2
数据库管理

知识提要：

本章介绍了如何创建学生成绩管理系统的表空间、表，以及如何用 PL/SQL 方式和命令方式操作表，如何创建主键约束、外键约束、唯一性约束、检查约束、非空约束，如何创建序列、同义词，还介绍了索引简介、索引的分类、创建索引，最后介绍如何进行插入记录、删除记录、修改记录等更新数据库的操作。

教学目标：

- 了解表空间;
- 能够用 PL/SQL 方式和命令方式操作表;
- 能够创建主键约束、外键约束、唯一性约束、检查约束、非空约束;
- 能够创建序列、同义词、索引;
- 能够进行插入记录、删除记录、修改记录等更新数据库的操作。

2.1 创建表空间

表空间（TableSpace）是 Oracle 的开创性理念。表空间使得数据库管理更加灵活，而且极大地提高了数据库性能，比如：

（1）避免磁盘空间突然耗竭的风险。

（2）规划数据更灵活。

（3）提高数据库性能。

（4）提高数据库安全性。

下面进行具体的介绍。

（1）创建一个简单的表空间。

```
Create tablespace user1 datafile 'e:\database\oracle\user1_data.dbf' size 200M;
```

```
SQL> create tablespace user1 datafile 'e:\database\oracle\user1_data.dbf' size 2
00M;

Tablespace created.

SQL>
```

（2）指定数据文件的可扩展性。

```
Create tablespace user2 datafile 'e:\database\oracle\user2_data.dbf' size 200M autoextend on;
```

```
SQL> create tablespace user2 datafile 'e:\database\oracle\user2_data.dbf' size 2
00M autoextend on;

Tablespace created.

SQL>
```

（3）指定数据文件的增长幅度。

```
Create tablespace user3 datafile 'e:\database\oracle\user3_data.dbf' size 200M autoextend on next 5M;
```

```
SQL> create tablespace user3 datafile 'e:\database\oracle\user3_data.dbf' size 2
00M autoextend on next 5M;

Tablespace created.

SQL>
```

（4）指定数据文件的最大尺寸。

```
Create tablespace user4 datafile 'e:\database\oracle\user4_data.dbf' size 200M autoextend on next 5M Maxsize 500M;
```

```
SQL> create tablespace user4 datafile 'e:\database\oracle\user4_data.dbf' size 2
00M autoextend on next 5M Maxsize 500M;

Tablespace created.

SQL>
```

2.2 创建表

Oracle 表空间的下一层逻辑结构即为数据表。数据表也是各种数据库中共有的、开发人员和 DBA 最常打交道的数据库对象，本节着重介绍如何创建 Oracle 数据表。

2.2.1 PL/SQL 方式操作表

很多数据库管理工具都提供了图形化界面来创建数据表，如 MS SQL Server 企业管理器。

针对 Oracle 数据库，PL/SQL Developer 是一个不错的选择。利用 PL/SQL 工具创建数据表，操作简单、直观、易于掌握。

用 PL/SQL Developer 创建数据表的步骤如下：“打开文件”→“新建”→“表”命令（如图 2-1 所示），进入创建界面，创建数据库名称，添加数据列，设置类型，然后单击“应用”按钮创建数据库。

图 2-1　利用 PL/SQL 工具创建数据表

2.2.2　命令方式操作表

利用命令同样可以创建数据表，其效果与利用工具完全相同。

```
Create table cjb (c_xh char(10),c_kch char(10),c_cj float)
```

创建成功后可以用 Describe cjb 命令查看创建的数据库结构。

2.3　用约束保障数据的完整性

约束是每个数据库必不可少的一部分。约束的根本目的在于保持数据的完整性。数据完整性是指数据的精确性和可靠性。即数据库中的数据都是符合某种预定义规则。当用户输入的数据不符合这些规则时，将无法实现对数据库的更改。

主键约束：主键约束是数据库中最常见的约束。主键约束可以保证数据完整性。即防止数据表中的两条记录完全相同，通过将主键纳入查询条件，可以达到查询结果最多返回一条记录的目的。

