十一、DDL与数据库对象

# 表的创建及管理

所谓数据对象就是DDL定义的所有操作，包括表、视图、索引、序列、约束等，这些都属于对象的操作，所以表的建立就是对象的建立，而对象的操作主要分为以下三种语法：

|  |  |
| --- | --- |
| 创建对象 | CREATE 对象名称…; |
| 删除对象 | DROP 对象名称…; |
| 修改对象 | ALTER 对象名称…; |

常见数据对象：

|  |  |
| --- | --- |
| **数据对象** | **描述** |
| 表 | 存放数据的基本数据库对象，由行（记录）和列（字段）组成 |
| 约束条件 | 执行数据校验，保证数据完整性的系列规则 |
| 视图 | 表中数据的逻辑显示 |
| 索引 | 根据表中指定的字段建立起来的顺序，用于提高查询性能 |
| 序列 | 一组有规律的整数值 |
| 同义词 | 对象的别名 |

## Oracle数据对象的命名规则：

* 必须以字母开头;
* 可以包含字母、数字、\_、$、和#;
* 同一方案（用户）下的对象不能重名;
* 不能使用Oracle 的保留字;

## 常用的数据类型

数据库中的每张表都是由若干个字段（即列名）组成的，每个字段都有一个与之对应的数据类型。在Oracle中，常用的数据类型有以下几种：

|  |  |  |  |
| --- | --- | --- | --- |
| **NO.** | **数据类型** | **关键字** | **描述** |
| **1** | 字符型 | char(n) | 定长字符类型，最大长度2000B,缺省长度为1B，查询效率较高，长度不够自动补齐； |
| **2** | 字符型 | nchar(n) | 基于NLS国家字符集的字符型，最大长度2000B，缺省为1个字符 |
| **3** | 字符串型 | VARCHAR2(n) | 可变字符串，存放少量文本，n为字符串最大长度，不补齐长度，节省空间； |
| **2** | 整数 | NUMBER(n) | 表示最多存放n位整数，有时可以用INT代替 |
| **3** | 浮点数 | NUMBER(n,m) | 表示最多存放n位浮点数，m为小数位，有时可以用FLOAT代替 |
| **4** | 日期 | DATE | 用以存放日期和时间 |
| **5** | 大文本 | CLOB | 用以存放海量文本数据，最大4G |
| **6** | 大对象 | BLOB | 用以存放二进制数据，最大4G |
| **7** | 精度日期 | timestamp | 精度较高 |

**注**：

* 一般情况下，不推荐使用BLOB存放数据，原因一、造成数据库过于庞大；原因二、读取不方便；
* 对于大数据，一般是将其索引存放在数据库，然后通过索引访问相应数据；
* 只有在处于安全性考虑时，才会将大数据放入数据库；

## 表的创建语法

|  |
| --- |
| **CREATE TABLE 表名称(**  **字段1 数据类型 [DEFAULT 默认值],**  **字段2 数据类型 [DEFAULT 默认值],**  **字段3 数据类型 [DEFAULT 默认值],**  **.**  **.**  **.**  **字段n 数据类型 [DEFAULT 默认值]**  **);** |

1. 创建一个member表，包含姓名、年龄、生日、具体内容字段；

|  |
| --- |
| CREATE TABLE member(  name VARCHAR2(50) DEFAULT '无名氏',  age NUMBER(3) ,  birthday DATE DEFAULT SYSDATE ,  content CLOB  ); |

**注**：

* 表的创建属于数据库对象的创建，故要用CREATE语法；
* CREATE TABLE为DDL语句，一经执行不可撤消；

## 表的复制

语法：

|  |
| --- |
| **CREATE TABLE 表名称 AS 子查询；** |

1. 创建一个表，复制emp表中所有20部门的员工信息；

|  |
| --- |
| CREATE TABLE emp20 AS  (SELECT \* FROM emp  WHERE deptno=20); |

1. 创建一个表，复制emp表的表结构而不要数据；

思路：要复制表结构，只需将查询条件写成一个永远不可能满足的条件即可；

|  |
| --- |
| CREATE TABLE empnull AS  (SELECT \* FROM emp  WHERE 1=2); |

**注**：以上操作为Oracle数据库所特有的操作，其他数据库不一定支持；

## 表的重命名

数据字典：在Oracle中，所有数据都是以文件形式保存的，而所有的内容都会在数据字典中注册，在Oracle中提供了三种数据字典：dba\_,user\_；所谓修改表名称，即相当于修改数据字典中一条数据；

|  |
| --- |
| **SELECT \* FROM tab;** |

tab也是一种数据字典；

查询user\_tables可以看到当前用户的所有表的信息：

|  |
| --- |
| **SELECT \* FROM user\_tables;** |

表的重命名语法：

|  |
| --- |
| **RENAME 旧表名称 TO 新表名称；** |

1. 将member表名称改为person；

|  |
| --- |
| RENAME member TO person; |
|  |

**注**：

* 此操作也属于Oracle所独有的；
* 此操作也可修改其他数据库对象（视图、序列、同义词等）的名称;
* 执行重命名操作的必须是对象的所有者;
* 为DDL语句，一经执行不可撤消;

## 截断表

之前对表中数据的DELETE操作删除数据后可以通过回滚操作恢复，并不会直接将表中所占用的资源释放，且记录日志，可以恢复数据，但速度较慢；而表的截断操作可以直接释放表所占用的全部资源（例如：表空间、索引等）直接清空表中数据，且不记录日志，同时删除的数据不可以通过回滚操作恢复，故数据无法恢复，但速度较快。

语法如下：

|  |
| --- |
| **TRUNCATE TABLE 表名称；** |

**注**：TRUNCATE为DDL语句，一经执行不可撤消；

1. 截断person;

|  |
| --- |
| TRUNCATE TABLE person; |
|  |

## 表的删除

删除表即删除一个数据对象，其操作用DROP操作，语法如下；

|  |
| --- |
| **DROP TABLE 表名称；** |

1. 删除person表；

|  |
| --- |
| DROP TABLE person; |
|  |

**注**：

* 表中所有数据将被删除；
* 此前未完成的事务将被提交；
* 所有相关的索引被删除；
* 为DDL语句，一经执行不可撤消；

删除表后，通过tab查看，发现原先删除的表没有了，但有几条特殊名称的记录，这就涉及到了闪回技术；

## 闪回技术（Oracle10g新特性）

为了避免用户误删数据，提供了回收站的功能，即用户删除的表通常默认放在回收站中，用户可以在回收站中恢复误删的表数据；

### 8.1查看回收站

语法：

|  |
| --- |
| **SHOW RECYCLEBIN;** |
|  |

### 8.2恢复回收站中的表

语法：

|  |
| --- |
| **FLASHBACK TABLE 表名称 TO BEFORE DROP;** |

1. 恢复emp20表；

|  |
| --- |
| FLASHBACK TABLE emp20 BEFORE DROP; |
|  |

### 8.3删除回收站中的表

语法：

|  |
| --- |
| **PURGE TABLE 表名称；** |

1. 删除empnull表；

|  |
| --- |
| PURGE TABLE empnull; |
|  |

### 8.4清空回收站

语法：

|  |
| --- |
| **PURGE RECYCLEBIN;** |
|  |

### 8.5不经回收站直接删除表

语法：

|  |
| --- |
| **DROP TABLE 表名称 PURGE;** |

1. 不经过回收站删除emp20表；

|  |
| --- |
| DROP TABLE emp20 PURGE; |
|  |

## 修改表结构

修改表结构可以是增加字段、删除字段、修改字段，属于对数据对象的修改，故用ALTER语句；首先创建一个member表，包含name,age字段；

|  |
| --- |
| CREATE TABLE member(  name VARCHAR2(30) DEFAULT '无名氏',  age NUMBER(3)  ); |

