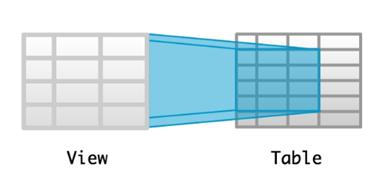
* 视图、序列、索引

**1.1. 视图**

**1.1.1. 什么是视图**

视图(VIEW)也被称作虚表，即虚拟的表，是一组数据的逻辑表示,其本质是对应于一条SELECT语句，结果集被赋予一个名字，即视图名字。

视图本身并不包含任何数据，它只包含映射到基表的一个查询语句，当基表数据发生变化，视图数据也随之变化。



创建视图的语法：

01.CREATE [OR REPLACE] VIEW view\_name[(alias[, alias…])]

02.AS subquery ;

视图创建后，可以像操作表一样操作视图，主要是查询操作。

语法中的Subquery是SELECT查询语句，对应的表被称作基表。

根据视图所对应的子查询种类分为几种类型:

•SELECT语句是基于单表建立的，且不包含任何函数运算、表达式或分组函数，叫做简单视图，此时视图是基表的子集；

•SELECT语句同样是基于单表，但包含了单行函数、表达式、分组函数或GROUP BY子句，叫做复杂视图；

•SELECT语句是基于多个表的，叫做连接视图。

**1.1.2. 视图的作用**

如果需要经常执行某项复杂查询，可以基于这个复杂查询建立视图，此后查询此视图即可，简化复杂查询；

视图本质上就是一条SELECT语句，所以当访问视图时，只能访问到所对应的SELECT语句中涉及到的列，对基表中的其它列起到安全和保密的作用，可以限制数据访问。

**1.1.3. 授权创建视图**

创建视图的DDL语句是CREATE VIEW，用户必须有CREATE VIEW系统权限，才能创建视图。如果没有权限，创建视图时会提示：权限不足。

管理员可以通过DCL语句授予用户创建视图的权限。下例中管理员给用户tarena创建视图的权限：

01.GRANT CREATE VIEW TO tarena;

**1.1.4. 创建简单视图（单表）**

创建一个简单视图V\_EMP\_10，来显示部门10中的员工的编码、姓名和薪水：

01.CREATE VIEW v\_emp\_10

02.AS

03.SELECT empno, ename, sal, deptno

04.FROM emp

05.WHERE deptno = 10;

查看视图结构：

01.DESC v\_emp\_10;

**1.1.5. 查询视图**

查询视图和查询表的操作相同：

01.SELECT \* FROM v\_emp\_10;

此时视图的列名，和创建视图时的列名一致，不一定是原列名：

01.SELECT id, name, salary FROM v\_emp\_10;

**1.1.6. 对视图进行INSERT操作**

视图本身并不包含数据，只是基表数据的逻辑映射。所以当对视图执行DML操作时，实际上是对基表的DML操作。对视图执行DML操作的基本原则：

•简单视图能够执行DML操作，下列情况除外：在基表中定义了非空列，但简单视图对应的SELECT语句并没有包含这个非空列，导致这个非空列对视图不可见，这时无法对视图执行INSERT操作；

•如果视图定义中包含了函数、表达式、分组语句、DISTINCT关键字或ROWNUM伪列，不允许执行DML操作；

•DML操作不能违反基表的约束条件。

对简单视图执行INSERT操作，成功插入数据到基表中:

01.INSERT INTO v\_emp\_10

02.VALUES(1234, 'DOCTOR', 4000, 10);

简单视图可以通过DML操作影响到基表数据。

**1.1.7. 创建具有CHECK OPTION约束的视图**

语法如下：

01.CREATE [OR REPLACE] VIEW view\_name[(alias[, alias…])]

02.AS subquery

03.[WITH CHECK OPTION];