外键约束：外键与主键一样用于保证数据完整性，主键是针对单个表的约束，而外键则描述了表之间的关系。即两个表之间的数据的相互依存性。

唯一性约束：唯一性约束与主键一样，用于唯一标识一行。使用唯一性约束的列或列的组合，其值或值的组合必须是唯一的。

检查约束：在前面介绍的约束（主键、外键、唯一性约束）实际在定义多个列值之间的关系，例如，主键和唯一性都约束表中的两个列值或列值组合不能相同，而外键则约束了两个表之间的数据保持父子关系。检查约束则是针对列值本身进行限制。

默认值约束：默认值约束也是数据库中常用约束。当向数据表中插入数据时，并不总是将所有字段一一插入。对于某些特殊字段，其值总是固定或者差不多的。用户希望如果没有显式指定值，就使用某个特定的值进行插入，即默认值。为列指定默认值的操作即为设置默认值约束。

2.3.1 主键约束的创建

主键被创建在一个或多个列上，通过这些列的值或者值的组合，唯一地标识一条记录。例如，对于存储了学生信息的 student 表，一般会为每个学生分配一个 student_id，也就是说将主键建立在 student_id 这个列上。student_id 将成为每个学生的唯一标识。当向 student 表中插入新的学生信息时，如果要插入的 student_id 已经存在，数据库将拒绝插入该条记录。这就是主键保证数据完整性的体现。对于主键，有以下几点需要注意。

- 主键列的数据类型并不一定是数值型。
- 主键列不一定只有一列。
- 主键是规则制定者的主观体现，不要将其与现实世界混淆。

1. 创建主键约束

创建主键约束：

```
Create table xsb(c_xh char(10),c_xm char(12),c_xb char(6),c_csrq date,c_zy char(16),c_xf float,c_bz char(40),primary key(c_xh));
```

查看主键约束：

```
Select table_name,constraint_name,constraint_type,status user_constraints where table_name='XSB';
```

2. 测试主键约束

插入一行数据：

```
Insert into xsb values ('101101','张 1','男',to_date('1-01-1981 00:00:00','DD-MM-YYYY HH24:MI:SS'),'计算机',50,null);
```

再插入同样的一行数据则会提示错误，因为主键冲突所以无法插入数据。

3. 修改主键约束

表的主键也是作为表的对象存在的，因此，同样可以对其进行修改。这其中包括，为表添加主键、删除主键、启用/禁用主键、重命名主键等。

（1）为表添加主键。

使用“Alter table 表名 add primary key (主键)”来添加主键。例：

```
Alter table xsb add primary key(c_xh);
```

（2）为表添加多列主键。

使用同样的方法用“Alter table 表名 add primary key (主键)”来添加多列主键。例：

```
Alter table cjb add primary key(c_xh,c_kch);
```

（3）删除主键。

使用“Alter table 表名 drop primary key”语句来删除数据库。例：

```
Alter table cjb drop primary key;
```

（4）启用/禁用主键。

我们可以通过“Alter table only_test disable primary key”语句来禁用主键。例：

```
alter table xsb disable primary key;
```

禁用主键之后可以重复插入同一个数据。

同样，可以用类似语句来启用主键。例：

```
alter table xsb enable primary key
```

2.3.2 外键约束的创建

外键实际是一种关联，描述了表之间的父子关系。即子表中的某条数据与父表中的某条数据有着依附关系。当父表中的某条数据被删除或进行更改时，会影响子表中的相应数据。例如，父表中的数据被删除，则子表中的相应数据也应该被删除；当父表中的数据进行更新，子表中的数据也应该做出适当的反应。

外键约束是建立在子表之上的，并要求子表的每条记录必须在父表中有且仅有一条记录与之对应。例如，某条 orders 的记录，没有对应的 customers 的信息是不允许的，亦即有订单没有客户是不允许的。