### 9.1增加字段

语法：

|  |
| --- |
| **ALTER TABLE 表名称 ADD(**  **字段1 数据类型 [DEFAULT 默认值],**  **字段2 数据类型 [DEFAULT 默认值],**  **.**  **.**  **.**  **字段n 数据类型 [DEFAULT 默认值]**  **);** |

**注：此语句也可以用来在建表之后添加约束；**

1. 为member表增加birthday,content字段；

|  |
| --- |
| ALTER TABLE member ADD(  birthday DATE DEFAULT SYSDATE,  content CLOB  ); |
|  |

### 9.2修改字段

语法：

|  |
| --- |
| **ALTER TABLE 表名称 MODIFY(**  **字段1 数据类型 [DEFAULT 默认值],**  **字段2 数据类型 [DEFAULT 默认值],**  **.**  **.**  **.**  **字段n 数据类型 [DEFAULT 默认值]**  **);** |

**注：此语句也可以用来在建表之后添加约束；**

1. 修改member表中的name字段，将其默认值设置为’未命名’；

|  |
| --- |
| ALTER TABLE member MODIFY(  name VARCHAR2(30) DEFAULT '未命名'  ); |
|  |

**注**：当创建表时，对于未设定默认值的字段一律设为null,对于有默认值的字段一律设为默认值；

**注**：虽然Oracle数据库中给出了修改表结构的语法，但这些语法是不推荐使用的。在一般情况下，表结构一旦确定就不再修改，若需要修改则重建表结构，防止造成数据混乱或丢失；在IBM DB2数据库中，不允许修改表结构；

## 思考题

要求建立一个nation表，表中有name字段，表内保存四条记录：中国、巴西、美国、荷兰，要求通过查询实现如下操作效果：

* 中国 美国
* 中国 巴西
* 中国 荷兰
* 美国 巴西
* 美国 中国
* 美国 荷兰

… …

依次类推，要求建表并实现上述查询；

思路：本思考题在于规范数据库脚本的创建步骤，该脚本应该以\*.sql结尾，步骤如下：

1. 删除同名数据表；
2. 编写建表语句；
3. 添加测试数据；
4. 提交事务；

|  |
| --- |
| 1.删除同名数据表 |
| DROP TABLE nation PURGE; |
| 2.编写建表语句 |
| CREATE TABLE nation(  name VARCHAR2(30)  ); |
|  |
| 3.添加测试数据 |
| INSERT INTO nation(name) VALUES('中国');  INSERT INTO nation(name) VALUES('美国');  INSERT INTO nation(name) VALUES('荷兰');  INSERT INTO nation(name) VALUES('巴西'); |
| 4.提交事务 |
| COMMIT; |

建表完成后进行查询操作：此查询操作为自身关联；

|  |
| --- |
| SELECT n1.name,n2.name  FROM nation n1,nation n2  WHERE n1.name!=n2.name; |
|  |

# 数据字典

## 1. Oracle中的数据表：

1）用户定义的表

* 用户自己创建并维护的一组表；
* 包含了用户所需的信息；
* 数据字典表；

2）由Oracle数据库自动创建并维护的一组表:数据字典

* 包含数据库信息；

## 2. 数据字典的概念：

* 数据字典是Oracle数据库的核心，用于描述数据库及其所有对象。
* 数据字典由一系列只读的表和视图组成，这些表和视图属sys用户拥有，由Oracle server负责维护，用户可以通过select语句进行访问。

## 3. 数据字典的分类：

* dba - 所有方案包含的对象信息
* all - 用户可以访问的对象信息
* user - 用户方案的对象信息

示例：

|  |
| --- |
| --查看当前用户拥有的所有表的名字  SELECT table\_name FROM user\_tables;  --查看当前用户可以访问的所有表的名字  SELECT table\_name FROM all\_tables;  --查看当前用户拥有的所有对象的类型  SELECT DISTINCT OBJECT\_TYPE FROM user\_objects;  --查看所有用户拥有的所有对象的类型  SELECT table\_name FROM dba\_tables; |

## 4. 数据字典的内容：

* 数据库的物理和逻辑结构：
* 对象的定义和空间分配
* 完整性约束条件
* 用户
* 角色
* 权限
* 审计记录：跟踪记录用户操作；

# 约束

## 数据的完整性

数据的完整性用于确保数据库遵从一定的商业和逻辑规则；在Oracle中，数据的完整性可以使用约束、触发器、应用程序（过程、函数）三种方法来实现；在这三种方法中，因为约束易于维护，并且具有最好的性能，所以作为维护数据完整性的首选；

## 约束的分类

约束是在表上强制执行的数据校验规则，用于保护数据的完整性。约束可以对数据表中的数据进行过滤，从而保证数据的合法性，约束可以分为五种：

* 域完整性约束

• not null（非空约束）

• check（检查约束）

* 实体完整性约束

• unique（唯一约束）

• primary key（主键约束）

* 参照完整性约束

• foreign key（外键约束）

## 约束的创建

* 约束可以在创建表时创建，也可以在表创建完成之后添加；
* Oracle使用SYS\_Cn格式命名约束，也可以由用户命名；
* 可以在表级或字段级定义；

## 非空约束（NOT NULL KEY）：NK

当创建数据表时，如果不希望某个字段数据为null的话，就可以为该字段自定非空约束；非空约束只能在字段级定义；

示例：

|  |
| --- |
| CREATE TABLE student(  --定义无名非空约束  sid NUMBER(3) **NOT NULL** ,  name VARCHAR2(20) ,  --定义有名非空约束  BIRTH DATE **CONSTRAINT STUDENT\_BIRTH\_NN NOT NULL**  ); |

1. 创建member表，包含mid,name字段，且name字段非空；

|  |
| --- |
| DROP TABLE member PURGE;  CREATE TABLE member(  mid NUMBER(3),  name VARCHAR2(30) NOT NULL  ); |
|  |

现在向表中插入合法数据和非法数据：

|  |
| --- |
| INSERT INTO member(mid,name) VALUES(1,'张三');  INSERT INTO member(mid,name) VALUES(null,'李四');  INSERT INTO member(name) VALUES('王五');  INSERT INTO member(mid,name) VALUES(2,null);  INSERT INTO member(mid,name) VALUES(3,); |
|  |

## 唯一约束（UNIQUE KEY）：UK

唯一性约束特点：

* 唯一性约束用于确保所在的字段（或字段组合）不出现重复值;
* 唯一性约束的字段允许出现空值;
* Oracle会自动为唯一性约束创建对应的唯一性索引;
* 唯一性约束既可以在字段级定义、也可以在表级定义;

示例：

|  |
| --- |
| CREATE TABLE STUDENT(  sid NUMBER(3) **UNIQUE**,  name VARCHAR2(20)  ); |
| CREATE TABLE STUDENT(  sid NUMBER(3) ,  name VARCHAR2(20) ,  **CONSTRAINT student\_sid\_un UNIQUE(sid)**  ); |

1. 创建member表，包含mid,name,email字段，且name字段非空,email字段加唯一约束；

|  |
| --- |
| DROP TABLE member PURGE;  CREATE TABLE member(  mid NUMBER(3),  name VARCHAR2(30) NOT NULL,  email VARCHAR2(30) UNIQUE  ); |

