第 6 章　数据库对象的创建与管理

创建完数据库，并完成数据库存储结构设置后，就应该根据设计在数据库中创建各种数据库对象，并应用数据库对象。为了实现数据操纵和查询，需要在数据库中创建表，并设置表中的各种约束，以保证数据的一致性；为了提高数据的查询效率，需要在表上创建适当的索引；为了对巨型表进行高效的管理，需要将表进行分区；为了简化复杂查询，需要创建视图；为了自动产生表中数据编号（流水号），需要创建序列对象。本章将介绍人力资源管理系统数据库表、索引、视图、序列等数据库对象的创建，同时介绍这些数据库对象的应用与管理。

6.1　Oracle 数据库对象概述

6.1.1　模式的概念

在 Oracle 数据库中，数据库对象是以模式为单位进行组织和管理的。所谓模式是指一系列逻辑数据结构或对象的集合。

模式与用户相对应，一个模式只能被一个数据库用户所拥有，并且模式的名称与这个用户的名称相同。通常情况下，用户所创建的数据库对象都保存在与自己同名的模式中。在同一模式中数据库对象的名称必须唯一，而在不同模式中的数据库对象可以同名。例如，数据库用户 usera 和 userb 都在数据库中创建一个名为 test 的表。因为用户 usera 和 userb 分别对应 usera 和 userb 模式，故 usera 用户创建的 test 表放在 usera 的模式中，而 userb 用户创建的 test 表放在 userb 模式中。在默认情况下，用户引用的对象是与自己同名模式中的对象，如果要引用其他模式中的对象，则需要在该对象名之前指明对象所属模式。例如，如果用户 usera 要引用 userb 模式中的 test 表，则必须使用 userb.test 形式来表示；如果用户 usera 要引用 usera 模式中的 test 表，则可以使用 usera.test 形式或者直接引用 test 都可以。

Oracle 数据库中并不是所有的对象都是模式对象。表、索引、索引化表、分区表、物化视图、视图、数据库链接、序列、同义词、PL/SQL 包、存储函数与存储过程、Java 类与其他 Java 资源等属于特定的模式，称为模式对象。表空间、用户、角色、目录、概要文件及上下文等数据库对象不属于任何模式，称为非模式对象。

6.1.2　案例数据库模式的创建

为了方便案例数据库中数据库对象的应用与管理，在 HUMAN_RESOURCE 数据库中创建一个名为 human 的用户，以该用户登录数据库并创建各种数据库对象，这些对象都将成为 human 模式的对象。

创建 human 用户，并为其授权，方法为：

```
SQL>CONNECT sys/tiger @HUMAN_RESOURCE AS SYSDBA
SQL>CREATE USER human IDENTIFIED BY human DEFAULT TABLESPACE USERS;
SQL>GRANT CONNECT,RESOURCE,CREATE VIEW TO human;
```

以后，就可以以 human 用户登录 HUMAN_RESOURCE 数据库，并创建各种数据库对象。

```
SQL>CONNECT human/human @HUMAN_RESOURCE
```

6.2 表的创建与管理

在 Oracle 数据库中，数据是以二维表的方式来组织的。表由行和列组成，一行对应一个实体的实例，又称为记录、元组；一列对应一个实体的属性，又称为字段；每一行在某一个列的取值，称为数据单元，又称为数据项、属性值、字段值等。表是数据库中最基本的模式对象，所有的数据操作都是围绕表进行的。因此表的管理是数据管理的基础，应用系统对数据库的使用主要是通过对表的使用实现的。

6.2.1 利用 CREATE TABLE 语句创建表

可以使用 CREATE TABLE 语句创建表，语法为：

```
CREATE TABLE table_name(
column_name datatype [column_level_constraint]
[,column_name datatype [column_level_constraint]…]
[,table_level_constraint])
[parameter_list];
```

创建表时，通常需要设置表的 4 个属性，包括表名、数据类型、约束及表的参数设置。

1．表名

创建表时，必须为表起一个在当前模式中唯一的名称，该名称必须是合法标识符，长度为 1～30 字节，并且以字母开头，可以包含字母（A～Z，a～z）、数字（0～9）、下画线（_）、美元符号（$）和#。表名称不能是 Oracle 数据库的保留字。

2．数据类型

Oracle 数据库中常用的数据类型见表 6-1。

表 6-1 Oracle 数据库中常用的数据类型

数据类型名称	说　明
CHAR(n)	存储固定长度的字符串，长度以字节为单位，默认和最小字符数为 1，最大字符数为 2000
VARCHAR2(n)	存储可变长度的字符串，长度以字节为单位，最小字符数是 1，最大字符数是 4000
NUMBER (p, s)	可以存储 0、正数和负数。p 表示数值的总位数（精度），取值范围为 1～38；s 表示刻度，取值为–84～127
DATE	用于存储日期和时间。可以存储的日期范围为公元前 4712 年 1 月 1 日到公元后 9999 年 12 月 31 日，占据 7 字节的空间，由世纪、年、月、日、时、分、秒组成
TIMESTAMP(n)	表示时间戳，是 DATE 数据类型的扩展，允许存储小数形式的秒值。p 表示秒的小数位数，取值范围为 0～9，默认值为 6
CLOB	用于存储单字节或多字节的大型字符串对象，支持使用数据库字符集的定长或变长字符。在 Oracle 11g 中 CLOB 类型最大存储容量为 128TB
BLOB	用于存储大型的、未被结构化的变长的二进制数据（如二进制文件、图片文件、音频和视频等非文本文件）。在 Oracle 11g 中 BLOB 类型最大存储容量为 128TB
BFILE	用于存储指向二进制格式文件的定位器，该二进制文件保存在数据库外部的操作系统中。在 Oracle 11g 中 BFILE 文件最大容量为 128TB，不能通过数据库操作修改 BFILE 定位器所指向的文件
RAW(n)	用于存储变长的二进制数据，n 表示数据长度，取值范围为 1～2000 字节
LONG RAW	用于存储变长的二进制数据，最大存储数据量为 2GB。Oracle 建议使用 BLOB 类型代替 LONG RAW 类型
ROWID	行标识符，表示表中行的物理地址的伪列类型