1. 建立外键

以 cjb 和 xsb 为例，创建表 cjb。

```
Create table cjb (c_xh char(10),c_kch char(10),c_cj float,primary key(c_xh,c_kch));
```

创建表 xsb：

```
Create table xsb(c_xh char(10),c_xm char(12),c_xb char(6),c_csrq date,c_zy char(16),c_xf float,c,bz char(40), primary key(c_xh));
```

在这两个表中建立了主键 c_xh 和 c_xh，现在建立 cjb 到 xsb 表的外键关联，代码如下：

```
Alter table cjb add constraint fk_xsb_cjb foreign key(c_xh) references xsb(c_xh);
```

查看外键的关联信息：

```
Select table_name,constraint_name constraint_type,r_constraint_name from user_constraints where table_name='cjb';
```

2. 验证外键约束的有效性

（1）xsb 为空时向子表 cjb 插入一条数据。

```
Insert into cjb values ('101102','102',81);
```

则会出现报错。

（2）在 xsb 表中添加信息。

```
Insert into xsb values ('101102','张 2','男',TO_DATE('2-01-1981 00:00:00','DD-MM- YYYY HH24:MI:SS'),'计算机',50,null);
```

再次往 cjb 中添加数据：

```
Insert into cjb values ('101102','102',81);
```

能够成功地添加数据。

（注：当修改子表时，若外键列被修改则会报错，修改非外键列的值则不影响修改结果）

3. 级联更新与级联删除

在具有外键的情形下，尝试修改主表中的数据并不一定能够成功。但是有时又的确有这种需求，即修改主表中的主键列的值。当然，子表中的数据也应该同时更新。对于主表中的记录删除亦是如此。但是因为外键约束，造成了两种操作都不能成功进行。这就是级联更新与级联删除问题的提出背景。

（1）级联更新。

级联更新是指当主表中的主键列进行修改时，子表的外键列也应该进行相应的修改。

（2）级联删除。

```
Alter table cjb add constraint fk_cjb_xsb foreign key (c_xh) references xsb(c_xh) on delete cascade
```

（如果提示错误，可先执行 Alter table cjb drop constraint fk_xsb_cjb;）

4. 修改外键属性

外键也是约束中的一种，因此可以像修改其他约束一样对其进行修改。修改外键的主要操作有：重命名、启用/禁用、修改、删除。

（1）重命名外键。

```
Alter table cjb rename constraint fk_cjb_xsb    to fk_cjbs;
```

查看 cjb 表的约束信息：

```
Select table_name,constraint_name constraint_type,r_constraint_name from user_constraints where table_name='cjb';
```

（2）禁用/启用外键。

外键可以被禁用，禁用外键以后向表中插入数据将不经过约束校验，禁用的约束可以再开启，在开启过程中将进行数据校验，校验不通过则该外键不能启动成功。

1）禁用外键。

```
Alter table cjb modify constraint fk_cjbs disable;
```

2）启用外键。

```
Alter table cjb modify constraint fk_cjbs enable;
```

在过程中可以查看外键使用情况

```
Select constraint_name,status from user_constraints where constraint_name='fk_cjbs';
```

（3）是否对已有数据进行校验。

当 cjb 建立在 xsb 的外键处于禁用时，在外键列数值不同的情况下启动外键，则会出现报错，启用外键会失败。

此时我们可以使用 novalidate 选项，使其不进行校验直接启用外键。

```
Alter table cjb modfy constraint fk_cjbs enable novalidate;
```

启用之后的约束依然对数据库起作用，若输入外键不一致，依然会抛出错误。

（4）删除约束。

删除约束的统一语法：

```
Alter table cjb drop constraint fk_cjbs;
```

2.3.3 唯一性约束的创建

主键列上的值都是唯一的，主键是记录唯一性的保证。但是，一个表只能有一个主键。很多时候，对于其他列同样要求列值唯一。例如，在用户表中，列 USER_ID 作为主键可以保证用户的唯一性，同时又要求其 E-Mail 地址唯一，防止多个用户同时使用同一邮箱。