先表中插入合法数据和非法数据：

|  |
| --- |
| INSERT INTO member(mid,name,email) VALUES(1,'张三','zhangsan@163.com');  INSERT INTO member(mid,name,email) VALUES(2,'李四','李四@163.com');  INSERT INTO member(mid,name,email) VALUES(1,'王五','zhangsan@163.com'); |
|  |

上述例子可以看出，违反约束时，约束的名字由系统默认指定，但这个约束的名字指代并不明确，可以通过CONSTRAINT关键字为约束指定名称，规范名称是‘约束类型\_字段名’，故上述语句可作如下修改：

|  |
| --- |
| DROP TABLE member PURGE;  CREATE TABLE member(  mid NUMBER(3),  name VARCHAR2(30) NOT NULL,  email VARCHAR2(30),  CONSTRAINT UK\_EMAIL UNIQUE(email)  ); |

向表中插入合法数据和非法数据：

|  |
| --- |
| INSERT INTO member(mid,name,email) VALUES(1,'张三','zhangsan@163.com');  INSERT INTO member(mid,name,email) VALUES(2,'李四','李四@163.com');  INSERT INTO member(mid,name,email) VALUES(1,'王五','zhangsan@163.com'); |
|  |

可以看出，错误提示变得更加清晰；

## 主键约束（PRIMARY KEY）：PK

主键约束特点：

* 主键用于唯一标识表中的某一行记录，功能上相当于非空且唯一;
* 一个表中只允许一个主键，主键可以是单个字段或多字段的组合;
* Oracle会自动为主键字段创建对应的唯一性索引;
* 主键约束既可以在字段级定义、也可以在表级定义;

示例：

|  |
| --- |
| CREATE TABLE student(  sid NUMBER(3) **PRIMARY KEY**,  name VARCHAR2(20)  ); |
| CREATE TABLE student(  sid NUMBER(3) ,  name VARCHAR2(20) ,  **CONSTRAINT STUDENT\_SID\_PK PRIMARY KEY(SID)**  ); |

1. 创建member表，包含mid,name,email字段，且mid字段添加主键约束,name字段非空,email字段加唯一约束；

|  |
| --- |
| DROP TABLE member PURGE;  CREATE TABLE member(  mid NUMBER(3) PRIMARY KEY,  name VARCHAR2(30) NOT NULL,  email VARCHAR2(30),  CONSTRAINT UK\_EMAIL UNIQUE(email)  ); |

向表中插入合法数据和非法数据：

|  |
| --- |
| INSERT INTO member(mid,name,email) VALUES(1,'张三','zhangsan@163.com');  INSERT INTO member(mid,name,email) VALUES(2,'李四','李四@163.com');  INSERT INTO member(mid,name,email) VALUES(1,'王五','王五@163.com'); |
|  |

从上表可以看出，错误指代并不明确，故需要通过CONSTRAINT重新命名；

|  |
| --- |
| DROP TABLE member PURGE;  CREATE TABLE member(  mid NUMBER(3) ,  name VARCHAR2(30) NOT NULL,  email VARCHAR2(30),  CONSTRAINT UK\_EMAIL UNIQUE(email),  CONSTRAINT PK\_MID PRIMARY KEY(mid)  ); |
|  |

联合主键：

* 由多个字段组合而成的主键也称联合主键；
* 联合主键中每一个字段都不能为空；
* 联合主键字段组合的值不能出现重复；
* 联合主键只能定义为表级约束；

示例：

|  |
| --- |
| CREATE TABLE record(  student\_id NUMBER(3),  subject\_id VARCHAR2(20),  record NUMBER(3),  **CONSTRAINT record\_stuid\_subid\_pk PRIMARY KEY(student\_id, subject\_id)**  ); |

1. 创建member表，包含mid,name,email字段，指定mid,name为主键,name字段非空,email字段加唯一约束；

|  |
| --- |
| DROP TABLE member PURGE;  CREATE TABLE member(  mid NUMBER(3) ,  name VARCHAR2(30) NOT NULL,  email VARCHAR2(30),  CONSTRAINT UK\_EMAIL UNIQUE(email),  CONSTRAINT PK\_MID\_NAME PRIMARY KEY(mid,name)  ); |
|  |

## 检查约束（CHECK KEY）：CK

用以限定字段数据的取值范围，过滤输入数据；

检查约束的特点：

* 定义每一行（的指定字段）都必须满足的条件；
* 以条件表达式的形式给出数据需要符合的条件；
* 条件表达式中不允许出现如下内容：
  + currval, nextval, level, rownum等伪列
  + sysdate, uid, user, userenv等函数
  + 对其它字段值的引用
* 只能在字段级定义；

1. 创建member表，包含mid,name,sex,age字段，且mid为主键,name,sex字段非空,限制数据字段在合理的范围之内；

|  |
| --- |
| DROP TABLE member PURGE;  CREATE TABLE member(  mid NUMBER(3) ,  name VARCHAR2(30) NOT NULL,  sex VARCHAR2(30) NOT NULL,  age NUMBER(3),  CONSTRAINT PK\_MID\_NAME PRIMARY KEY(mid),  CONSTRAINT CK\_SEX CHECK(sex IN('男','女')),  CONSTRAINT CK\_AGE CHECK(age BETWEEN 0 AND 200)  ); |

先表中插入合法数据和非法数据：

|  |  |
| --- | --- |
| INSERT INTO member(mid,name,sex,age) VALUES(1,'张三','男',50);  INSERT INTO member(mid,name,sex,age) VALUES(2,'李四','中',40);  INSERT INTO member(mid,name,sex,age) VALUES(3,'王五','女',250); | |
|  | |

## 外键约束

前面四种约束都是对一张表中的数据进行约束，而外键约束是对多张表中的数据进行约束；例如在一个图书管理系统中，一个人可以借多本书，则涉及两张表member,book,且book表从属于member表，此时book表中应该有一个字段用以标定是谁借了该书，可以用member表中的主键作为标志。

外键约束特点：

* 外键用于确保相关的两个字段之间的参照关系，以实现参照完整性约束；
* 外键约束通常构建于来自不同表的两个字段之间；
* 子表外键列的值必须在主表参照列值的范围内，或者为空；
* 外键参照的必须是主表的主键或者唯一键；
* 主表主键/唯一键值被子表参照时，主表相应记录不允许被删除；

示例1：

|  |
| --- |
| CREATE TABLE salary(  eid NUMBER(3),  basic\_salary NUMBER(8,2),  job\_allowance NUMBER(8,2),  travelling\_allowance NUMBER(8,2),  personal\_income\_tax NUMBER(8,2),  **CONSTRAINT salary\_eid\_fk FOREIGN KEY(eid) REFERENCES empinfo(eid)**  ); |
| CREATE TABLE salary(  **EID NUMBER(3) REFERENCES empinfo(eid),**  ---  ); |
| CREATE TABLE salary(  eid NUMBER(3) ,  **PRIMARY KEY REFERENCES empinfo(eid)**  ---  ); |

示例2：

建表如下：

|  |
| --- |
| DROP TABLE member PURGE;  DROP TABLE book PURGE;  CREATE TABLE member(  mid NUMBER(3),  name VARCHAR2(30) NOT NULL,  CONSTRAINT PK\_MID PRIMARY KEY(mid)  );  CREATE TABLE book(  bid NUMBER(3),  title VARCHAR2(30) NOT NULL,  mid NUMBER(3),  CONSTRAINT PK\_BID PRIMARY KEY(bid)  ); |