3．约束

约束是在表中定义的用于维护数据完整性的一些规则，用于规范表中列的取值。在 Oracle 数据库中，可以通过为表设置约束来防止无效数据插入到表中。

在 Oracle 数据库中，约束分为主键约束（PRIMARY KEY）、唯一性约束（UNIQUE）、检查约束（CHECK）、外键约束（FOREIGN KEY）和非空约束（NOT NULL）5 种。

（1）主键约束：作用在一列或多列上，用于唯一标识一条记录。主键约束的列或列的组合的取值唯一且不能为空。一个表中只能定义一个主键约束。

（2）唯一性约束：作用在一列或多列上，列或列的组合的取值唯一，但可以为空。一个表中可以定义多个唯一性约束。

（3）检查约束：作用在一列或多列上，限制列或列组合的取值。

（4）外键约束：作用在一列或多列上，体现了表与表之间的关系。外键约束列的参照列为主表的主键约束列或唯一性约束列。外键约束列的取值受限于主表参照列的取值或为空值。

（5）非空约束：作用于某一个列上，该列取值不能为空。

在 Oracle 数据库中，约束的定义有两种形式：列级约束和表级约束。列级约束是在定义表中列的同时定义约束，表级约束是表中列都定义完后再单独定义约束。

定义列级约束的语法为：

```
[CONSTRAINT constraint_name] constraint_type [conditioin];
```

表级约束通常用于对多个列一起进行约束，定义表级约束的语法为：

```
[CONSTRAINT constraint_name] constraint_type([column1_name,column2_
name,…]|[condition]);
```

非空约束只能采用列级约束，其他约束既可以采用列级约束，也可以采用表级约束。表中的约束可以在定义表时创建，也可以在表创建后添加。

4．表参数

在创建表时可以通过参数指定表的存储位置、存储空间分配等。

- TABLESPACE：用于指定表存储的表空间。若不指定，则默认为当前用户的默认表空间。
- STORAGE：用于设置表的存储参数。若不指定，则继承表空间的存储参数设置。
- LOGGING、NOLOGGING：指明表的创建过程是否写入重做日志文件。默认为 LOGGING。
- CACHE、NOCACHE：指明表中数据是否缓存。默认为 CACHE。
- NOPARALLEL、PARALLEL：指明是否允许并行创建表以及随后对表中数据进行并行操作。默认为 NOPARALLEL。

【例 6-1】 在案例数据库的 human 模式下创建 emp 表。

```
SQL>CREATE TABLE emp (
emp_id    NUMBER(6,0) PRIMARY KEY,
first_name  VARCHAR2(20),
last_name  VARCHAR2(25),
email     VARCHAR2(25) UNIQUE,
job_id     VARCHAR2(10),
salary     NUMBER(8,2),
commission_pct  NUMBER(2,2),
manager_id      NUMBER(6,0),
department_id    NUMBER(4,0),
```

```
CONSTRAINT c_salary CHECK(salary>0)
)
TABLESPACE USERS;
```

6.2.2 案例数据库中表的创建

根据人力资源管理系统中表的设计，以 human 用户登录 HUMAN_RESOURCE 数据库创建表。

```
SQL>CONNECT human/human @HUMAN_RESOURCE
```

（1）创建 REGIONS 表。

```
SQL>CREATE TABLE regions(
    region_id NUMBER PRIMARY KEY,
    region_name VARCHAR2(25)
    )
    TABLESPACE USERS;
```

（2）创建 COUNTRIES 表。

```
SQL>CREATE TABLE countries(
    country_id CHAR(2) PRIMARY KEY,
    country_name VARCHAR2(40),
    region_id NUMBER REFERENCES regions(region_id)
    )
    TABLESPACE USERS;
```

（3）创建 LOCATIONS 表。

```
SQL>CREATE TABLE locations(
    location_id NUMBER(4) PRIMARY KEY,
    street_address VARCHAR2(40),
    postal_code VARCHAR2(12),
    city VARCHAR2(30) NOT NULL,
    state_province VARCHAR2(25),
    country_id CHAR(2)  REFERENCES countries(country_id)
    )
    TABLESPACE USERS;
```

（4）创建 DEPARTMENTS 表。

```
SQL>CREATE TABLE departments(
    department_id NUMBER(4) PRIMARY KEY,
    department_name VARCHAR2(30) NOT NULL,
    manager_id NUMBER(6),
    location_id NUMBER(4) REFERENCES locations (location_id)
    )
    TABLESPACE USERS;
```

（5）创建 JOBS 表。

```
SQL>CREATE TABLE jobs(
    job_id VARCHAR2(10) PRIMARY KEY,
    job_title VARCHAR2(35) NOT NULL,
    min_salary NUMBER(6),
    max_salary NUMBER(6),
    )
```

```
    TABLESPACE USERS;
```

（6）创建 EMPLOYEES 表。

```
SQL>CREATE TABLE employees(
    employee_id NUMBER(6) PRIMARY KEY,
    first_name VARCHAR2(20),
    last_name VARCHAR2(25) NOT NULL,
    email VARCHAR2(25) NOT NULL UNIQUE,
    phone_number VARCHAR2(20),
    hire_date DATE NOT NULL,
    job_id VARCHAR2(10) NOT NULL REFERENCES jobs (job_id),
    salary NUMBER(8,2) CHECK (salary > 0),
    commission_pct NUMBER(2,2),
    manager_id NUMBER(6),
    department_id NUMBER(4) REFERENCES departments(department_id)
    )
    TABLESPACE USERS;
```

（7）创建 JOB_HISTORY 表。

```
SQL>CREATE TABLE job_history(
    employee_id NUMBER(6) NOT NULL REFERENCES employees(employee_id),
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    job_id VARCHAR2(10) NOT NULL REFERENCES jobs(job_id),
    department_id NUMBER(4) REFERENCES departments(department_id),
    CONSTRAINT jhist_date_interval CHECK (end_date > start_date),
    CONSTRAINT jhist_emp_id_st_date_pk PRIMARY KEY (employee_id, start_date)
    )
    TABLESPACE USERS;
```