所以，可以这样理解，主键设计为标识唯一一条记录，而唯一性约束则设计为保证列自身值的唯一性。

1. 创建唯一性约束

```
Create table kcb(c_kch char(10) unique,c_kcm char(20),c_kkxq decimal(16, 0),c_xs float,c_xf decimal(16, 0));
```

查看约束是否创建成功

```
Select table_name,constraint_name constraint_type,r_constraint_name from user_constraints where table_name='kcb';
```

2. 验证唯一性约束

向 kcb 表中添加课程号都为 101,则会返回违反唯一约束的错误信息。

```
Insert into kcb values ('101','课程 1',1,60,1);
Insert into kcb values ('101','课程 2',1,61,2);
```

（1）添加唯一性约束。

在建表之后我们可以添加唯一性约束

```
Alter table table_name add constraint up_name unique (name);
```

（说明：add constraint 添加约束，table_name 添加约束的表名，up_name 定义约束名称，小括号内 name 是约束添加的列名）

（2）删除唯一性约束。

```
Alter table kcb drop constraint  约束;
```

（3）重命名唯一性约束。

```
Alter table kcb rename constraint  旧约束名  to c_new;
```

（4）禁用/启用唯一性约束。

1）禁用约束。

Alter table 表名 modify constraint 约束名 disable;

禁用唯一约束之后可以不受约束限制。

2）启用约束。

Alter table 表名 modify constraint 约束名 enable

如果禁用约束之后插入过不合唯一约束的数据，则无法启用约束。

2.3.4 检查约束的创建

检查约束对列值进行限制，将表中的一列或多列限制在某个范围内。例如，在学生成绩表中，可能需要将学生单科成绩限制在 0～100 之内，超过 100 分的单科成绩将不能够录入。又如，在员工表中，可能需要限制经理级薪水不能超过 8000，主管级薪水不能超过 5000，普通员工薪水不能超过 4000。这些都可以通过检查约束来实现。

检查约束实际可以看作一个布尔表达式，该布尔表达式如果返回为真，则约束校验将通过，反之，约束校验将无法通过。

1. 创建检查约束

检查约束可以在创建表时进行创建，使用选项 check。

```
Create table cjb (c_xh char(10),c_kch char(10),c_cj float, check (c_cj<100));
```

当更新表中记录时，Oracle 都将计算 check 的布尔值。

2. 修改检查约束

检查约束可以像其他约束一样被修改，针对检查约束的操作包括添加、删除、重命名和禁用/启用。

（1）为 CJB 表添加约束。

```
alter table cjb add constraint cjb_cj check (length(c_cj)<=100);
```

删除检查约束：

```
alter table cjb drop constraint cjb_cj;
```

（2）重命名检查约束。

```
alter table cjb rename constraint cjb_cj to cj_cj;
```

（3）禁用/启用检查约束。

使用禁用/启用统一约束方法操作约束。

1）禁用约束语句：

```
alter table cjb disable constraint cj_cj;
```

2）启用约束语句：

```
alter table cjb enable constraint cj_cj;
```

2.3.5 非空约束的创建

非空约束是指该数据列的数据不能为空，但是可以为其设定一个默认值。

创建语句如下：

```
SQL> create table user2(user_id number primary key,user_name varchar2(20),status
 varchar2(3) default 'act');

Table created.
```

该语句中 status 为非空，我们为其设定了一个默认值“act”。

2.4 序列

序列（Sequence）像其他数据库对象（表、约束、视图、触发器等）一样，是实实在在的数据库对象。一旦创建，即可存在于数据库中，并可在适用场合进行调用。序列总是从指定整数开始，并按照特定步长进行累加，以获得新的整数。

2.4.1 创建序列

创建序列，应该使用 create sequence 命令。下文演示了如何创建一个用于生成表 xsb 主键 ID 的序列。

```
create sequence xsb_seq;
测试序列值：
select xsb_seq.nextval    from dual;
使用序列：
insert into kcb values ('101','课程 1',1, xsb_seq.nextval,1);
```