向表中插入相关数据：

|  |
| --- |
| INSERT INTO member(mid,name) VALUES(1,'张三');  INSERT INTO member(mid,name) VALUES(2,'李四');  INSERT INTO book(bid,title,mid) VALUES(1,'微积分',1);  INSERT INTO book(bid,title,mid) VALUES(2,'线性代数',1);  INSERT INTO book(bid,title,mid) VALUES(3,'概率论',2);  INSERT INTO book(bid,title,mid) VALUES(4,'数理统计',2);  INSERT INTO book(bid,title,mid) VALUES(5,'离散数学',2); |

现在进行查询：

统计每个人各有几本书

|  |
| --- |
| SELECT m.mid,m.name,COUNT(b.bid)  FROM member m,book b  WHERE m.mid=b.mid  GROUP BY m.name,m.mid; |
|  |

查询所有书籍并列出相应的借书人

|  |
| --- |
| SELECT b.bid,b.title,b.mid,m.name  FROM book b,member m  WHERE b.mid=m.mid; |
|  |

但member与book之间的逻辑并不受数据库控制，book中的mid值可以随意指定而不受member表影响，故需要在两张表之间增加一种约束，用以限制这种不合理；

先向表中插入不合理数据:

|  |
| --- |
| INSERT INTO book(bid,title,mid) VALUES(6,'组合数学',3);  INSERT INTO book(bid,title,mid) VALUES(7,'数学分析',4); |
|  |

可以看出，虽然member表中并不存在mid为3，4的记录，但数据还是成功添加了。接下来就为book表中的mid添加外键，用以约束这种错误；

|  |
| --- |
| CREATE TABLE book(  bid NUMBER(3),  title VARCHAR2(30) NOT NULL,  mid NUMBER(3),  CONSTRAINT PK\_BID PRIMARY KEY(bid),  CONSTRAINT FK\_MID FOREIGN KEY(mid) REFERENCEs member(mid)  ); |

向表中插入合法数据和非法数据：

|  |
| --- |
| INSERT INTO member(mid,name) VALUES(1,'张三');  INSERT INTO member(mid,name) VALUES(2,'李四');  INSERT INTO book(bid,title,mid) VALUES(1,'微积分',1);  INSERT INTO book(bid,title,mid) VALUES(2,'线性代数',1);  INSERT INTO book(bid,title,mid) VALUES(3,'概率论',2);  INSERT INTO book(bid,title,mid) VALUES(4,'数理统计',2);  INSERT INTO book(bid,title,mid) VALUES(5,'离散数学',2);  INSERT INTO book(bid,title,mid) VALUES(6,'组合数学',3);  INSERT INTO book(bid,title,mid) VALUES(7,'数学分析',4); |
|  |

可以看出，插入不合法数据时，数据库报错；

### 问题1：删除级联

虽然外键 保证了父表与子表中数据的一致性，但也带来了一些问题，例如当删除父表中的记录时，如果有子表中的记录与之关联，则不可以删除；

|  |
| --- |
|  |

解决该问题，可以通过两种方法：

**方法1**：先删除子表中相应mid的数据，在删除父表中相应mid的数据；

|  |
| --- |
| DELETE FROM book WHERE mid=1;  DELETE FROM member WHERE mid=1; |
|  |

**方法2**：通过在创建表时指定删除级联，在删除父表数据时自动删除相应子表数据；

|  |
| --- |
| DROP TABLE book PURGE;  DROP TABLE member PURGE;  CREATE TABLE member(  mid NUMBER(3),  name VARCHAR2(30) NOT NULL,  CONSTRAINT PK\_MID PRIMARY KEY(mid)  );  CREATE TABLE book(  bid NUMBER(3),  title VARCHAR2(30) NOT NULL,  mid NUMBER(3),  CONSTRAINT PK\_BID PRIMARY KEY(bid),  CONSTRAINT FK\_MID FOREIGN KEY(mid) REFERENCES member(mid) ON DELETE CASCADE  ); |
|  |

### 问题2：删除设空

上述方法虽然删除了父表数据，但对应子表中的数据也被删除了，这种处理方式非常不友好，故可以考虑删除父表数据，而将子表对应的数据设置为空；

|  |
| --- |
| CREATE TABLE book(  bid NUMBER(3),  title VARCHAR2(30) NOT NULL,  mid NUMBER(3),  CONSTRAINT PK\_BID PRIMARY KEY(bid),  CONSTRAINT FK\_MID FOREIGN KEY(mid) REFERENCES member(mid) ON DELETE SET NULL  ); |
|  |

### 问题 3：先删子表再删父表

若要删除父表，首先要删除其相关的子表：

|  |
| --- |
| DROP TABLE book PURGE;  DROP TABLE member PURGE; |

但在许都情况下，并不知道数据表与其他数据表之间的关系，Oracle提供一种能够强制删除父表的方法：

|  |
| --- |
| **DROP TABLE member CASCADE CONSTRAINT PURGE;** |
|  |

但以上做法并不推荐使用，因为可能会造成关系缺失或混乱，一般情况下，还是要先删子表，再删父表；

## 修改约束(表级定义)

当数据表已经建立好之后，如果要对数据表中数据或表间数据增加或删除约束时，就需要用到修改约束的语法：

**语法1：增加约束**

|  |
| --- |
| ALTER TABLE 表名称 ADD CONSTRAINT 约束名称 约束类型（字段）; |

**特例**：非空约束必须使用modify子句添加

|  |
| --- |
| ALTER TABLE student  MODIFY (name CHAR(15) DEFAULT 'N/A' NOT NULL); |

1. 创建一个member表，包含字段：mid,name,age,并通过增加约束的语法为其添加主键；

|  |
| --- |
| DROP TABLE member CASCADE CONSTRAINT PURGE;  CREATE TABLE member (  mid NUMBER(3),  name VARCHAR2(30) NOT NULL,  age NUMBER(3)  );  ALTER TABLE member ADD CONSTRAINT pk\_mid PRIMARY KEY(mid); |
|  |

1. 对于上述例子，如果表中已经存在非法数据，如果要增加检查约束则数据库会报错；

|  |
| --- |
| INSERT INTO member(mid,name,age) VALUES(1,'张三',300);  INSERT INTO member(mid,name,age) VALUES(2,'李四',-20);  ALTER TABLE member ADD CONSTRAINT ck\_age CHECK(age BETWEEN 0 AND 200); |
|  |

**语法2：删除约束**

|  |
| --- |
| **ALTER TABLE 表名称 DROP CONSTRAINT 约束名称 [CASCADE];** |

**注：**若所删除的字段为主键且与其他表有外键关联的话，则需要在删除时指定级联删除，在语句后加关键字CASCADE;

1. 删除上述member表中的主键；

|  |
| --- |
| ALTER TABLE member DROP CONSTRAINT pk\_mid; |
|  |

**注：**

* 对于约束来说，一般确定好的约束就不要更改，防止关系混乱；
* 要想修改约束，首先要知道约束的名称，对于非空约束来说，修改操作不能使用；

## 查询约束

如果想要知道约束的信息诸如约束名称、约束所属的数据表、约束字段等，可以通过查询数据字典得到；Oracle提供了两种数据字典供约束查询：user\_constraints , user\_cons\_columns;