其中：WITH CHECK OPTION短语表示，通过视图所做的修改，必须在视图的可见范围内：

•假设INSERT，新增的记录在视图仍可查看

•假设UPDATE，修改后的结果必须能通过视图查看到

•假设DELETE，只能删除现有视图里能查到的记录

创建带有CHECK OPTION约束的视图：

01.CREATE OR REPLACE VIEW v\_emp\_10

02.AS

03.SELECT empno id, ename name, sal salary, deptno

04.FROM emp

05.WHERE deptno = 10

06.WITH CHECK OPTION;

下述DML语句操作失败，因为部门20不在视图可见范围内：

01.INSERT INTO v\_emp\_10 VALUES(1008,‘donna’,5500, 20);

02.UPDATE v\_emp\_10 SET deptno = 20 WHERE id = 7782;

**1.1.8. 创建具有READ ONLY约束的视图**

对简单视图进行DML操作是合法的，但是不安全的。如果没有在视图上执行 DML 操作的必要，在建立视图时声明为只读来避免这种情况，保证视图对应的基表数据不会被非法修改。加入READ ONLY约束的视图语法如下：

01.CREATE [OR REPLACE] VIEW view\_name[(alias[, alias…])]

02.AS subquery

03.[WITHREAD ONLY];

创建视图，带有READ ONLY约束：

01.CREATE OR REPLACE VIEW v\_emp\_10

02.AS

03.SELECT empno, ename, sal, deptno FROM emp

04.WHERE deptno = 10

05.WITH READ ONLY;

此时对只读视图执行DML操作，将会失败：

01.INSERT INTO v\_emp\_10

02.VALUES(1258, 'DONNA', 3000, 10);

得到如下结果：

ERROR 位于第 1 行:

ORA-01733: 此处不允许虚拟列

**1.1.9. 通过查询USER\_VIEWS获取相关信息**

和视图相关的数据字典有：

•USER\_OBJECTS

•USER\_VIEWS

•USER\_UPDATABLE\_COLUMNS

例一：在数据字典USER\_OBJECTS中查询所有视图名称：

01.SELECT object\_name FROM user\_objects

02.WHERE object\_type = 'VIEW';

例二：在数据字典USER\_VIEWS中查询指定视图：

01.SELECT text FROM user\_views

02.WHERE view\_name = 'V\_EMP\_10';

例三：在数据字典USER\_UPDATABLE\_COLUMNS中查询视图：

01.SELECT column\_name, insertable, updatable, deletable

02.FROM user\_updatable\_columns

03.WHERE table\_name = 'V\_EMP\_10';

**1.1.10. 创建复杂视图（多表关联）**

复杂视图指在子查询中包含了表达式、单行函数或分组函数的视图。此时必须为子查询中的表达式或函数定义别名。

例如，创建一个视图V\_EMP\_SALARY，把职员表的数据按部门分组，获得每个部门的平均薪水、薪水总和、最高薪水和最低薪水：

01.CREATE VIEW v\_emp\_salary

02.AS

03.SELECT d.dname, avg(e.sal) avg\_sal, sum(e.sal) sum\_sal,

04.max(e.sal) max\_sal, min(e.sal) min\_sal

05.FROM emp e join dept d

06.ON e.deptno = d.deptno

07.GROUP BY d.dname;

查询复杂视图：

01.SELECT \* FROM v\_emp\_salary;

复杂视图不允许DML操作，会报错。

当不再需要视图的定义，可以使用DROP VIEW语句删除视图，语法如下：

01.DROP VIEW view\_name;

例如删除视图v\_emp\_10：

01.DROP VIEW v\_emp\_10;

视图虽然是存放在数据字典中的独立对象，但视图仅仅是基于表的一个查询定义，所以对视图的删除不会导致基表数据的丢失，不会影响基表数据。

**1.2. 序列**

**1.2.1. 什么是序列**

序列(SEQUENCE)是一种用来生成唯一数字值的数据库对象。序列的值由Oracle程序按递增或递减顺序自动生成，通常用来自动产生表的主键值，是一种高效率获得唯一键值的途径。

序列是独立的数据库对象，和表是独立的对象，序列并不依附于表。

通常情况下，一个序列为一个表提供主键值，但一个序列也可以为多个表提供主键值。

**1.2.2. 创建序列**

创建序列的语法：

01.CREATE SEQUENCE [schema.]sequence\_name

02. [ START WITH i ] [ INCREMENT BY j ]