2.4.2 修改序列

通过 alter 命令可以修改序列属性。可修改的属性包括 minvalue、maxvalue、increment by、cache 和 cycle。

（1）修改 minvale 和 maxvalue。

minvalue 和 maxvalue 用于指定序列的最小值和最大值。序列最小值的意义在于限定 start with 和循环取值时的起始值；而最大值则用于限制序列所能达到的最大值。序列最小值不能大于序列的当前值。例如，尝试将序列 employee_start with 的最小值设置为 20，Oracle 将会抛出错误提示。

```
alter sequence xsb_seq minvalue 20;
alter sequence xsb_seq maxvalue 99999;
alter sequence xsb_seq nomaxvalue;
```

（2）修改 increment by。

increment by 相当于编程语言 for 循环中的步长。即每次使用 nextval 时，在当前值累加该步长来获得新值。序列的默认步长为 1，可以通过 alter 命令和 increment by 选项来修改序列步长。

```
alter sequence xsb_seq increment by 5;
select xsb_seq.currval from xsb;
select xsb_seq.nextval from xsb;
测试获得最大值。
```

（3）修改 cycle。

cycle 选项用于指定序列在获得最大值的下一个值时，从头开始获取。这里的“头”即为

minvalue 指定的值。为了说明 cycle 的功能及 start with 与 minvalue 的区别，首先创建该序列，并为各选项指定特定值。

```
create sequence xsb_seq start with 5 minvalue 1 maxvalue 30 increment by 1;
select xsb_seq.nextval from xsb;
alter sequencexsb_seq cycle;
alter sequence xsb_seq nocycle;
```

（4）修改 cache。

顾名思义，cache 是序列缓存，其实际意义为，每次利用 nextval 并非直接操作序列，而是一次性获取多个值的列表到缓存。使用 nextval 获得的值，实际是从缓存抓取。抓取的值，依赖于序列的 currval 和步长 increment by。默认缓存的大小为 20，可以通过 alter 命令修改缓存大小。可以通过如下步骤测试 cache 的存在。

```
alter sequence xsb_seq increment by 2;
alter sequence xsb _seq maxvalue 39;
alter sequence xsb _seq increment by 2;
alter sequence xsb _seq maxvalue 40;
alter sequence xsb _seq increment by 2;
```

2.5 同义词

Oracle 数据库中提供了同义词管理的功能。同义词是数据库方案对象的一个别名，经常用于简化对象访问和提高对象访问的安全性。在使用同义词时，Oracle 数据库将它翻译成对应方案对象的名字。与视图类似，同义词并不占用实际存储空间，只在数据字典中保存了同义词的定义。在 Oracle 数据库中的大部分数据库对象，如表、视图、同义词、序列、存储过程、包等等，数据库管理员都可以根据实际情况为它们定义同义词。

2.5.1 创建同义词

语法格式：

```
create public synonym cjb_new for cjb;
```

注：public 表示创建一个公用同义词，同义词的指向对象可以是表、视图、过程、函数、包和序列。

2.5.2 使用同义词

创建同义词后，数据库用户可以直接通过同义词名称访问该同义词的数据库对象，而不需要特别指出该对象的所属关系。

语法格式：

```
select * from cjb_new;
```

2.5.3 删除同义词

语法格式：

```
drop public synonym cjb_new;
```

2.6 索引

2.6.1 索引简介

在 Oracle 中，索引是一种供服务器在表中快速查找一个行的数据库结构。在数据库中建立索引主要有以下作用。

（1）快速存取数据。

（2）既可以改善数据库性能，又可以保证列值的唯一性。

（3）实现表与表之间的参照完整性。

（4）在使用 order by、group by 子句进行数据检索时，利用索引可以减少排序和分组的时间。

2.6.2 索引的分类

在关系数据库中，每一行都有一个行唯一标识 RowID。RowID 包括该行所在的条件、在文件中的块数和块中的行号。索引中包含一个索引条目，每一个索引条目都有一个键值和一个 RowID，其中键值可以是一列或者多列的组合。