（8）创建 SAL_GRADES 表。

```
SQL>CREATE TABLE sal_grades(
    grade  NUMBER PRIMARY KEY,
    min_salary  NUMBER(8,2),
    max_salary  NUMBER(8,2)
    )
    TABLESPACE USERS;
```

（9）创建 USERS 表。

```
SQL>CREATE TABLE users(
    user_id NUMBER(2)PRIMARY KEY,
    user_name CHAR(20),
    password  VARCHAR2(20) NOT NULL
    )
    TABLESPACE USERS;
```

6.2.3 利用子查询创建表

在 Oracle 数据库中，可以利用子查询创建表。基本语法为：

```
CREATE TABLE table_name
(column_name [column_level_constraint]
[,column_name [column_level_constraint]…]
```

```
[,table_level_constraint])
[parameter_list]
AS subquery;
```

利用子查询创建表需要注意以下事项。

- 通过该方法创建表时，可以修改表中列的名称，但是不能修改列的数据类型和长度。
- 源表中的约束条件和列的默认值都不会复制到新表中。
- 子查询中不能包含 LOB 类型和 LONG 类型列。
- 当子查询条件为真时，新表中包含查询到的数据；当子查询条件为假时，则创建一个空表。

【例 6-2】 创建一个表，保存工资高于 15000 元的员工的员工号、员工姓名和部门号。

```
SQL>CREATE TABLE sub_emp11(empno,fname,lname,deptno)
    AS
    SELECT employee_id,first_name,last_name,department_id FROM employees
    WHERE salary>15000;
```

【例 6-3】 创建一个包含员工号、员工 email、员工工资及部门号信息的空表，其中员工号为主键、email 唯一。

```
SQL>CREATE TABLE sub_emp2(employee_id PRIMARY KEY,email UNIQUE,
salary,department_id)
    AS
    SELECT emp_id,email,salary,department_id FROM emp WHERE 1=2;
```

6.2.4 修改表

表创建后，可以使用 ALTER TABLE 语句添加列、修改列的数据类型与名称、将列设置为不可用、删除列、修改表名称等维护操作。

ALTER TABLE 语句的基本语法为：

```
ALTER TABLE table
[ADD new_column datatype [constraint] ]|
[MODIFY  column datatype ]|
[SET UNUSED COLUMN column]|
[SET UNUSED COLUMNS(column1,column2…)]|
[DROP   COLUMN column]|
[DROP  (column1,column2…)]|
[DROP   UNUSED  COLUMNS]|
[RENAME  COLUMN oldname TO newname]|
[RENAME TO new_table]
```

【例 6-4】 为 emp 表添加 phone_number 和 hiredate 两列。

```
SQL>ALTER TABLE emp
ADD(phone_number VARCHAR2(20),hiredate DATE DEFAULT SYSDATE NOT NULL);
```

【例 6-5】 修改 emp 表中 first_name 和 phone_number 两列的数据类型。

```
SQL>ALTER TABLE emp MODIFY first_name CHAR(25);
SQL>ALTER TABLE emp MODIFY phone_number CHAR(30);
```

【例 6-6】 修改 emp 表中 hiredate 列的名称为 hire_date。

```
SQL>ALTER TABLE emp RENAME COLUMN hiredate TO hire_date;
```

【例 6-7】 删除 emp 表中的 emp_id，phone_number，hire_date 三列。

```
SQL>ALTER TABLE emp DROP COLUMN emp_id CASCADE CONSTRAINTS;
SQL>ALTER TABLE emp DROP (phone_number,hire_date);
```

【例 6-8】 将 emp 表中的 first_name，last_name，salary 列设置为 UNUSED 状态。

```
SQL>ALTER TABLE emp SET UNUSED COLUMN salary;
SQL>ALTER TABLE emp SET UNUSED (first_name,last_name);
SQL>ALTER TABLE emp DROP UNUSED COLUMNS;
```

【例 6-9】 为 emp 表重新命名为 new_emp。

```
SQL>ALTER TABLE emp RENAME TO new_emp;
```

6.2.5 修改约束

表创建后，可以根据需要使用 ALTER TABLE 语句为表添加、修改、删除约束。语法为：

```
ALTER TABLE table_name
[ADD [CONSTRAINT constraint_name] constraint_type(column1,…)
[condition]]|
[MODIFY column [NOT NULL]|[NULL]]|
[DROP [CONSTRAINT constraint_name]|[PRIMARY KEY]|[UNIQUE(column)]]
```

下面以 player 表为例说明添加各种约束的方法。其中 player 定义如下：

```
SQL>CREATE TABLE player(
    ID    NUMBER(6),
    sno   NUMBER(6),
    sname  VARCHAR2(10),
    sage   NUMBER(6,2),
    resume VARCHAR2(1000)
);
```

1）添加主键约束

```
SQL>ALTER TABLE player ADD CONSTRAINT P_PK PRIMARY KEY(ID);
```

2）添加唯一性约束

```
SQL>ALTER TABLE player ADD CONSTRAINT P_UK UNIQUE(sname);
```

3）添加检查约束

```
SQL>ALTER TABLE player ADD CONSTRAINT P_CK  CHECK(sage BETWEEN 20 AND 30);
```

4）添加外键约束

```
SQL>ALTER TABLE player
    ADD CONSTRAINT P_FK FOREIGN KEY(sno) REFERENCES student(sno) ON DELETE
    CASCADE;
```

5）添加空/非空约束

为表列添加空/非空约束时必须使用 MODIFY 子句代替 ADD 子句。

```
SQL>ALTER TABLE player MODIFY resume NOT NULL;
SQL>ALTER TABLE player MODIFY resume  NULL;
```