03. [ MAXVALUE m | NOMAXVALUE ]

04. [ MINVALUE n | NOMINVALUE ]

05. [ CYCLE | NOCYCLE ][ CACHE p | NOCACHE ]

其中：

•sequence\_name是序列名，将创建在schema方案下

•序列的第一个序列值是i，步进是j

•如果j是正数，表示递增，如果是负数，表示递减

•序列可生成的最大值是m，最小值是n

•如果没有设置任何可选参数，序列的第一个值是1，步进是1

•CYCLE表示在递增至最大值或递减至最小值之后是否继续生成序列号，默认是NOCYCLE

•CACHE用来指定先预取p个数据在缓存中，以提高序列值的生成效率，默认是20

**1.2.3. 使用序列**

举例说明，创建一个序列，起始数据是100，步进是10：

01.CREATE SEQUENCE emp\_seq

02. START WITH 100

03. INCREMENT BY 10;

当序列被创建后，第一个序列值将是100，将要生成的序列号分别是110、120、130等。

序列中有两个伪列：

•NEXTVAL：获取序列的下个值

•CURRVAL：获取序列的当前值

当序列创建以后，必须先执行一次NEXTVAL，之后才能使用CURRVAL。

获取序列的第一个值，并且使用序列值为EMP表插入新的记录：

01. SELECT emp\_seq.NEXTVAL FROM DUAL;

02. INSERT INTO emp(empno, ename)

03. VALUES(emp\_seq.NEXTVAL, 'donna');

查询刚刚生成的记录，主键值将是110：

01.SELECT empno, ename FROM emp

02. WHERE ename = 'DONNA';

此时查询序列的当前值，会得到110的数字。

01.SELECT emp\_seq.CURRVAL FROM DUAL;

在序列的使用过程中，比如执行了一条语句：

01.SELECT emp\_seq.NEXTVAL FROM DUAL

则浪费了一个序列值，会导致表的主键值不连续。而CURRVAL的使用不会导致序列值的递进。

**1.2.4. 删除序列**

删除序列的语法如下：

01.DROP SEQUENCE sequence\_name;

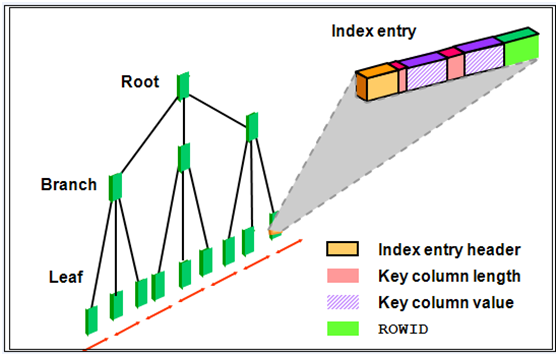
**1.3. 索引**

**1.3.1. 索引的原理**

索引是一种允许直接访问数据表中某一数据行的树型结构，为了提高查询效率而引入，是独立于表的对象，可以存放在与表不同的表空间（TABLESPACE）中。索引记录中存有索引关键字和指向表中数据的指针（地址）。对索引进行的I/O操作比对表进行操作要少很多。

索引一旦被建立就将被Oracle系统自动维护，查询语句中不用指定使用哪个索引，是一种提高查询效率的机制

图索引的结构



ROWID: 伪列，唯一标识一条数据记录，可理解为行地址。

**1.3.2. 创建索引**

创建索引的语法：

01.CREATE [UNIQUE] INDEX index\_name

02. ON table(column[, column…]);

其中：

•index\_name表示索引名称

•table表示表名

•column表示列名，可以建立单列索引或复合索引

•UNIQUE表示唯一索引

在EMP表的ENAME列上建立索引：

01.CREATE INDEX idx\_emp\_ename ON emp(ename);

复合索引也叫多列索引，是基于多个列的索引。如果经常在ORDER BY子句中使用job和salary作为排序依据，可以建立复合索引：

01.CREATE INDEX idx\_emp\_job\_sal ON emp(job, sal);