（1）索引按存储方法分类，可以分为两类：B*树索引和位图索引。

1）B*树索引的存储结构类似书的索引结构，有分支和叶两种类型的存储数据块，分支块相当于书的大目录，叶块相当于索引到的具体的书页。Oracle 用 B*树机制存储索引条目，以保证用最短路径访问键值。默认情况下大多使用 B*树索引，该索引就是通常所见的唯一索引、逆序索引。

2）位图索引存储主要用于节省空间，减少 Oracle 对数据块的访问。它采用位图偏移方式来与表的行 ID 号对应，采用位图索引一般是重复值太多的表字段。位图索引之所以在实际密集型 OLTP（联机事物处理）中用得比较少，是因为 OLTP 会对表进行大量的删除、修改、新建操作。Oracle 每次进行操作都会对要操作的数据块加锁，以防止多人操作容易产生的数据库锁等待甚至死锁现象。在 OLAP（联机分析处理）中应用位图有优势，因为 OLAP 中大部分是对数据库的查询操作，而且一般采用数据仓库技术，所以大量数据采用位图索引节省空间比较明显。当创建表的命令中包含有唯一性关键字时，不能创建位图索引，创建全局分区索引时也不能用位图索引。

（2）索引按功能和索引对象分还有以下类型。

1）唯一索引意味着不会有两行记录相同的索引键值。唯一索引表中的记录没有 RowID，

不能再对其建立其他索引。在 Oracle 11g 中，要建立唯一索引，必须在表中设置主关键字，建立了唯一索引的表只按照该唯一索引结构排序。

2）非唯一索引不对索引列的值进行唯一性限制。

3）分区索引是指索引可以分散地存在于多个不同的表空间中，其优点是可以提高数据查询的效率。

4）未排序索引也称为正向索引。Oracle 11g 数据库中的行是按升序排序的，创建索引时不必指定对其排序而使用默认的顺序。

5）逆序索引也称反向索引。该索引同样保持列按顺序排列，但是颠倒已索引每列的字节。

6）基于函数的索引是指索引中的一列或者多列是一个函数或者表达式，索引根据函数或表达式计算索引列的值。可以将基于函数的索引建立创建成位图索引。另外，按照索引所包含的列数可以把索引分为单列索引和复合索引。索引列只有一列的索引为单列索引，对多列同时索引称为复合索引。

2.6.3 创建索引

（1）首先创建表。

```
SQL> create table dex (id int, sex char(1),name char(10));

Table created.

SQL>
```

（2）插入数据。

```
SQL> begin
  2  for i in 1..1000
  3  loop
  4  insert into dex values (i,'M','chongqing');
  5  end loop;
  6  commit;
  7  end;
  8  /

PL/SQL procedure successfully completed.

SQL>
```

（3）查看表记录。

```
SQL> select * from dex;
        ID S NAME
---------- - ----------
       991 M chongqing
       992 M chongqing
       993 M chongqing
       994 M chongqing
       995 M chongqing
       996 M chongqing
       997 M chongqing
       998 M chongqing
       999 M chongqing
      1000 M chongqing

1000 rows selected.
```

（4）创建索引。

```
SQL> create index dex_idx1 on dex(id);

Index created.
```

（5）查看创建的表和索引。

```
SQL> select object_name,object_type from user_objects;

OBJECT_NAME
--------------------------------------------------------------------------------

OBJECT_TYPE
-------------------
DEX_IDX1
INDEX
```

（6）使用索引查看记录。

```
SQL> select * from dex where id>30 and id<35;

        ID S NAME
---------- - ----------
        31 M chongqing
        32 M chongqing
        33 M chongqing
        34 M chongqing
```