6）删除约束

如果要删除已经定义的约束，可以使用 ALTER TABLE…DROP 语句。可以通过直接指定约束的名称来删除约束，或指定约束的内容来删除约束。

（1）删除指定类型的约束：

```
SQL>ALTER TABLE player DROP UNIQUE(sname);
```

（2）删除指定名称的约束：

```
SQL>ALTER TABLE player DROP CONSTRAINT P_CK;
```

（3）如果要在删除约束的同时，删除引用该约束的其他约束（如子表的 FOREIGN KEY 约束引用了主表的 PRIMARY KEY 约束），则需要在 ALTER TABLE…DORP 语句中指定 CASCADE 关键字。

```
SQL>ALTER TABLE player DROP CONSTRAINT P_PK CASCADE;
```

6.2.6 查询表

可以通过查询数据字典视图 DBA_TABLES、ALL_TABLES、USER_TABLES、DBA_TAB_COLUMNS、ALL_TAB_COLUMNS、USER_TAB_COLUMNS 获取表及其列的信息。

【例 6-10】 查询当前用户拥有的所有表的信息。

```
SQL>SELECT table_name,tablespace_name,status,logging FROM user_tables;
TABLE_NAME   TABLESPACE_NAME   STATUS    LOG
------------------------ --------------------------------
REGIONS          USERS              VALID    YES
COUNTRIES        USERS              VALID    YES
LOCATIONS        USERS              VALID    YES
DEPARTMENTS      USERS              VALID    YES
…
```

可以通过查询数据字典视图 DBA_CONSTRAINTS、ALL_CONSTRAINTS、USER_CONSTRAINTS、DBA_CONS_COLUMNS、USER_CONS_COLUMNS、USER_CONS_COLUMNS 等获取中存在约束信息

【例 6-11】 查询 employees 表中所有约束的名称与类型。

```
SQL>SELECT constraint_name,constraint_type,status FROM user_constraints
WHERE table_name='EMPLOYEES';
CONSTRAINT_NAME              C           STATUS
---------------------------------------- ---------
SYS_C0010833                 C           ENABLED
SYS_C0010834                 P           ENABLED
SYS_C0010835                 U           ENABLED
…
```

6.2.7 删除表

如果表不再需要，可以使用 DROP TABLE 语句将其删除。如果要删除的表中包含有被其他表外键引用的主键列或唯一性约束列，并且希望在删除该表的同时删除其他表中相关的外键约束，则需要使用 CASCADE CONSTRAINTS 子句。

【例 6-12】 删除 player 表。

```
SQL>DROP TABLE player CASCADE CONSTRAINTS;
```

在 Oracle 11g 数据库中，使用 DROP TABLE 语句删除一个表时，通常并不立即回收该表的空间，只是将表及其关联对象的信息重命名后写入一个称为“回收站”（RECYCLEBIN）的逻辑容器中，从而可以实现表的闪回删除（FLASHBACK DROP）操作。如果要回收该表的存储空间，可以清空“回收站”（PURGE RECYCLEBIN）或在 DROP TABLE 语句中使用 PURGE 语句。例如：

```
SQL>DROP TABLE player CASCADE CONSTRAINTS PURGE;
```

6.3 索引的创建与管理

在人力资源管理系统中，随着表中数据量的增加，数据查询效率逐渐降低，为此需要在表中创建适当索引，以提高数据检索的效率。

6.3.1 索引概述

索引是一种提高数据检索效率的数据库对象，能够为数据的查询提供快捷的存取路径，减少磁盘 I/O。虽然索引是基于表而建立的，但索引并不依赖于表。索引由系统自动维护和使用，不需要用户参与。

1．索引类型

Oracle 数据库为了提高数据检索性能，提供了多种类型的索引，以满足不同的应用需求。

（1）B-树索引：按平衡树结构组织的索引，是最常用的索引，也是默认创建的索引类型。B-树索引占用空间多，适合索引值取值范围广（基数大）、重复率低的应用。

（2）位图索引：按位图结构组织的索引，适合索引值取值范围小（基数小），重复率高的应用。

（3）函数索引：基于包含索引列的函数或表达式创建的索引（索引值为计算后的值）。

（4）唯一性索引与非唯一性索引：唯一性索引是索引值不重复的索引，非唯一性索引是索引值可以重复的索引。在默认情况下，Oracle 创建的索引是非唯一性索引。当在表中定义主键约束或唯一性约束时，Oracle 会自动在相应列上创建唯一性索引。

（5）单列索引与复合索引：索引可以创建在一个列上，也可以创建在多个列上。创建在一个列上的索引称为单列索引，创建在多个列上的索引称为复合索引。

2．索引使用原则

由于索引作为一个独立的数据库对象存在，占用存储空间，并且需要系统进行维护，因此索引的使用需要遵循下列原则。

（1）导入数据后再创建索引。

（2）在适当的表和列上创建适当的索引。如果经常查询的记录数目少于表中记录总数的5%时就应当创建索引；如果经常进行连接，应该在连接列上建立索引；对于取值范围很大的列应当创建 B 树索引，而对于取值范围很小的列应当创建位图索引。

（3）合理设置复合索引中列的顺序，应将频繁使用的列放在其他列的前面。

（4）限制表中索引的数目。表中索引数目越多，查询速度越快，但表的更新速度越慢。

（5）选择存储索引的表空间。在默认情况下，索引与表存储在同一表空间中。

6.3.2 使用 CREATE INDEX 语句创建索引

可以使用 CREATE INDEX 语句创建索引，语法为：

```
CREATE [UNIQUE][BITMAP]INDEX index ON [schema.]table(column[ASC|DESC][,…])
[REVERSE] [parameter_list];
```

其中：

- UNIQUE 表示建立唯一性索引。
- BITMAP 表示建立位图索引。
- ASC|DESC 用于指定索引值的排列顺序。ASC 表示按升序排列，DESC 表示按降序排列，

默认值为 ASC。

- REVERSE 表示建立反键索引。
- parameter_list 用于指定索引的存放位置、存储空间分配和数据块参数设置。

【例 6-13】 在 emp 表的 last_name 列上创建一个非唯一性索引。

```
SQL>CREATE INDEX emp_lname_indx ON emp(last_name) TABLESPACE indx;
```