1. 采用user\_constraints数据字典查看约束信息；

|  |  |
| --- | --- |
| SELECT OWNER,CONSTRAINT\_NAME,TABLE\_NAME FROM user\_constraints; | |
|  |

user\_constraints数据字典无法查看约束所对应的字段，这就需要用到user\_cons\_columns数据字典；

1. 采用user\_cons\_columns数据字典查看约束信息；

|  |
| --- |
| COL owner FOR A15;  COL constraint\_name FOR A15;  COL table\_name FOR A15;  COL column\_name FOR A15;  SELECT owner,constraint\_name,table\_name,column\_name FROM user\_cons\_columns; |
|  |

## 禁用约束

在ALTER TABLE语句中，还可使用DISABLE CONSTRAINT子句禁用已有约束,也可使用CASCADE选项将相关联的约束也一并禁用。

|  |
| --- |
| ALTER TABLE *table*  DISABLE CONSTRAINT *constraint\_name* [CASCADE]; |

## 11. 启用约束

在ALTER TABLE语句中，可使用ENABLE CONSTRAINT子句启用先前被禁用的约束。

|  |
| --- |
| ALTER TABLE *table*  ENABLE CONSTRAINT *constraint\_name*; |

示例：

|  |
| --- |
| CREATE TABLE student(  sid NUMBER(10) ,  name VARCHAR2(20),  CONSTRAINT student\_sid\_pk PRIMARY KEY(sid)  );  ---  ALTER TABLE student DISABLE CONSTRAINT student\_sid\_pk ;  ---  ALTER TABLE student ENABLE CONSTRAINT student\_sid\_pk ; |

**注**：:和关闭约束操作的情形有所不同，此时无法再使用cascade选项一并启用相关联的其它约束。

## 表级定义与列级定义

* 表级定义：在定义了所有的字段之后，在定义约束；
* 列级定义：在定义字段的同时定义约束；

## 综合案例

### 商店售货系统表设计案例1

现有一个商店的数据库，记录客户及其购物情况，由下面三个表组成：

* 商品goods（商品号goodsId，商品名 goodsName，单价 unitprice，商品类别category，供应商provider）；
* 客户customer（客户号customerId，姓名name，住在address，电邮email，性别sex，身份证cardId）；
* 购买purchase（客户号customerId，商品号goodsId，购买数量nums）；

请用SQL语言完成下列功能：

建表，在定义中要求声明：

(1). 每个表的主外键；

(2). 客户的姓名不能为空值；

(3). 单价必须大于0，购买数量必须在1到30之间；

(4). 电邮不能够重复；

(5). 客户的性别必须是 男 或者 女，默认是男；

|  |
| --- |
| DROP TABLE purchase PURGE;  DROP TABLE goods PURGE;  DROP TABLE customer PURGE;  CREATE TABLE goods(  goodsId CHAR(8) PRIMARY KEY,  goodsName VARCHAR2(30) NOT NULL,  unitPrice NUMBER(10,2) CHECK(unitPrice > 0),  category VARCHAR2(10),  provider VARCHAR2(30)  );  CREATE TABLE customer(  customerId CHAR(8) PRIMARY KEY,  name VARCHAR2(30) NOT NULL,  address VARCHAR2(30),  email VARCHAR2(30) UNIQUE,  sex CHAR(2) CHECK (sex IN('男','女')),  cardId CHAR(18) UNIQUE  );  CREATE TABLE purchase(  customerId CHAR(8) REFERENCES customer(customerId),  goodsId CHAR(8) REFERENCES goods(goodsId),  nums NUMBER(5) CHECK (nums BETWEEN 1 AND 30)  ); |

若建表后还需要添加约束，则可以用ALTER TABLE语句。例如：要为上例中地址设定约束，范围在“海淀”，“朝阳”，“东城”，“西城”，“通州”，“崇文”；

|  |
| --- |
| ALTER TABLE customer ADD CONSTRAINT address CHECK(address IN('海淀','朝阳','东城','西城','通州','崇文')); |

# 建表、更新、查询综合练习

有某个学生运动会比赛信息的数据库，保存了如下的表：

* 运动员sporter(运动员编号 sporterid, 运动员姓名 name, 运动员性别 sex, 所属系号 department);
* 项目 item(项目编号 itemid, 项目名称 itemname, 项目比赛地点 location);
* 成绩 grade(运动员编号 sporterid, 项目编号 itemid, 积分 mark);

请用SQL语句完成如下功能：

1. 建表，并在相应字段上增加约束；

* 定义各个表的主键和外键约束；
* 运动员的姓名和所属系别不能为空；
* 积分要么为空值，要么为6,4,2,0，分别代表第一，二，三名和其他名次的积分，注意名次可以有并列名次，后面的排名不往前提升，例如，如果有两个并列第一，则没有第二名。

创建数据表

|  |
| --- |
| DROP TABLE grade PURGE;  DROP TABLE sporter PURGE;  DROP TABLE item PURGE;  CREATE TABLE sporter(  sporterid NUMBER(5),  name VARCHAR2(30) NOT NULL,  sex VARCHAR2(2),  department VARCHAR2(30) NOT NULL,  CONSTRAINT pk\_sporterid PRIMARY KEY(sporterid),  CONSTRAINT ck\_sex CHECK(sex IN('男','女'))  );  CREATE TABLE item(  itemid NUMBER(5),  itemname VARCHAR2(30),  location VARCHAR2(30),  CONSTRAINT pk\_itemid PRIMARY KEY(itemid)  );  CREATE TABLE grade(  sporterid NUMBER(5),  itemid NUMBER(5),  mark NUMBER(5),  CONSTRAINT fk\_sporterid FOREIGN KEY(sporterid) REFERENCES sporter(sporterid) ON DELETE CASCADE,  CONSTRAINT fk\_itemid FOREIGN KEY(itemid) REFERENCES item(itemid) ON DELETE CASCADE,  CONSTRAINT ck\_mark CHECK(mark IN(null,6,4,2,0))  ); |

1. 向表中插入指定数据；

|  |
| --- |
| 运动员( |
| 1001，李明，男，计算机系 |
| 1002，张三，男，数学系 |
| 1003，李四，男，计算机系 |
| 1004，王二，男，物理系 |
| 1005，李娜，女，心理系 |
| 1006，孙丽，女，数学系) |
| 项目( |
| x001，男子五千米，一操场 |
| x002，男子标枪，一操场 |
| x003，男子跳远，二操场 |
| x004，女子跳高，二操场 |
| x005，女子三千米，三操场) |
| 积分( |
| 1001,x001,6 |
| 1002,x001,4 |
| 1003,x001,2 |
| 1004,x001,0 |
| 1001,x003,4 |
| 1001,x003,6 |
| 1002,x003,2 |
| 1004,x001,6 |
| 1005,x004,6 |
| 1006,x004,4 |
| 1003,x002,6 |
| 1005,x002,4 |
| 1006,x002,2 |
| 1001,x002,0) |

插入数据

|  |
| --- |
| INSERT INTO sporter(sporterid,name,sex,department) VALUES(1001,'李明','男','计算机系');  INSERT INTO sporter(sporterid,name,sex,department) VALUES(1002,'张三','男','数学系');  INSERT INTO sporter(sporterid,name,sex,department) VALUES(1003,'李四','男','计算机系');  INSERT INTO sporter(sporterid,name,sex,department) VALUES(1004,'王二','男','物理系');  INSERT INTO sporter(sporterid,name,sex,department) VALUES(1005,'李娜','女','心理系');  INSERT INTO sporter(sporterid,name,sex,department) VALUES(1006,'孙丽','女','数学系'); |
| INSERT INTO item(itemid,itemname,location) VALUES('x001','男子五千米','一操场');  INSERT INTO item(itemid,itemname,location) VALUES('x002','男子标枪','一操场');  INSERT INTO item(itemid,itemname,location) VALUES('x003','男子跳远','二操场');  INSERT INTO item(itemid,itemname,location) VALUES('x004','女子跳高','二操场');  INSERT INTO item(itemid,itemname,location) VALUES('x005','女子三千米','三操场'); |
| INSERT INTO grade(sporterid,itemid,mark) VALUES(1001,'x001',6);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1002,'x001',4);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1003,'x001',2);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1004,'x001',0);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1001,'x003',4);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1001,'x003',6);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1002,'x003',2);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1004,'x001',6);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1005,'x004',6);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1006,'x004',4);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1003,'x002',6);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1005,'x002',4);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1006,'x002',2);  INSERT INTO grade(sporterid,itemid,mark) VALUES(1001,'x002',0); |

