1	1-2	30407	2015/9/11 星期五	1
3	3	马腾	1	1-3	30408	2015/9/11 星期五	1
4	4	林小玲	2	2-4	30409	2015/9/11 星期五	1

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。

2. **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。

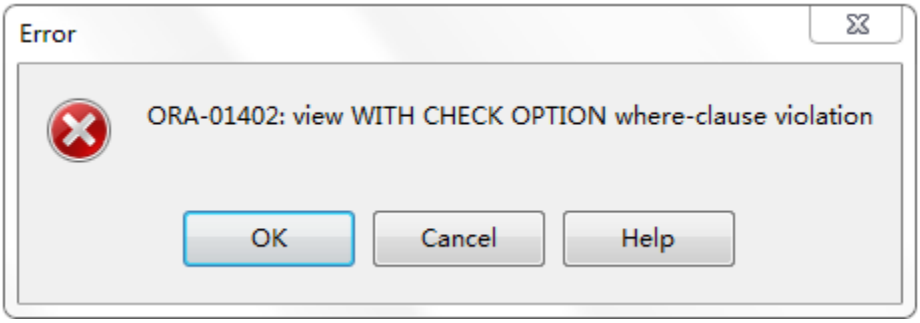
语句：

```
create or replace view view_address2 as
select * from T_ADDRESS where areaid=2
with check option
```

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：



3. **只读视图**的创建与使用

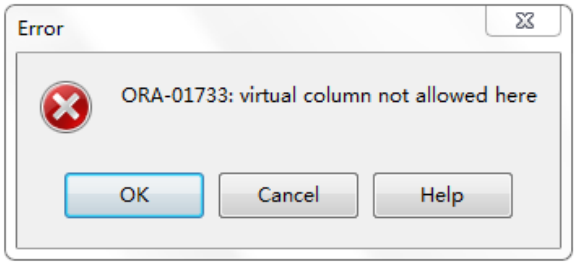
如果我们创建一个视图，并不希望用户能对视图进行修改，那我们就需要创建视图时指定 `WITH READ ONLY` 选项，这样创建的视图就是一个只读视图。

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

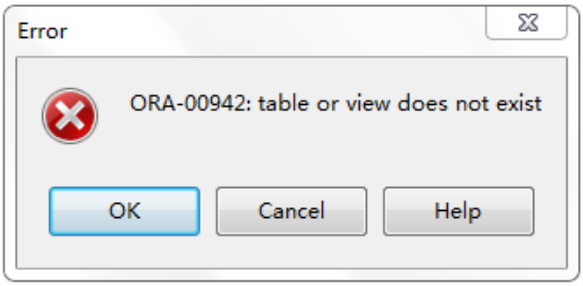


4. 创建**带错误的视图**

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。

5. **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：

（1）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：

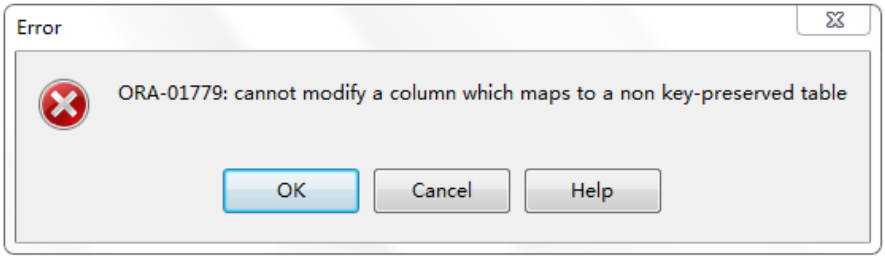
```
update view_owners set 业主名称='范小冰' where 业主编号=1;
```