【例 6-14】 在 emp 表的 email 列上创建一个唯一性索引。

```
SQL>CREATE UNIQUE INDEX emp_email_indx ON emp(email) TABLESPACE indx;
```

【例 6-15】 在 emp 表的 job_id 列上创建一个位图索引。

```
SQL>CREATE BITMAP INDEX emp_job_indx ON emp(job_id) TABLESPACE indx;
```

【例 6-16】 基于 emp 表的 first_name 列创建一个函数索引。

```
SQL>CREATE INDEX emp_fname_indx ON emp(UPPER(first_name)) TABLESPACE indx;
```

6.3.3 案例数据库中索引的创建

根据人力资源管理系统中索引的设计，以 human 用户登录 HUMAN_RESOURCE 数据库创建索引。

```
SQL>CONNECT human/human @HUMAN_RESOURCE
```

（1）在 employees 表的 department_id 列上创建名为“EMP_DEPARTMENT_INDX”的索引。

```
SQL>CREATE INDEX emp_department_indx ON employees (department_id) TABLESPACE
    indx;
```

（2）在 employees 表的 job_id 列上创建名为“EMP_JOB_INDX”的平衡树索引。

```
SQL>CREATE INDEX emp_job_indx ON employees(job_id) TABLESPACE indx;
```

（3）在 employees 表的 manager_id 列上创建名为“EMP_MANAGER_INDX”的平衡树索引。

```
SQL>CREATE INDEX emp_manager_indx ON employees (manager_id) TABLESPACE indx;
```

（4）在 employees 表的 last_name，first_name 列上创建名为“EMP_NAME_INDX”的索引。

```
SQL>CREATE INDEX emp_name_indx ON employees (last_name, first_name) TABLESPACE
    indx;
```

（5）在 departments 表 location_id 列上创建名为“DEPT_LOCATION_INDX”的平衡树索引。

```
SQL>CREATE INDEX dept_location_indx ON departments (location_id) TABLESPACE
    indx;
```

（6）在 job_history 表的 job_id 列上创建名为“JHIST_JOB_INDX”的平衡树索引。

```
SQL>CREATE INDEX jhist_job_indx ON job_history (job_id) TABLESPACE indx;
```

（7）在 job_history 表的 employee_id 列上创建名为“JHIST_EMP_INDX”的平衡树索引。

```
SQL>CREATE INDEX jhist_emp_indx ON job_history (employee_id) TABLESPACE indx;
```

（8）在 job_history 表的 department_id 列上创建名为“JHIST_DEPT_INDX”的平衡树索引。

```
SQL>CREATE INDEX jhist_dept_indx ON job_history (department_id) TABLESPACE
    indx;
```

（9）在 locations 表的 city 列上创建名为“LOC_CITY_INDX”的平衡树索引。

```
SQL>CREATE INDEX loc_city_indx ON locations (city) TABLESPACE indx;
```

（10）在 locations 表的 country_id 列上创建名为“LOC_COUNTRY_INDX”的平衡树索引。

```
SQL>CREATE INDEX loc_country_indx ON locations (country_id) TABLESPACE indx;
```

6.3.4 删除索引

如果索引不再使用，或者由于移动了表数据而导致索引失效，或者由于索引中包含损坏的

数据块、过多的存储碎片等，可以考虑删除索引。

如果索引是通过 CREATE INDEX 语句创建的，可以使用 DROP INDEX 语句删除索引。

【例 6-17】 删除 emp 表中的 emp_new_lname_indx 索引。

```
SQL>DROP INDEX emp_new_lname_indx;
```

如果索引是定义约束时自动建立的，则在禁用约束或删除约束时会自动删除对应的索引。

6.3.5 查询索引

可以查询数据字典视图 DBA_INDEXES、ALL_INDEXES、USER_INDEXES、DBA_IND_COLUMNS、ALL_ IND_COLUMNS、USER_ IND_COLUMNS 获取索引信息。

【例 6-18】 查询 employees 表中所有索引的名称与类型。

```
SQL>SELECT index_name,index_type FROM dba_indexes WHERE table_name=
    'EMPLOYEES';
INDEX_NAME                INDEX_TYPE
----------------------------------------- ----------------------
EMP_NAME_INDX             NORMAL
EMP_MANAGER_INDX          NORMAL
EMP_JOB_INDX              NORMAL
…
```

6.4 视图的创建与管理

6.4.1 视图概述

视图是从一个或多个表或视图中提取出来的数据的一种逻辑表现形式。在数据库中只有视图的定义，而没有实际对应“表”的存在，因此视图是一个“虚”表。创建视图时数据来源的表统称为基表，视图实际是基表中数据的多样性表现，可以为不同的用户创建不同的视图。通过视图的使用可以提高数据的安全性，隐藏数据的复杂性，简化查询语句，分离应用程序与基础表，保存复杂查询等。

根据视图定义时复杂程度的不同，视图分为简单视图和复杂视图两类。在简单视图定义中，数据来源于一个基表，不包含函数、分组等，可以直接进行 DML 操作。在复杂视图定义中，数据来源于一个或多个基表，可以包含连接、函数、分组、伪列、表达式等元素，能否直接进行 DML 操作取决于视图的具体定义。

当对视图进行操作时，系统会根据视图的定义临时生成数据。系统会自动将所有对视图的操作转换为对基表的操作。

6.4.2 使用 CREATE VIEW 语句创建视图

可以使用 CREATE VIEW 语句创建视图，语法为：

```
CREATE [OR REPLACE] [FORCE| NOFORCE] view[(alias[,alias]…)]
AS
Subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

其中：

- FORCE：不管基表是否存在都创建视图。
- NOFORCE：仅当基表存在时才创建视图（默认）。
- WITH CHECK OPTION：指明对视图操作时，必须满足子查询中的约束条件。
- WITH READ ONLY：指明该视图为只读视图，不能修改。

【例 6-19】 创建一个视图，包含员工号、员工名、工资和部门号等员工基本信息。