当做下面的查询时，会自动应用索引idx\_emp\_job\_sal

01.SELECT empno, ename, sal, job FROM emp

02.ORDER BY job, sal;

**1.3.3. 创建基于函数的索引**

如果需要在emp表的ename列上执行大小写无关搜索，可以在此列上建立一个基于UPPER函数的索引:

01. CREATE INDEX emp\_ename\_upper\_idx

02. ON emp(UPPER(ename));

当做下面的查询时，会自动应用刚刚建立的索引:

01. SELECT \* FROM emp

02. WHERE UPPER(ename) = 'KING';

**1.3.4. 修改和删除索引**

如果经常在索引列上执行DML操作，需要定期重建索引，提高索引的空间利用率，语法如下：

01.ALTER INDEX index\_name REBUILD;

当一个表上有不合理的索引，会导致操作性能下降，删除索引的语法：

01.DROP INDEX index\_name;

**1.3.5. 合理使用索引提升查询效率**

为提升查询效率，创建和使用索引的原则：

•为经常出现在WHERE子句中的列创建索引

•为经常出现在ORDER BY、DISTINCT后面的字段建立索引。如果建立的是复合索引，索引的字段顺序要和这些关键字后面的字段顺序一致

•为经常作为表的连接条件的列上创建索引

•不要在经常做DML操作的表上建立索引

•不要在小表上建立索引

•限制表上的索引数目，索引并不是越多越好

•删除很少被使用的、不合理的索引

**2. 约束**

**2.1. 约束概述**

**2.1.1. 约束的作用**

约束（CONSTRAINT）的全称是约束条件，也称作完整性约束条件。约束是在数据表上强制执行的一些数据校验规则，当执行DML操作时，数据必须符合这些规则，如果不符合则无法执行。

约束条件可以保证表中数据的完整性，保证数据间的商业逻辑。

**2.1.2. 约束的类型**

约束条件包括：

•非空约束(Not Null)，简称NN

•唯一性约束(Unique)，简称UK

•主键约束(Primary Key)，简称PK

•外键约束(Foreign Key)，简称FK

•检查约束(Check)，简称CK

**2.2. 非空约束**

**2.2.1. 建表时添加非空约束**

非空约束用于确保字段值不为空。默认情况下，任何列都允许有空值，但业务逻辑可能会要求某些列不能取空值。当某个字段被设置了非空约束条件，这个字段中必须存在有效值，即：

•当执行INSERT操作时，必须提供这个列的数据

•当执行UPDATE操作时，不能给这个列的值设置为NULL

建表时添加非空约束：

01.CREATE TABLE employees (

02.eid NUMBER(6),

03.name VARCHAR2(30) NOT NULL,

04.salary NUMBER(7, 2),

05.hiredate DATE

06. CONSTRAINT employees\_hiredate\_nn NOT NULL

07.);

**2.2.2. 修改表时添加非空约束**

可以在建表之后，通过修改表的定义，添加非空约束：

01.ALTER TABLE employees

02.MODIFY (eid NUMBER(6) NOT NULL);

**2.2.3. 取消非空约束**

如果业务要求取消某列的非空约束，可以采用重建表或者修改表的方式：

01.ALTER TABLE employees

02.MODIFY (eid NUMBER(6));

**2.3. 唯一性约束**

**2.3.1. 什么是唯一性约束**

唯一性(Unique)约束条件用于保证字段或者字段的组合不出现重复值。当给表的某个列定义了唯一约束条件，该列的值不允许重复，但允许是NULL值。

唯一性约束条件可以在建表同时建立，也可以在建表以后再建立。

2.3.2. 添加唯一性约束

在建表employees的同时，在eid、email列上创建唯一约束条件，并在建表后在name列上建立一个名为employees\_name\_uk的唯一约束条件：