1. 执行以下数据操作；
2. 求出目前总积分最高的系名，及其积分；

**思路：**

确定相关数据表：sporter表，求出系名称；grade表，求出相应积分；

确定相关联字段：sporter.sporterid=grade.sporterid;

**第一步**：求出部门姓名，及其总积分；

|  |
| --- |
| SELECT s.department,SUM(g.mark)  FROM sporter s,grade g  WHERE (s.sporterid=g.sporterid)  GROUP BY s.department; |

**第二步**：利用统计函数求出最大值；

**方法1**：将表中数据降序排列，并取出第一条数据；

|  |
| --- |
| SELECT \* FROM(  SELECT s.department,SUM(g.mark) sum\_mark  FROM sporter s,grade g  WHERE (s.sporterid=g.sporterid)  GROUP BY s.department  ORDER BY sum\_mark DESC)  WHERE ROWNUM=1; |

**方法2**：利用含有统计函数的查询语句嵌套；

|  |
| --- |
| SELECT s.department,SUM(g.mark)  FROM sporter s,grade g  WHERE (s.sporterid=g.sporterid)  GROUP BY s.department  HAVING(SUM(g.mark)=  (SELECT MAX(SUM(g.mark))  FROM sporter s,grade g  WHERE (s.sporterid=g.sporterid)  GROUP BY s.department)); |
|  |

1. 找出在一操场进行比赛的各项目名称及其冠军的姓名；

**思路：**

确定相关的成绩表：sporter表：运动员的姓名；

item表：比赛项目名称；

grade表：按积分求出冠军名称；

确定相关联的条件：sporter.sporterid=grade.sporterid;

item.itemid=grade.itemid;

**第一步：**查询出一操场进行的项目id；

|  |
| --- |
| SELECT i.itemid FROM item i  WHERE i.location='一操场'; |
|  |

**第二步：**求出各个项目中冠军的成绩；

|  |
| --- |
| SELECT i.itemid ,MAX(g.mark)  FROM item i,grade g  WHERE (i.location='一操场')  AND (i.itemid=g.itemid)  GROUP BY i.itemid; |
|  |

**第三步：**根据成绩找出运动员姓名；

|  |
| --- |
| SELECT s.name,g.itemid,temp.max\_mark  FROM sporter s,grade g,  (SELECT i.itemid itemid,MAX(g.mark) max\_mark  FROM item i,grade g  WHERE (i.location='一操场')  AND (i.itemid=g.itemid)  GROUP BY i.itemid) temp  WHERE(s.sporterid=g.sporterid)  AND(g.mark=temp.max\_mark)  AND(g.itemid=temp.itemid); |
|  |

**第四步：**根据itemid引入项目名称；

|  |
| --- |
| SELECT s.name,g.itemid,temp.itemname,temp.max\_mark  FROM sporter s,grade g,  (SELECT i.itemid itemid,i.itemname itemname,MAX(g.mark) max\_mark  FROM item i,grade g  WHERE (i.location='一操场')  AND (i.itemid=g.itemid)  GROUP BY i.itemid,i.itemname) temp  WHERE(s.sporterid=g.sporterid)  AND(g.mark=temp.max\_mark)  AND(g.itemid=temp.itemid); |

1. 找出参加了张三所参加的所有项目的其他同学的姓名；

**思路：**

确定所需要的表：sporter表：确定姓名

item表：确定参加项目

grade表：关联sporter表与item表

确定关联条件：sporter.sportid=grade.sportid;

item.itemid=grade.itemid;

**第一步：**找出李明参加的项目id

|  |
| --- |
| SELECT g.itemid  FROM grade g,sporter s  WHERE (s.name='张三')  AND (g.sporterid=s.sporterid) |
|  |

**第二步：**找出参与项目的运动员id及姓名

|  |
| --- |
| SELECT DISTINCT s.sporterid ,s.name  FROM sporter s,grade g  WHERE (g.sporterid=s.sporterid)  AND(g.itemid IN(  SELECT g.itemid  FROM grade g,sporter s  WHERE (s.name='张三')  AND (g.sporterid=s.sporterid)))  AND(s.name<>'张三'); |
|  |

1. 经查张三因为使用了违禁药品，其成绩都记0分，请在数据库中作出修改；

**思路：**

确定关联的表：sporter表：确定姓名；

grade表：找到项目并设置成绩；

确定关联字段：sporter.sporterid=grade.sporterid;

**第一步：**找到张三所参加的项目id;

|  |
| --- |
| SELECT s.sporterid  FROM sporter s  WHERE (s.name='张三'); |

**第二步：**根据运动员id更新grade分数;

|  |
| --- |
| UPDATE grade SET mark=0  WHERE grade.sporterid=(SELECT s.sporterid  FROM sporter s  WHERE (s.name='张三')); |

1. 经组委会协商，需要删除女子跳高比赛项目；

**思路：**因为之前建表之时已经确定了级联删除，故可以直接删除父表中的数据；

|  |
| --- |
| DELETE FROM item WHERE itemname='女子跳高'; |

# 集合操作

在数学中，有集合的交、并、补运算，而在数据查询中也由此概念；集合操作有如下几个关键字：

* UNION:连接两个查询，作并运算，去掉重复数据；
* UNION ALL:连接两个查询，作并运算，且显示重复数据；
* INTERSECT:对两个查询作交运算
* MINUS：返回两个查询中不同的部分

首先创建一个数据表emp20用来保存部门号为20的所有员工信息；

|  |
| --- |
| CREATE TABLE emp20 AS (SELECT \* FROM emp WHERE deptno=20); |
|  |

1. UNION操作：

|  |
| --- |
| SELECT \* FROM emp  UNION  SELECT \* FROM emp20; |
|  |

1. UNION ALL操作：

|  |
| --- |
| SELECT \* FROM emp  UNION ALL  SELECT \* FROM emp20; |
|  |

1. INTERSECT操作：

|  |
| --- |
|  |

1. MINUS操作：

|  |  |
| --- | --- |
| SELECT \* FROM emp  MINUS  SELECT \* FROM emp20; | |
|  |

1. 查询emp表，对所有领取奖金的人求出平均工资，对不领取奖金的人求出平均工资；

|  |
| --- |
| SELECT 'UNCOMM',AVG(sal) FROM emp WHERE comm IS NULL  UNION  SELECT 'COMM',AVG(sal) FROM emp WHERE comm IS NOT NULL; |
|  |

**注**：集合操作的速度要比直接查询的效率要高一些；

# 序列

## 5.1 序列的定义

在许多数据库中，存在着对数据字段进行自动增长的方法。Oracle中，数据的自动增长由序列控制。

* 系统自动生成的、不重复的整数值；
* 序列是一种数据库对象，可以被多个用户共享；
* 典型用途是做为主键值，它对于每一行必须是唯一的；
* 序列可以代替应用程序编号；
* 可以对序列值进行缓冲存储，以提高访问效率；