```
SQL>CREATE VIEW emp_base_info_view (empno,fename,lname,sal,deptno)
    AS
    SELECT employee_id,first_name,last_name,salary,department_id FROM
    employees;
```

【例 6-20】 创建一个视图，包含各个部门中不同职位的员工人数、平均工资。

```
SQL>CREATE VIEW dept_job_stat_view
    AS
    SELECT department_id,job_id,count(*) num,avg(salary) avgsal
    FROM employees GROUP BY department_id,job_id;
```

如果子查询中包含条件，创建视图时可以使用 WITH CHECK OPTION 选项。

【例 6-21】 创建一个视图，包含工资大于 2000 元的员工的员工号、员工名及员工的年工资。

```
SQL>CREATE VIEW emp_sal_view
    AS
    SELECT employee_id,first_name,last_name,salary*12 year_salary FROM
    employees  WHERE salary>2000 WITH CHECK OPTION;
```

【例 6-22】 创建一个包含员工号、员工名、员工工资以及员工所在部门名的只读视图。

```
SQL>CREATE VIEW emp_dept_view
    AS
    SELECT employee_id,first_name,last_name,salary,department_name FROM
    employees e,departments d WHERE e.department_id=d.department_id WITH READ
    ONLY;
```

如果在创建视图时基表不存在，则可以强制创建视图，但创建该视图时提示存在编译错误，当基表创建后，对视图重新编译后，视图可以正常使用。

【例 6-23】 基于当前还不存在的 test 表创建一个视图。

```
SQL>CREATE FORCE VIEW test_view
    AS
    SELECT * FROM test;
```

6.4.3 案例数据库中视图的创建

根据人力资源管理系统中视图的设计，以 human 用户登录 HUMAN_RESOURCE 数据库创建视图。

```
SQL>CONNECT human/human @HUMAN_RESOURCE
```

（1）创建一个名为“EMP_DETAILS_VIEW”的视图，用于员工信息综合查询，包括员工编号、员工名、工资、奖金、职位编号、职位名称、部门编号、部门名称、部门所在地信息、国家信息、区域信息等。

```
SQL>CREATE OR REPLACE VIEW emp_details_view(
    employee_id,job_id,manager_id,department_id,location_id,country_id,fir
    st_name,last_name,salary,commission_pct,department_name,job_title,city,
    state_province,country_name,region_name)
```

```
    AS
    SELECT .employee_id,e.job_id,e.manager_id,e.department_id
    d.location_id,l.country_id,e.first_name,e.last_name,e.salary,
    e.commission_pct,d.department_name,j.job_title,l.city,
    l.state_province,c.country_name,r.region_name FROM employees e,
    departments d,jobs j,locations l,countries c,regions r WHERE e.department_id
    = d.department_id  AND d.location_id = l.location_id AND l.country_id =
    c.country_id AND c.region_id = r.region_id AND j.job_id = e.job_id
    WITH READ ONLY;
```

（2）创建一个名为“DEPT_STAT_VIEW”的视图，包含部门号、部门人数、部门平均工资、部门最高工资、部门最低工资以及部门工资总和。

```
SQL>CREATE OR REPLACE VIEW dept_stat_view
    AS
    SELECT department_id,count(*) num,avg(salary) avgsal,max(salary)
    maxsal,min(salary) minsal,sum(salary) totalsal FROM employees GROUP BY
    department_id;
```

6.4.4 视图操作的限制

视图创建后，就可以对视图进行操作，包括数据查询、DML 操作（数据的插入、删除、修改）等。因为视图是“虚表”，因此对视图的操作最终会转换为对基本表的操作。

对视图的查询与对标准表查询一样，但是对视图执行 DML 操作时需要注意，如果视图定义包括下列任何一项，则不可直接对视图进行插入、删除和修改等操作，需要通过触发器来实现：

- 集合操作符（UNION，UNION ALL，MINUS，INTERSECT）；
- 聚集函数（SUM，AVG 等）；
- GROUP BY，CONNECT BY 或 START WITH 子句；
- DISTINCT 操作符；
- 由表达式定义的列；
- 伪列 ROWNUM；
- （部分）连接操作。

例如，可以直接修改视图 emp_base_info_view，但不能修改 dept_job_stat_view、emp_sal_view 和 emp_dept_view。

```
SQL>UPDATE emp_base_info_view SET sal=sal+100;
```

如果在视图的定义中包含了 WITH CHECK OPTION 子句，则对视图操作时必须满足相应的约束条件。

6.4.5 修改视图定义

可以采用 CREATE OR REPLACE VIEW 语句修改视图的定义，其实质是删除原视图并重建该视图，但是会保留该视图上授予的各种权限。

【例 6-24】 修改视图 dept_job_stat_view，增加各个部门中不同职位的工资总和。

```
SQL>CREATE OR REPLACE VIEW dept_job_stat_view
    AS
    SELECT department_id,job_id,count(*) num,avg(salary) avgsal,
    sum(salary) total FROM employees GROUP BY department_id,job_id;
```

6.4.6 删除视图

可以使用 DROP VIEW 语句删除视图。删除视图后，该视图的定义也从数据字典中删除，同时该视图上的权限被回收，但是对数据库表没有任何影响。

【例 6-25】 删除视图 dept_job_stat_view。

```
SQL>DROP VIEW dept_job_stat_view;
```

6.4.7 查询视图信息

可以通过查询数据字典视图 DBA_VIEWS，ALL_VIEWS，USER_VIEWS 获取视图信息。

【例 6-26】 查询当前用户所有视图名称及视图定义信息。