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;
```

这次我们会发现，系统弹出错误提示：



这个是什么意思？是说我们所需改的列不属于键保留表的列。

什么叫键保留表呢？

键保留表是理解连接视图修改限制的一个基本概念。该表的主键列全部显示在视图中,并且它们的值在视图中都是唯一且非空的。也就是说，表的键值在一个连接视图中也是键值，那么就称这个表为键保留表。

在我们这个例子中，视图中存在两个表，业主表（T_OWNERS）和业主类型表（T_OWNERTYPE），其中 T_OWNERS 表就是键保留表，因为 T_OWNERS 的主键也是作为视图的主键。键保留表的字段是可以更新的，而非键保留表是不能更新的。

（2）分组聚合统计查询的例子

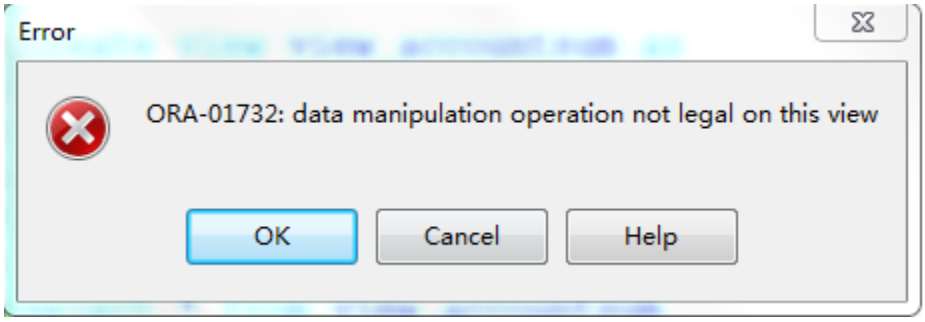
需求：创建视图，按年月统计水费金额，效果如下

	YEAR	MONTH	MONEY
5	2012	01	98046.55
9	2012	02	50254.4
1	2012	03	66635.1
2	2012	04	14320.25
8	2012	05	9790.2
10	2012	06	5833.45
3	2012	07	39873.75
11	2012	08	5206.25
12	2012	09	4412.45
6	2012	10	17527.3
4	2012	11	5500.25
7	2012	12	20376.65

语句：

```
create view view_accountsum as
select year,month,sum(money) moneysum
from T_ACCOUNT
group by year,month
order by year,month
```

此例用到聚合函数，没有键保留表，所以无法执行 update 。



二、物化视图

（一）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。

（二）创建物化视图语法

```
CREATE  METERIALIZED 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。

（三）案例

1.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：

	ID	ADNAME	AR_NAME
1	30	华龙苑南里小区	昌平
2	32	霍营	昌平
3	28	明兴花园	海淀
4	29	鑫源秋墅	海淀
5	31	河畔花园	昌平
6	33	回龙观东大街	西城
7	34	西二旗	西城

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：

	ID	ADNAME	AR_NAME
1	30	华龙苑南里小区	昌平
2	32	霍营	昌平
3	28	明兴花园	海淀
4	29	鑫源秋墅	海淀
5	31	河畔花园	昌平
6	33	回龙观东大街	西城
7	34	西二旗	西城

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;
```

或者通过下面的命令手动刷新物化视图：

```
EXEC DBMS_MVIEW.refresh('MV_ADDRESS','C');
```

注意：此语句需要在命令窗口中执行。

执行此命令后再次查询物化视图，就可以查询到最新的数据了。

DBMS_MVIEW.refresh 实际上是系统内置的存储过程，关于存储过程我们在第 4 章会详细讲解。

2.**创建自动刷新的物化视图**，和上例一样的结果集

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。

3.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：

	ID	ADNAME	AR_NAME

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：

	ID	ADNAME	AR_NAME
1	1	明兴花园	海淀
2	2	鑫源秋墅	海淀
3	3	华龙苑南里小区	昌平
4	4	河畔花园	昌平
5	5	霍营	昌平
6	6	回龙观东大街	西城
7	7	西二旗	西城
8	8	西三旗	西城

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

4.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：

	M_ROW$$	SNAPTIME$$	DMLTYPE$$	OLD_NEW$$	CHANGE_VECTOR$$
▶ 1	AAAM13AAGAAAAEfAAG	4000/1/1 星期六	I	N	FE

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;
```