## 5.2 序列创建

语法如下：

|  |
| --- |
| **CREATE SEQUENCE 序列名称**  **[INCREMENT BY n][START WITH n]**  **[{MAXVALUE n|NOMAXVALUE}]**  **[{MINVALUE n|NOMINVALUE}]**  **[{CYCLE|NOCYCLE}]**  **[{CACHE n|NOCACHE}];** |

1. 创建序列

|  |
| --- |
| CREATE SEQUENCE myseq; |
|  |

## 5.3访问序列

语法如下：

|  |
| --- |
| 序列名称.nextval:让序列增长到下一个内容； |
| 序列名称.currval:取得当前序列内容； |

1. 访问序列

|  |
| --- |
| SELECT myseq.currval FROM dual; |
|  |

**注**：如果要使用currval，必须先使用nextval;

|  |
| --- |
| SELECT myseq.nextval FROM dual; |
|  |
|  |

## 5.4序列的使用

序列一般作为数据表的主键使用，必须在插入数据时手动添加到记录中；

语法：

|  |
| --- |
| SELECT mysequence1.currval FROM dual;  SELECT mysequence1.nextval FROM dual;  INSERT INTO test1 VALUES(mysequence1.nextval, 'TOM'); |

1. 序列的使用；

|  |
| --- |
| CREATE TABLE mytab(  id NUMBER PRIMARY KEY,  name VARCHAR2(20) NOT NULL  );  INSERT INTO mytab(id,name) VALUES(myseq.nextval,'张三');  INSERT INTO mytab(id,name) VALUES(myseq.nextval,'李四');  INSERT INTO mytab(id,name) VALUES(myseq.nextval,'王五'); |
|  |

**注**：在默认情况下，序列从0开始增长，每次增长1；

1. 创建序列，从10开始，每次增长2；

|  |
| --- |
| DROP SEQUENCE myseq;  CREATE SEQUENCE myseq INCREMENT BY 2 START WITH 10; |
|  |

1. 创建一个序列，可以在1,3,5,7,9之间循环出现；

|  |
| --- |
| DROP SEQUENCE myseq;  CREATE SEQUENCE myseq INCREMENT BY 2 START WITH 1  MAXVALUE 10 MINVALUE 1  CYCLE NOCACHE; |
|  |

**注**：CACHE机制：在Oracle中，由于序列经常使用，每次使用时都会占用一定的资源和时间，为了提升性能，Oracle采用缓存机制，为用户提前分配一片空间，并在其中自动生成好了序列，用户在使用序列时，实际上是调用缓存区中的序列；但这样做有一个问题，当数据库实例关闭时，保存在缓存中的内容也就随之消失了，但表中数据已经增长了，就会出现跳号，解决方法就是将序列设置为不缓存，用NOCACHE声明；

|  |
| --- |
| DROP SEQUENCE myseq;  CREATE SEQUENCE myseq INCREMENT BY 2 START WITH 1  MAXVALUE 10 MINVALUE 1  CYCLE ; |
|  |

## 5.5 修改序列

语法：

|  |
| --- |
| **ALTER SEQUENCE [SCHEMA.]序列名**  **[INCREMENT BY *n*]**  **[{MAXVALUE *n* | NOMAXVALUE}]**  **[{MINVALUE *n* | NOMINVALUE}]**  **[{CYCLE | NOCYCLE}]**  **[{CACHE *n* | NOCACHE}]**  **[{ORDER | NOORDER}];** |

**注**：

* 操作者必须是序列的所有者、或者拥有alter该序列的权限;
* 只有未来再生成的序列数受影响;
* 序列的初始值不可更改;
* 更改中会进行一些验证，比如新的maxvalue如果小于当前的序列值就会报错;

## 5.6删除序列

* 使用drop sequence 语句删除序列；
* 操作者须是序列的所有者、或拥有drop该sequence的权限；

# 视图

## 视图（VIEW）的定义

视图是由从一个或多个表（或视图）中提取数据而成；视图是一种虚拟表，一经创建，可以当做表来使用；在数据库中并不保存视图所包含数据，只保存视图的定义，当需要视图数据时，数据库会根据视图的定义提取数据；视图相当于多条查询语句的集合；

|  |
| --- |
|  |

视图的优点：

1. 简单化复杂查询：
2. 提高运行效率：使用保存的SQL语句；
3. 隐藏数据库表结构，实现数据逻辑独立性，且有更好地封装性；
4. 限制对数据库的访问：可以限制到一个具体的数据对象上（表）甚至是字段和记录上；
5. 在相同的数据上提供不同的视图，便于数据共享；

简单视图和复杂视图：

|  |
| --- |
|  |

## 创建视图

通过在create view语句中嵌入子查询的方式创建视图；语法：

|  |
| --- |
| CREATE [OR REPLACE] [FORCE/NOFORCE]VIEW [*SCHEMA.*] 视图名 视图别名[(字段别名[,字段别名n]...)]  AS 子查询 [WITH READ ONLY / WITH CHECK OPTION]*;* |

**注**：

* 若缺省字段，则默认为查询全部字段；
* 若查询字段中包含表达式，则需要指定视图别名；

1. 创建视图，包含emp表中20部门的员工记录；

|  |
| --- |
| CREATE OR REPLACE VIEW my\_first\_view(编号,姓名,职位,工资)  AS (SELECT empno,ename,job,sal  FROM emp  WHERE deptno=20); |
|  |

## 补：视图权限授权

若创建视图时提示：第 1 行出现错误:ORA-01031: 权限不足

则说明登陆的用户没有建立视图的权限，可以换个有该权限的用户来登陆或为你登陆的用户加入权限  
（1）首先切换到普通管理员或者超级管理员

|  |
| --- |
| **CONN SYSTEM/密码 AS SYSDBA/SYSOPER;** 或者 **CONN SYS/密码 AS SYSDBA/SYSOPER;** |

（2）授权

|  |
| --- |
| **GRANT CREATE VIEW TO SCOTT;** |

（3）切换回普通用户

|  |
| --- |
| **CONN SCOTT/密码;** |

此时scott可以建立视图了。

## 查看视图结构

语法：

|  |
| --- |
| **DESC 视图类型；** |
|  |

## 删除视图

|  |
| --- |
| **DROP VIEW 视图名称；** |

## 查询视图

实现原理：

|  |
| --- |
|  |

用户输入查询语句，数据库查询视图定义，然后执行子查询读取数据，之后返回给用户；

## 强制创建视图：

不管创建的视图所基于的表是否存在，都强制先给出视图的定义；通过在CREATE后加FORCE/NOFORCE 来指定是否强制创建视图；默认为非强制创建视图；

1. 强制创建视图，使其基于一个未创建的表emp2;

|  |
| --- |
| CREATE OR REPLACE FORCE VIEW my\_second\_view  AS (SELECT empno,ename,job,sal  FROM emp2  WHERE deptno=20); |
|  |

强制创建视图告诉数据库，该表将会在之后创建；当emp2创建之后，视图便可以使用：

|  |
| --- |
|  |

## 创建复杂视图

1. 创建工资视图，统计emp表中各部门的最高工资，最低工资，平均工资；

|  |
| --- |
| CREATE OR REPLACE VIEW v\_sal(部门编号,最高工资,最低工资,平均工资)  AS(SELECT deptno,MAX(sal),MIN(sal),AVG(sal)  FROM emp  GROUP BY deptno); |
|  |