```
SQL>SELECT view_name,text FROM user_views;
VIEW_NAME          TEXT
-------------------- ---------------------------------------------
EMP_BASE_INFO_VIEW      SELECT employee_id,first_name,last_name,salary,
                        department_id  FROM employees
EMP_SAL_VIEW            SELECT employee_id,first_name,last_name,
                        salary*12 year_salary FROM employees WHERE salary>2000
WITH CHECK OPTION
DEPT_JOB_STAT_VIEW      SELECT department_id,job_id,count(*) num,
                        avg(salary) avgsal, sum(salary) total FROM employees
                        GROUP BY
                        department_id,job_id
…
```

6.5 序　列

6.5.1 序列的概念

序列用于产生唯一序号的数据库对象，可以为多个数据库用户依次生成不重复的连续整数，通常使用序列自动生成表中的主键值。序列产生的数字最大长度可达到 38 位十进制数。序列不占用实际的存储空间，在数据字典中只存储序列的定义描述。

序列具有下列特点：

- 可以为表中的记录自动产生唯一序号；
- 由用户创建并且可以被多个用户共享；
- 典型应用是生成主键值，用于标识记录的唯一性；
- 允许同时生成多个序列号，而每一个序列号是唯一的；
- 使用缓存可以加速序列的访问速度。

6.5.2 使用 CREATE SEQUENCE 语句创建序列

可以使用 CREATE SEQUENCE 语句创建序列，语法为：

```
CREATE SEQUENCE sequence
[START WITH integer]
[INCREMENT BY integer]
[MAXVALUE integer|NOMAXVALUE]
```

```
[MINVALUE integer|NOMINVALUE]
[CYCLE|NOCYCLE]
[CACHE integer|NOCACHE];
```

其中：

- START WITH：设置序列初始值，默认值为 1。
- INCREMENT BY：设置相邻两个元素之间的差值，即步长，默认值为 1。
- MAXVALUE：设置序列最大值。
- NOMAXVALUE：默认情况下，递增序列的最大值为 10^{28}–1，递减序列的最大值为–1。
- MINVALUE：设置序列最小值。
- NOMINVALUE：默认情况下，递增序列的最小值为 1，递减序列的最小值为–(10^{27} –1)。
- CYCLE：当序列达到其最大值或最小值后，开始新的循环。
- NOCYCLE：当序列达到其最大值或最小值后，序列不再生成值。默认选项。
- CACHE：设置 Oracle 服务器预先分配并保留在内存中的序列值的个数，默认为 20。
- NOCACHE：不缓存序列值。

【例 6-27】 创建一个序列，用于产生学生号码。

```
SQL>CREATE SEQUENCE student_seq
    START WITH 1000
    INCREMENT BY 2
    MAXVALUE 1000000
    CACHE 10;
```

6.5.3 案例数据库中序列的创建

根据人力资源管理系统中序列的设计，以 human 用户登录 HUMAN_RESOURCE 数据库创建序列。

```
SQL>CONNECT human/human @HUMAN_RESOURCE
```

（1）创建一个名为“EMPLOYEES_SEQ”的序列，用于产生员工编号，起始值为100，步长为 1，不缓存，不循环。

```
SQL>CREATE SEQUENCE employees_seq START WITH 100 INCREMENT BY 1 NOCACHE NOCYCLE;
```

（2）创建一个名为“DEPARTMENTS_SEQ”的序列，用于产生部门编号，起始值为 10，步长为 10，最大值为 9990，不缓存，不循环。

```
SQL>CREATE SEQUENCE departments_seq
    START WITH 10 INCREMENT BY 10 MAXVALUE 9990 NOCACHE NOCYCLE;
```

（3）创建一个名为“LOCATIONS_SEQ”的序列，用于产生位置编号，起始值为1000，步长为 100，最大值为 9990，不缓存，不循环。

```
SQL>CREATE SEQUENCE locations_seq
    START WITH 1000 INCREMENT BY 100 MAXVALUE 9900 NOCACHE NOCYCLE;
```

6.5.4 序列的使用

序列具有 CURRVAL 和 NEXTVAL 两个伪列。CURRVAL 返回序列的当前值，NEXTVAL 在序列中产生新值并返回此值。CURRVAL 和 NEXTVAL 都返回 NUMBER 类型值。可以通过 sequence_name.CURRVAL 和 sequence_name.NEXTVAL 形式来应用序列。

可以在下列语句中使用序列的 NEXTVAL 和 CURRVAL 伪列：

- SELECT 语句的目标列中；
- INSERT 语句的子查询的目标列中；
- INSERT 语句的 VALUES 子句中；
- UPDATE 语句的 SET 子句中。

在下列语句中不允许使用序列的 NEXTVAL 和 CURRVAL 伪列：

- 对视图查询的 SELECT 目标列中；
- 使用了 DISTINCT 关键字的 SELECT 语句中；
- SELECT 语句中使用了 GROUP BY、HAVING 或 ORDER BY 子句时；
- 在 SELECT、DELETE 或 UPDATE 语句的子查询中；
- 在 CREATE TABLE 或 ALTER TABLE 语句中的默认值表达式中。

【例 6-28】 利用 student_seq 序列产生学号并插入到 student 表中。

```
SQL>CREATE TABLE students(sno NUMBER PRIMARY KEY,sname CHAR(20));
SQL>INSERT INTO students(sno,sname) VALUES(student_seq.nextval,'Joan');
SQL>INSERT INTO students(sno,sname) VALUES(student_seq.nextval,'Mary');
SQL>SELECT * FROM students;
SNO   SNAME
---------- -----------
1002  Joan
1004  Mary

SQL>SELECT students_seq.currval FROM dual;
CURRVAL
---------------
1004
```

6.5.5 修改序列

序列创建完成后，可以使用 ALTER SEQUENCE 语句修改序列。除了不能修改序列的 START WITH 参数外，可以对序列其他参数进行修改。如果要修改 MAXVALUE 参数，需要保证修改后的最大值大于序列的当前值（CURRVAL）。此外，序列的修改只影响以后生成的序列值。

【例 6-29】 修改序列 student_seq 的步长为 1，缓存值的个数为 5。

```
SQL>ALTER SEQUENCE student_seq INCREMENT BY 1 CACHE 5;
```

6.5.6 查看序列信息

可以查询数据字典视图 DBA_SEQUENCES、ALL_SEQUENCES、USER_SEQUENCES 获取序列信息。

【例 6-30】 查询序列 student_seq 的信息。