三、序列

（一）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象。

（二）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
```

（三）创建复杂序列

语法：

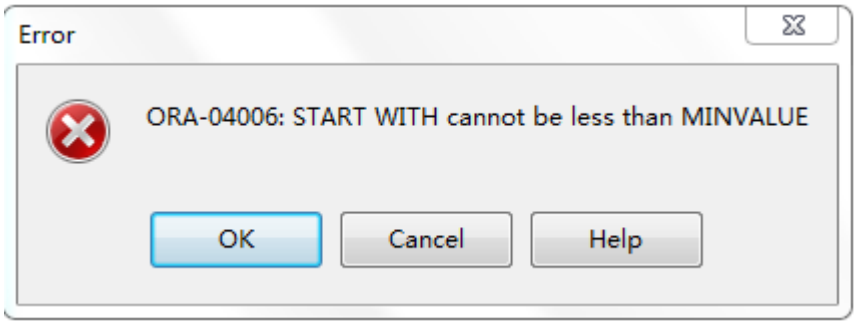
```
CREATE SEQUENCE sequence  //创建序列名称
    [INCREMENT BY n]  //递增的序列值是n  如果n是正数就递增,如果是负数就递减  默认是1
    [START WITH n]    //开始的值,递增默认是 minvalue  递减是 maxvalue
    [{MAXVALUE n | NOMAXVALUE}] //最大值
    [{MINVALUE n | NOMINVALUE}] //最小值
    [{CYCLE | NOCYCLE}] //循环/不循环
    [{CACHE n | NOCACHE}];//分配并存入到内存中
```

（四）案例

1. 有最大值的非循环序列

创建序列的语句：

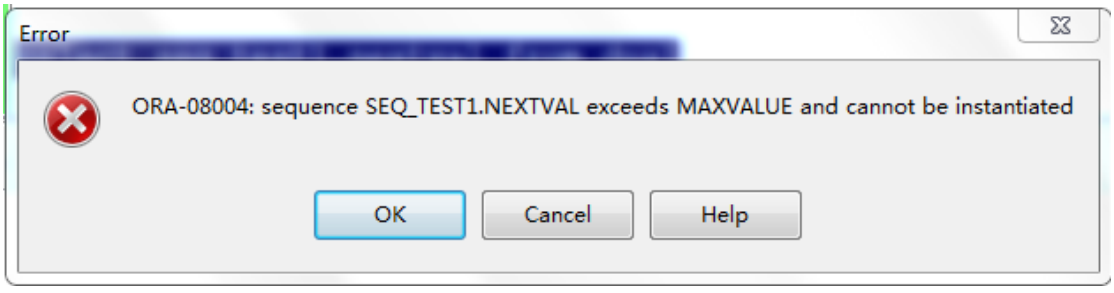
```
create sequence seq_test1
increment by 10
start with 10
maxvalue 300
minvalue 20
```



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
```

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。



2. 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
```

提取的值为：

	NEXTVAL
1	5

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 ;
```

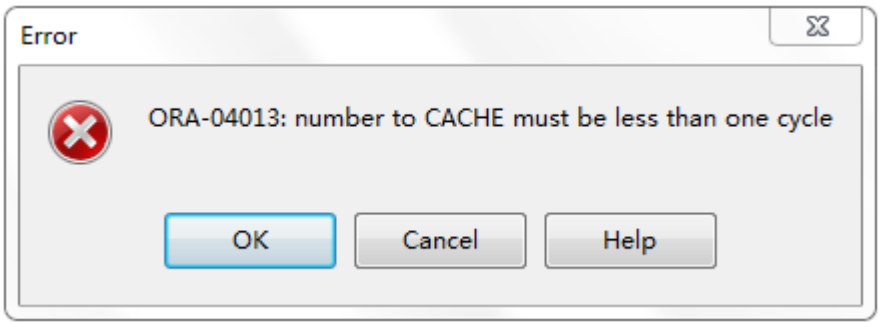
答：此为错误的语句。因为你创建的是一个循环的序列，所以必须指定最大值，否则会报错。

3. 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 490 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。