01.DROP TABLE employees ; --将表删掉重新创建

02.CREATE TABLE employees (

03.eid NUMBER(6) UNIQUE,

04.name VARCHAR2(30),

05.email VARCHAR2(50),

06.salary NUMBER(7, 2),

07.hiredate DATE,

08.CONSTRAINT employees\_email\_uk UNIQUE(email)

09. );

在建表之后增加唯一性约束条件：

01.ALTER TABLE employees

02.ADD CONSTRAINT employees\_name\_uk UNIQUE(name);

**2.4. 主键约束**

**2.4.1. 主键的意义**

主键(Primary Key)约束条件从功能上看相当于非空（NOT NULL）且唯一（UNIQUE）的组合。主键字段可以是单字段或多字段组合，即：在主键约束下的单字段或者多字段组合上不允许有空值，也不允许有重复值。

主键可以用来在表中唯一的确定一行数据。一个表上只允许建立一个主键，而其它约束条件则没有明确的个数限制。

**2.4.2. 主键选取的原则**

•主键应是对系统无意义的数据

•永远也不要更新主键，让主键除了唯一标识一行之外，再无其他的用途

•主键不应包含动态变化的数据，如时间戳

•主键应自动生成，不要人为干预，以免使它带有除了唯一标识一行以外的意义

•主键尽量建立在单列上

**2.5. 外键约束**

**2.5.1 外键约束的意义**

外键约束条件定义在两个表的字段或一个表的两个字段上，用于保证相关两个字段的关系。比如emp表的deptno列参照dept表的deptno列，则dept称作主表或父表，emp表称作从表或子表。

**2.5.2 添加外键约束**

先建表，在建表后建立外键约束条件：

08.ALTER TABLE employees4

09.ADD CONSTRAINT employees4\_deptno\_fk

10.FOREIGN KEY (deptno) REFERENCES dept(deptno);

**2.5.3.外键约束条件包括两个方面的数据约束：**

•从表上定义的外键的列值，必须从主表被参照的列值中选取，或者为NULL；

•当主表参照列的值被从表参照时，主表的该行记录不允许被删除。

数据时不用考虑外键约束，以及大量数据DML操作时不需考虑外键耗费时间。

**2.6. 检查约束**

**2.6.1 什么是检查约束**

检查(Check)约束条件用来强制在字段上的每个值都要满足Check中定义的条件。当定义了Check约束的列新增或修改数据时，数据必须符合Check约束中定义的条件。

2.6.2 添加检查约束

员工的薪水必须大于2000元，增加检查约束：

01.ALTER TABLE employees4

02.ADD CONSTRAINT employees4\_salary\_check

03.CHECK (salary > 2000);

当插入大于2000的数据，操作成功：

01.INSERT INTO employees4(eid, name, salary, deptno)

02.VALUES(1236, 'donna noble', 2500, 40);

试图修改职员的薪水为1500元，更新失败：

01.UPDATE employees4 SET salary = 1500

02. WHERE eid = 1236;

* 课后练习整理(基础查询)

1:查看工资高于2000的员工

SELECT ename,sal FROM emp\_wwww

WHERE sal>2000

2:查看不是"CLERK"职位的员工

SELECT ename,job FROM emp\_wwww

WHERE job NOT IN('CLERK')

3:查看工资在1000-2500之间的员工

SELECT ename,sal FROM emp\_wwww

WHERE sal>1000 AND sal<2500

SELECT ename,sal FROM emp\_wwww

WHERE sal BETWEEN 1000 AND 2500

4:查看名字是以K结尾的员工

SELECT ename FROM emp\_wwww

WHERE ename LIKE '%K'

5:查看20,30号部门的员工

SELECT ename,job,deptno FROM emp\_wwww

WHERE deptno IN(20,30)

6:查看奖金为NULL的员工

SELECT ename,comm

FROM emp\_wwww

WHERE comm IS NULL

7:查看年薪高于20000的员工

SELECT ename,sal\*12 "年薪"

FROM emp\_wwww

WHERE sal\*12>20000

8:查看公司共有多少种职位

SELECT DISTINCT job From emp\_wwww

9:按部门号从小到大排列查看员工

SELECT ename,deptno

FROM emp\_wwww

ORDER BY deptno ASC

10:查看每个部门的最高，最低，平均工资，和工资总和

SELECT MAX(sal),MIN(sal)