```
SQL>SELECT sequence_name,min_value,max_value,increment_by,cycle_flag,
    cache_size FROM user_sequences WHERE sequence_name='STUDENT_SEQ';
SEQUENCE_NAME   MIN_VALUE  MAX_VALUE   INCREMENT_BY  C   CACHE_SIZE
--------------------------- -------------------- -------------------
STUDENT_SEQ     1           1000000      1             N    5
```

6.5.7 删除序列

当一个序列不再需要时，可以使用 DROP SEQUENCE 语句删除序列。删除序列时，系统将序列的定义从数据字典中删除，对于之前序列的应用没有任何影响。

【例 6-31】 删除序列 student_seq。

```
SQL>DROP SEQUENCE student_seq;
```

6.6 分区表与分区索引

6.6.1 分区的概念

在 Oracle 数据库中，当表中数据量达到 GB 级别时，为了方便对表中数据的管理，可以考虑将表进行分区。所谓分区就是将一个巨型表分成若干个独立的组成部分进行存储和管理，每个相对小的、可以独立管理的部分，称为原来表的分区。表分区后，可以对表的分区进行独立的存取和控制。每个分区都具有相同的逻辑属性，但物理属性可以不同。如具有相同列、相同数据类型、相同约束等，但可以具有不同的存储参数、位于不同的表空间中。

对巨型表进行分区具有下列优点：

- 提高数据的安全性，一个分区的损坏不影响其他分区中数据的正常使用；
- 将表的各个分区存储在不同磁盘上，提高数据的并行操作能力；
- 简化数据的管理，可以将某些分区设置为不可用状态，某些分区设置为可用状态，某些分区设置为只读状态，某些分区设置为读/写状态；
- 操作的透明性，对表进行分区并不影响操作数据的 SQL 语句。

6.6.2 分区方法

在 Oracle 11g 数据库中，对表进行分区有多种方法。

（1）范围分区：根据分区列值的范围对表进行分区，每条记录根据其分区列值所在的范围决定存储到哪个分区中。范围分区是最常用的分区方法，特别适合根据日期进行分区的情况。

（2）列表分区：如果分区列的值不能划分范围（非数值类型或日期类型），同时分区列的取值是一个包含少数值的集合，可以采用列表分区，将特定分区列值的记录保存到特定分区中。

（3）散列分区：又称 HASH 分区，是采用基于分区列值的 HASH 算法，将数据均匀分布到指定的分区中。一个记录到底分布到哪个分区是由 HASH 函数决定的。

（4）复合分区：结合两种基本分区方法，先采用一个分区方法对表或索引进行分区，然后再采用另一个分区方法将分区再分成若干个子分区。每个分区的子分区都是数据的一个逻辑子集。复合分区包括范围-范围复合分区、范围-散列复合分区、范围-列表复合分区、列表-范围复合分区、列表-散列复合分区、列表-列表复合分区等多种分区方法。

在 Oracle 11g 数据库中，分区技术得到进一步的增强，引入了间隔分区、引用分区、基于虚拟列分区以及系统分区（System Partitioning）等多种分区方法。

6.6.3 创建分区表

1．创建范围分区表

使用带 PARTITION BY RANGE 子句的 CREATE TABLE 语句创建范围分区表，基本语法为：

```
CREATE TABLE table(…)
PARTITION BY RANGE (column1[,column2,…])
( PARTITION partition1 VALUES LESS THAN(literal|MAXVALUE) [TABLESPACE
tablespace]
[,PARTITION partition2 VALUES LESS THAN(literal|MAXVALUE) [TABLESPACE
tablespace],…]
)
…
```

其中：

- PARTITION BY RANGE：指明采用范围分区方法。
- column：分区列，可以是单列分区，也可以是多列分区。
- PARTITION partition1：设置分区名称。
- VALUES LESS THAN：设置分区列值的上界。
- TABLESPACE：设置分区对应的表空间。

【例 6-32】 创建一个分区表，将学生信息根据其出生日期进行分区，将 1980 年 1 月 1 日前出生的学生信息保存在 ORCLTBS1 表空间中，将 1980 年 1 月 1 日到 1990 年 1 月 1 日出生的学生信息保存在 ORCLTBS2 表空间中，将其他学生信息保存在 ORCLTBS3 表空间中。

```
SQL>CREATE TABLE student_range(
    sno NUMBER(6) PRIMARY KEY,
    sname VARCHAR2(10),
    sage int,
    birthday DATE
    )
    PARTITION BY RANGE(birthday)
    (   PARTITION p1 VALUES LESS THAN(TO_DATE('1980-1-1', 'YYYY-MM-DD'))
            TABLESPACE ORCLTBS1,
        PARTITION p2 VALUES LESS THAN (TO_DATE('1990-1-1', 'YYYY-MM-DD'))
            TABLESPACE ORCLTBS2,
        PARTITION p3 VALUES LESS THAN(MAXVALUE)TABLESPACE ORCLTBS3
);
```

创建分区表后，通过修改分区表的各个分区所在的表空间的状态，可以实现对表分区的不同操作。可以将表的部分分区设置为脱机状态或只读状态，但不影响其他分区的使用。

【例 6-33】 将分区表的不同分区设置为不同状态，如脱机状态、只读状态，验证分区表的可用性。

```
SQL>INSERT INTO student_range VALUES(1,'Joan',40,to_date('1973-3-1',
'yyyy-mm-dd'));
SQL>INSERT INTO student_range VALUES(2,'Tom',30,to_date('1983-3-1',
'yyyy-mm-dd'));
SQL>INSERT INTO student_range VALUES(3,'Mary',20,to_date('1993-3-1',
'yyyy-mm-dd'));
SQL>SELECT * FROM student_range;
SNO   SNAME  SAGE   BIRTHDAY
---------- ------------ ----------
1     Joan      40     01-3月 -73
2     Tom       30     01-3月 -83
```