1. 创建视图，包含员工编号，员工姓名，职位，年薪，工龄（月），部门编号，部门名称；

|  |
| --- |
| CREATE OR REPLACE VIEW v\_emp\_dept(员工编号,姓名,职位,年薪,"工龄（月）",不门编号,部门名称)  AS (SELECT e.empno,e.ename,e.job,e.sal\*12,MONTHS\_BETWEEN(SYSDATE,e.hiredate),d.deptno,d.dname  FROM emp e,dept d  WHERE e.deptno=d.deptno); |
|  |

## 更新视图

在可更新视图上进行DML操作，可以修改基表数据；

可更新视图的条件：

* 可更新视图的定义中不能使用分组函数、GROUP BY子句、DISTINCT关

键字、ROWNUM伪列，字段的定义不能为表达式；

* 由两个以上基表中导出的视图不可更新；
* 基表中非空的列在视图定义中未包括，则不可在视图上进行INSERT操作；

在视图上进行DML的操作，语法与在表上操作相同；

## 创建只读视图

创建视图时，有两个选项：

1. WITH CHECK OPTION:在创建视图时，会指定创建条件，但当更新视图数据时，若修改数据的创建条件，则视图中的元素也会出现增减，为了避免此种结果，可以再创建视图时加WITH CHECK OPTION；从而不能更新视图创建条件；
2. WITH READ ONLY:对视图中数据项的更改，会更改基表中的项，故要防止通过视图修改基表中的数据，可以通过WITH READ ONLY来限制视图为只读视图；

为防止对视图进行DML操作而篡改基表，可以创建视图时，使用WITH READ ONLY将视图设置为可读，即不可以进行DML操作；

## 临时视图

嵌入到SQL语句中的子查询是临时视图，临时视图不是数据对象，其定义不会长久保存在数据库中，本次运行后即被清除；

|  |
| --- |
|  |

# 索引

## 索引的定义

* 一种用于快速提升查询效率的数据库对象；
* 通过快速定位数据的方法，减少磁盘的I/O操作；
* 索引信息与表独立存放；
* Oracle数据库自动使用和维护索引；
* 索引实现的原理是通过生成一个特定的单词表并排序，来实现快速查找，从而提升效率；

## 索引分类

1. 按照索引包含的字段数划分：

* 单列索引：基于单个列创建的索引；
* 复合索引：基于两个列或多个列的索引；

例：

|  |
| --- |
| CREATE INDEX goods\_idx1 ON goods(goodsName,unitprice); |
| CREATE INDEX goods\_idx2 ON goods(unitprice,goodsName); |

以上两个索引是不同的，因为字段顺序不同，创建索引的顺序也不同；数据库扫描时，会从后边字段开始向前扫描，因此，我们应该尽量将能够将数据范围限定很小的字段放在最后面，从而提高查询效率；

1. 按照索引指的唯一性：

* 唯一性索引：单词表中的单词都是唯一的；
* 非唯一索引：单词表中有可能存在重复的单词；

1. 按照存储方式：

* B\*树索引
* 反向索引
* 位图索引

1. 其他索引

* 函数索引
* 全局索引
* 分区索引

## 3. 创建索引的方式

1）自动创建：在定义主键或唯一约束时系统会自动在相应的字段上创建唯一性索引；

2）手动创建：用户可以在其他列上创建非唯一的索引，以加速查询；可以用CREATE INDEX来手动创建索引；

语法：

|  |
| --- |
| **CREATE INDEX [SCHEMA.] 索引名**  **ON 表名 (字段[,字段2]...);** |

1. 查询emp表中所有薪水大于1500的员工信息

按照之前的写法，应该如下：

|  |
| --- |
| SELECT \* FROM emp WHERE sal>1500; |
|  |

以上由于没有使用索引，所以查询方式采用逐行扫描逐行判断的方式，如果数据量很大，则效率低下；假设有一种数据结构如下：

|  |
| --- |
|  |

则查询比1600小大的数字时，就不必查找所有数据，只需要查找1600右边的所有节点即可；上例中可以为sal创建一个索引，从而提高查询效率；

|  |
| --- |
| CREATE INDEX my\_emp\_sal\_index ON emp(sal); |
|  |

以上索引称为B树索引，索引可以在查询大数据量的情况下提高查询效率，只要维持以上的树结构，但是如果树上的元素频繁修改的话，数据库需要不断维护，则会严重降低效率；故索引一般只会使用在不会频繁修改的表中；

1. 为emp表中的ename字段创建索引

|  |
| --- |
| CREATE INDEX my\_first\_index  ON emp(ename); |

## 删除索引

使用DROP INDEX删除索引；语法：

|  |
| --- |
| **DROP INDEX 索引名;** |

**注**：

* 操作者须是索引的所有者、或拥有drop该index的权限
* 删除表时相关的索引(和约束)将被自动删除，但视图和序列将保留

## 创建索引的原则

1)下述情况可以创建索引

* 若表中的字段为主键，则自动为主键创建索引；
* 字段取值分布范围很广；
* 字段中包含大量空值；
* 字段经常出现在where 子句或连接条件中；
* 表经常被访问、数据量很大，且通常每次访问的数据量小于记录总量的2%~4%；
* 索引的层次不要超过4层；
* 若字段约束为unique，则自动为其加索引；

2)下列情况不适合创建索引

* 表很小；
* 字段不经常出现在where子句中；
* 每次访问的数据量大于记录总数的2%~4%
* 表经常更新：表更新，索引也会随之更新；
* 被索引的字段作为表达式的的一部分被引用；
* 同一张表上可以有多个索引，但是字段的组合必须是唯一的；

3) 索引的缺点：

* 创建索引，大约要用掉占表大小1.2倍的硬盘和内存来保存索引；
* 更新数据时，系统必须要有额外的时间去更新索引，以维持数据和索引的一致性；

## 查看索引

* 查询用户字典视图user\_indexes可得到用户的所有索引；
* 查询用户字典视图user\_ind\_columns可获知索引建立在哪些字段上；

## 基于函数的的索引

基于表达式的索引被统称为基于函数的索引——索引表达式由表中的字段、常量、SQL函数和自定义函数构建而成。创建语法：

|  |
| --- |
| **CREATE INDEX [SCHEMA.] 索引名**  **ON 表名(函数名(字段名));** |

1. 创建并使用LOWER(ename)的索引；

|  |
| --- |
| CREATE INDEX myindex  ON emp(LOWER(ename)); |
| SELECT \* FROM emp  WHERE LOWER(ename)='king'; |

# 同义词

同义词相当于对象的别名，使用同义词可以方便访问其它用户的对象,缩短对象名字的长度

## 创建同义词

语法：

|  |
| --- |
| **CREATE [PUBLIC] SYNONYM 同义词**  **FOR 数据对象;** |

示例：

|  |
| --- |
| CREATE SYNONYM ge1 FOR emp; |

## 使用同义词

语法：

|  |
| --- |
| **SELECT \* FROM 同义词;** |

## 3.删除同义词

语法：

|  |
| --- |
| **DROP SYNONYM 同义词;** |

例如：dual表属于sys用户，但是其他用户访问dual表时并没有加上sys.dual,这是因为dual就是sys.dual的同义词；

1. 创建一个myemp同义词，指向scott.emp表；

|  |
| --- |
|  |
|  |

若创建同义词时没有加PUBLIC关键字，则同义词为局部同义词，只限定于创建同义词的用户使用；若要公用，则加PUBLIC关键字；