FROM emp\_wwww

GROUP BY job

SELECT AVG(NVL(comm,0)),SUM(sal)

FROM emp\_wwww

GROUP BY job

11:查看平均工资高于2000的部门的最低薪水

SELECT MIN(sal),job

FROM emp\_wwww

GROUP BY job

HAVING AVG(sal)>2000

12:查看在NEWYORK工作的员工

SELECT e.ename,d.loc

FROM emp\_wwww e JOIN dept\_wwww d

ON d.deptno=e.deptno

WHERE d.loc='NEW YORK'

13:查看所有员工及所在部门信息，若该员工没有部门，则部门信息以NULL显示

SELECT e.ename,d.dname

FROM emp\_wwww e LEFT OUTER JOIN dept\_wwww d

ON e.deptno=d.deptno

14:查看ALLEN的上司是谁

SELECT m.ename

FROM emp\_wwww e JOIN emp\_wwww m

ON e.mgr=m.empno

AND e.ename='ALLEN'

15:查看SMITH上司在哪个城市工作?

SELECT d.loc

FROM emp\_wwww e JOIN emp\_wwww m

ON e.mgr=m.empno

JOIN dept\_wwww d

ON e.deptno=d.deptno

AND e.ename='SMITH'

16:查看平均工资高于2000的那些部门名字以及所在城市?

SELECT dname,loc

FROM dept\_wwww

WHERE deptno IN

(SELECT e.deptno

FROM emp\_wwww e

GROUP BY e.deptno

HAVING AVG(e.sal)>2000)

17:在NEW YORK工作的员工有多少人?

SELECT e.ename,d.loc

FROM emp\_wwww e JOIN dept\_wwww d

ON d.deptno=e.deptno

WHERE d.loc='NEW YORK'

18:在DALLAS工作的员工的平均工资是多少?

SELECT AVG(e.sal)

FROM emp\_wwww e JOIN dept\_wwww d

ON e.deptno=d.deptno

WHERE d.loc='DALLAS'

19:查看SMITH的上司是谁?他在哪个城市工作?

SELECT m.ename,d.loc

FROM emp\_wwww e JOIN emp\_wwww m

ON e.mgr=m.empno

JOIN dept\_wwww d

ON d.deptno=e.deptno

AND e.ename='SMITH'

* 课后练习整理(高级查询)

1:查看与CLARK相同职位的员工

SELECT ename,job

FROM emp\_wwww

WHERE job=(SELECT job FROM emp\_wwww WHERE ename='CLARK')

2:查看低于公司平均工资的员工

SELECT ename

FROM emp\_wwww

WHERE sal<(SELECT AVG(sal) FROM emp\_wwww)

3:查看与ALLEN同部门的员工

SELECT ename,sal,deptno,job

FROM emp\_wwww

WHERE deptno IN(SELECT deptno

FROM emp\_wwww

WHERE ename='ALLEN')

4:查看平均工资低于20号部门平均工资的部门平均工资

SELECT AVG(sal),deptno

FROM emp\_wwww

GROUP BY deptno

HAVING

AVG(sal)<

(SELECT AVG(sal)

FROM emp\_wwww

WHERE deptno=20)

5:查看低于自己所在部门平均工资的员工

CREATE OR REPLACE VIEW v\_emp\_wwww\_sal

AS

SELECT AVG(sal) s,deptno

FROM emp\_wwww

GROUP BY deptno

SELECT DISTINCT e.ename,e.sal,e.deptno

FROM emp\_wwww e , v\_emp\_wwww\_sal v

WHERE e.sal<v.s

AND e.deptno=v.deptno

6:查看公司工资排名的第1-5名

SELECT ename,sal,empno,deptno,

ROW\_NUMBER()OVER(

ORDER BY sal DESC) rank

FROM(SELECT ROWNUM rank,empno,ename,sal,deptno

FROM emp\_wwww)

WHERE rank BETWEEN 1 AND 5

7:查看CLERK职位的人数和其他职位的总人数各多少?