（五）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;
```

四、同义词

（一）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。

（二）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，序列等我们要创建同义词的对象的名称。

（三）案例

1.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：

	ID	NAME	ADDRESSID	HOUSENUMBER	WATERMETER	ADDDATE	OWNERTYPEID
1	1	范小冰	1	1-1	30406	2015/9/10 星期四	1
2	2	王刚	1	1-2	30407	2015/9/11 星期五	1
3	3	马腾	1	1-3	30408	2015/9/11 星期五	1
4	4	林小玲	2	2-4	30409	2015/9/11 星期五	1
5	5	刘华	2	2-5	30410	2015/9/11 星期五	1
6	6	刘东	2	2-6	30411	2015/9/11 星期五	1
7	7	周健	3	2-6	30412	2016/9/11 星期日	1
8	8	张哲	4	2-2	30413	2016/9/11 星期日	1
9	21	昌平区中西医结合医院	5	2-2	30422	2016/10/11 星期二	2
10	22	美廉美超市	5	4-2	30423	2016/10/12 星期三	3

2.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：

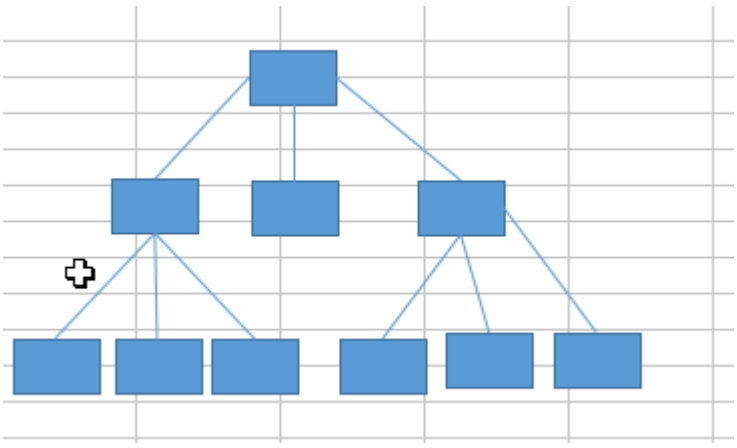
```
select * from OWNERS2 ;
```

五、索引

（一）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）



（二）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,'AA'||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='AA765432';
```

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

（三）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);
```

（四）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);
```

（五）反向键索引

应用场景：当某个字段的值为连续增长的值，如果构建标准索引，会形成歪脖子树。这样会增加查询的层数，性能会下降。建立反向键索引，可以使索引的值变得不规则，从而使索引树能够均匀分布。

1	‘0001	‘1000
2	’ 0010	’ 0100
3	‘0011	‘1100
4	’ 0100	’ 0010
5	‘0101	‘1010
6	’ 0110	’ 0110

语法：

```
create index 索引名称 on 表名(列名) reverse;
```

（六）位图索引

使用场景：位图索引适合创建在低基数列上

位图索引不直接存储 ROWID，而是存储字节位到 ROWID 的映射

优点：减少响应时间，节省空间占用

语法：

```
create bitmap index 索引名称 on 表名(列名);
```

需求：我们在 T_owners 表的 ownertypeid 列上建立位图索引，语句：

```
create bitmap index index_owners_typeid
on T_OWNERS(ownertypeid)
```

六、总结

（一）知识点总结

（二）上机任务布置

基于 scott 用户联系各种数据库对象的创建。

因为 scott 用户权限较低，而创建各种数据库对象需要较高权限，所以需要赋予

scott 用户 dba 权限

```
grant dba to scott
```

1. 创建视图 view_emp，显示雇员表中的 EMPNO　ENAME　JOB

2. 创建带约束的视图 view_emp30，显示部门编号为 30 的雇员信息。

3. 创建只读视图，显示部门表中的信息。

4. 创建物化视图（自动刷新），显示雇员编号、雇员名称、雇员职位和雇员部门。

5. 创建物化视图（手动刷新），查询列出各部门的部门名和部门经理名字。并编写手动刷新命令。

6. 编写序列 seq_1 ,从 100 开始 ，增长 10 ，最大值 1000，最小值 10 ，循环 。

7. 编写序列 seq_2 ,最大值 100 ，最小值 5，增长值 5 ，不循环。

8. 编写序列 SEQ_EMP, 起始值 8000，增长 1 ，不循环，不缓存。

9. 编写序列 SEQ_DEPT, 起始值 50，增长 10 ，不循环，缓存 30。

10. 根据雇员名称对雇员表建立索引

11. 根据部门编号和职位对雇员表建立索引

12. 在奖金表根据职位建立位图索引

13.为 EMP 表创建私有同义词

14.为 DEPT 表创建公有同义词