（7）修改索引。

```
SQL> alter index dex_idx1 rename to dex_id;

Index altered.
```

（8）查询索引。

```
SQL> select object_name,object_type from user_objects;
```

（9）删除索引。

```
SQL> drop index dex_id;

Index dropped.
```

2.7 更新数据库

Oracle 中可以利用 DML 更新数据。其 DML 语句与其他数据库的 SQL 语法完全一致，都是遵守了工业标准。与查询操作不同，更新数据将导致数据库状态的变化，因此，Oracle 同样提供了提交与回滚操作来保证数据库状态的一致性。

2.7.1 插入记录

插入数据即向数据表中插入新的记录，插入数据应该使用 insert 命令。插入数据的主要途径包括：通过指定各列的值直接插入、通过子查询插入、通过视图插入等。对于通过视图插入的方式，大多数应该使用 instead of 触发器来进行处理。

（1）用 Insert 语句向表中插入数据（3 个表 3 个语句）。

```
insert into xsb values ('101101','张 1','男',to_date('1-01-1981 00:00:00','DD-MM-YYYY HH24:MI:SS'),'计算机',50,null);
insert into kcb values ('101','课程 1',1,60,1);
insert into cjb values ('101101','101',80);
```

（2）利用子查询批量插入数据。

Oracle 可以利用子查询向表中批量插入数据。此时的 SQL 语句除了包含 insert into 命令之外，还应该包含一个查询语句。

```
insert into cjb select c_xh,c_kch,c_cj from t_cjb where c_xh>=101101;
```

注：在插入之前首先创建一个和 cjb 表内容一样的表，命名为 t_cjb:

```
create table t_cjb (c_xh char(10),c_kch char(10),c_cj float);
```

2.7.2 删除记录

数据删除的目标是数据表中的记录，而不是针对列来进行的。删除数据应该使用 delete 命令或者 truncate 命令。其中 delete 命令的作用目标是表中的某些记录，而 truncate 命令的作用目标是整个数据表。

像 update 命令一样，delete 命令经常与 where 子句一起出现，以删除数据表中的某些数据。

```
delete from cjb p where exists(select 1 from t_cjb e where e.c_xh = p.c_xh);
```

其中，delete 命令用于删除表中数据；from cjb p 用于指定删除的目标表为 cjb，并指定该表的别名为 p；where exists(select 1 from t_cjb e where e.c_xh = p.c_xh)用于指定删除记录的过滤条件——在表 t_cjb 中存在着一条记录，该记录的 c_xh 列值等于表 cjb 的当前记录的 c_xh 列值；该删除语句用于保证表 cjb 中，所有的姓名不再存在于表 t_cjb 中。

2.7.3 修改记录

像其他数据库一样，Oracle 使用 update 命令来修改数据。update 修改数据一般有以下几种情况：直接修改单列的值、直接修改多列的值、利用 where 子句限制修改范围和利用视图修改数据。利用视图修改数据往往需要利用 instead of 触发器实现。

（1）利用 update 修改单列的值。

```
update cjb set c_cj = '60';
select * from cjb;
```

（2）利用 update 修改多列的值。

update 命令既可以修改单列值，也可以同时修改多列的值。例如，有时为了合并两个表的

数据，需要为其中一个的主键 c_xh 添加一个基数，以避免两个表中主键的重复。此时，需要修改表中所有 c_xh 的值。以表 cjb 为例，在修改列 c_xh 的值的同时，也可以修改 c_cj 列的值。

```
update cjb set c_xh = (20+ c_xh ), c_cj = '61';
```

（3）利用 where 子句限制修改范围

where 子句是 update 命令最常用的子句。不使用 where 子句的 update 命令是不安全的。因为不使用 where 子句将一次性修改表中所有记录，这将带来极大的安全隐患。为了将表 cjb 中 c_ch 大于 101103 的 c_cj 列修改为“79”，则可以利用如下所示的 SQL 语句。

```
update cjb set c_cj = '79' where c_xh >101103;
```