SELECT COUNT(\*),

DECODE(job,

'CLERK','VIP',

'OTHER')

FROM emp\_wwww

GROUP BY DECODE(job,

'CLERK','VIP',

'OTHER')

8:查看每个职位的工资排名

SELECT ename,job,sal,

ROW\_NUMBER()OVER(

PARTITION BY job

ORDER BY sal DESC) rank

FROM emp\_wwww

9:查看每个职位的工资排名，若工资一致，排名一致

SELECT ename,job,sal,

RANK()OVER(

PARTITION BY job

ORDER BY sal DESC) rank

FROM emp\_wwww

10:查看每个职位的工资排名，若工资一致，排名一致，不跳名次。

SELECT ename,job,sal,

DENSE\_RANK()OVER(

PARTITION BY job

ORDER BY sal DESC

) rank

FROM emp\_wwww

11:分别查看:同部门同职位，同职位，以及所有员工的工资总和

SELECT deptno,job,SUM(sal)

FROM emp\_wwww

GROUP BY ROLLUP(job,deptno)

12:分别查看:同部门同职位，同职位，同部门以及所有员工的工资总和

SELECT deptno,job,SUM(sal)

FROM emp\_wwww

GROUP BY CUBE(job,deptno)

13:分别查看同部门同职位和同职位的员工的工资总和

14:查看公司最高工资的员工的名字以及所在部门名称

SELECT e.ename,d.dname

FROM emp\_wwww e,dept\_wwww d

WHERE e.deptno=d.deptno

AND e.ename IN(SELECT ename

FROM emp\_wwww e

WHERE sal=(SELECT MAX(sal)

FROM emp\_wwww))

15:查看每个部门的最高工资的员工名字

SELECT ename,sal

FROM emp\_wwww

WHERE sal IN (SELECT MAX(sal)

FROM emp\_wwww

GROUP BY deptno)

16:查看有下属的员工信息

SELECT DISTINCT e.ename,e.empno,e.job,e.deptno

FROM emp\_wwww e,emp\_wwww m

WHERE m.mgr=e.empno

* 课后练习整理(视图.序列.约束)

1:创建一个视图，包含20号部门的员工信息，

字段:empno,ename,sal,job,deptno

CREATE OR REPLACE VIEW v\_emp\_wwww\_20

AS

SELECT empno,ename,sal,job,deptno

FROM emp\_wwww

WHERE deptno=20;

SELECT \* FROM v\_emp\_wwww\_20

2:创建一个序列seq\_emp\_no,从10开始，步进为10

CREATE SEQUENCE seq\_emp\_no

START WITH 10

INCREMENT BY 1

3:编写SQL语句查看seq\_emp\_no序列的下一个数字

SELECT seq\_emp\_no.NEXTVAL

FROM dual

4:编写SQL语句查看seq\_emp\_no序列的当前数字

SELECT seq\_emp\_no.CURRVAL

FROM dual

5:为emp表的ename字段添加索引:idx\_emp\_ename

CREATE INDEX idx\_emp\_wwww\_ename ON emp\_wwww(enamel)

6:为emp表的LOWER(ename)字段添加索引:idx\_emp\_lower\_ename

CREATE INDEX idx\_emp\_wwww\_lower\_ename ON emp\_wwww(lower(ename))

7:为emp表的sal,comm添加多列索引

CREATE INDEX idx\_emp\_wwww\_sal\_comm ON emp\_wwww(sal,comm)

8:创建myemployee表，字段:

CREATE TABLE myemployee\_w(

id NUMBER(4) PRIMARY KEY, -- id作为主键

name VARCHAR2(20) NOT NULL, -- name要求不能为空

birthday DATE,

telephone VARCHAR2(11), --telephone需要唯一

score NUMBER(9,2), --score值必须>=0

CONSTRAINT employeesl\_telephone\_uk UNIQUE(telephone),

CHECK(score>=0)

)

DESC myemployee\_w