第三节

结构化查询语言

关系数据语言

- 关系数据库中，提供给用户对数据进行操作的语言称为关系数据语言
- 关系数据语言的功能：

1数据定义：定义数据模式、数据类型以建立数据模型

2数据操纵：对数据进行查询、更新（插入删除修改等）

3数据控制：对数据的使用权限、完整性、一致性进行控制，以达到数据既能共享又安全保密

关系数据语言的特点1

- 一体化的特点

  层次数据库与网状数据库系统采用格式化模型的数据语言，分为模式数据定义语言、子模式数据定义语言、数据操纵语言和数据存储描述语言。

  而关系数据语言同时具有数据操纵、定义、控制功能

关系数据语言的特点2

- 非过程化特点

  格式化模型的数据语言较为繁琐，操作是过程化的，用户必须按照模型中所描述的存取路径来存取。

  而关系数据库的用户不必了解存取路径

关系数据语言的特点3

- 面向集合的存取方式

  关系数据语言的存取方式是面向集合的，操作对象是一个或者多个关系，操作结果也是一个关系，语言简练，操作方便

关系数据语言的特点4

- 两种使用形式

1.自含系统

某些关系数据语言不能与其他算法语言混合使用，因此称为自含系统

2. 宿主系统

某些关系数据语言可以嵌入到其他主语言中编成同一的程序，称为宿主系统

SQL语言概述

SQL语言是结构化的查询语言，是一种综合的、通用的、功能极强的关系数据库语言。它的功能包括：创建数据库对象、操纵对象、往数据库表中填充数据、在数据表中更新已存在的数据、删除数据、执行数据库查询、控制数据库访问权限和数据库总体管理。主要分为以下几类：

1、DDL数据定义语言

CREATE DATABASE	ALTER DATABASE
DROP DATABASE	CREATE TABLE
CREATE INDEX	ALTER TABLE
DROP TABLE	DROP VIEW
DROP INDEX	CREATE PROCEDURE
DROP PROCEDURE	CREATE RULE
DROP RULE	

2、DML数据操作语言

INSERT　UPDATE　DELETE

3、DQL数据查询语言

SELECT

4、事务控制命令

COMMIT	ROLLBACK
GRANT	REVOKE
SAVEPOINT	SET TRANSACTION

创建表

create table<表名>（<列名><数据类型>[列级完整性约束条件]

[, <列名><数据类型>[列级完整性约束条件]]

[,<表级完整性约束条件>];

```
Create table employee (emp_id       char(5) NOT NULL
UNIQUE,
                       emp_name char(20) NOT NULL,
                       emp_age int);
```

从现有的表中创建新的表

```
create table <新表名> as select [ * | <列名1>,<列名2>] from <旧表名> [where <条件>]

Create table employeebk
        select * from employee;
Create table employeebk
        select emp_name,emp_age from employee;
```

修改表

```
alter table <表名>
[ add <列名><数据类型>[gq列级完整性约束条件]]
[drop <完整性约束名>]
[modify <列名><数据类型>];
Alter table employee add(emp_salary
number(8));
Alter table emp_name modify emp_name
char(24);
Alter table employee drop unique(eno);
```

删除表

语法:

drop table <表名>

例如:删除表employee

Drop table employee ;

创建索引

```
Create [UNIQUE][CLUSTER] INDEX <索引名>
ON <表名>（<列名>[<次序>][,<列名>[<次序>]]...);
```

建立一个普通索引

```
Create index good_idx on 订购单（商品号）;
```

在多个字段上建立索引

```
Create index good_name on 订购单（商品号,生产厂家）
```

索引的类型

- 单列索引

只对表中的某一列进行索引

- 唯一索引

不允许在表中插入相同的值

- 复合索引

针对表中两个或两个以上的列建立的索引

- 单列索引与复合索引

在决定建定单列索引还是复合索引应当首先考虑在查询的where子句中用作过滤条件的列。

避免使用索引的情况

- 对于记录少的表不应当使用索引
- 当某些列用作为过滤条件时，如果数据行的返回率很高，对这样的列不应当使用索引
- 有些表需要进行频繁地、大批量的数据更新，但是索引又会使得数据更新的工作变得相当慢。解决这一问题的方法是删除索引再更新数据，然后再重建索引

- 对于那些有很多值为NULL的列不应当建立索引
- 对于那些需要经常操作的列不应当被索引，因为系统维护索引的工作会很繁重
- 特别提起注意的是：对某张表的特别长的键字应尽量避免使用索引，因为数据检索的速度会因为高的I/O开销而不可避免的变慢

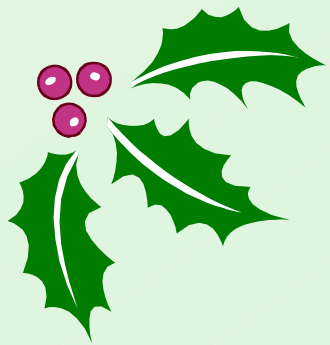
删除索引

语法：

drop index <索引名>

例如删除索引 good_idx

Drop index good_idx ;



创建视图

例如在职工表上为业务部的职式建立一个视图：

```
CREATE VIEW DEPT AS
SELECT EMP_NAME,EMP_SALARY FROM T_EMP
WHERE EMP_DEPT=“业务部”
```

```
Create view <视图名> as <子查询> with
check option;
```

更改视图的条件

- 视图必须未涉及连接
- 视图必须不包含GROUP BY 子句
- 视图不能包括对伪列 ROWNUM的任何引用
- 视图不能包括任何组合函数
- 不能使用 DISTINCT子句
- Where 子句不能包含表的嵌套引用，即在where子句中被引用过的表不能再被引用。

删除视图

语法:

drop view <视图名> [RESTRICT|CASCADE]

使用RESTRICT选项删除视图时，如果约束条件中有任何其它视图被引用，该命令误会出错；

使用CASCADE选项删除视图时，即使约束条件中有任何其它视图被引用，该命令也能成功执行并且基础视图或约束也同时被删除。

对数据库操作

对数据库有两个操作，即创建数据库命令和修改数据库命令：

Create Database

Alter Database

填充数据

语法：

INSERT INTO <表名>[（<属性名1><属性名2>...）]

[values([<常量1>][,<常量2>])...];

例如：添加一条新记录

Insert into employee.tbl

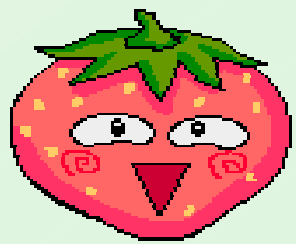
Values(‘ 00001’, ‘ 陈红’, ‘ 女’, ‘ 25’, ‘ 市场部’)

向表中插入有限列的值

```
Insert into employee  Values(‘ 00001’,
‘ 陈红’)
```

向表中插入空值

```
Insert into employee  Values(‘ 00001’,
‘ 陈红’, ‘ 女’, ’25’, NULL)
```



更新已存在的数据

语法：

update <表名>

Set <列名1>=<表达式1>[, <列名2>=<表达式2>]

[Where <条件>]；

例如：

Update employee set emp_dept=‘ 业务部’ where emp_name=‘ 陈红’;

删除数据

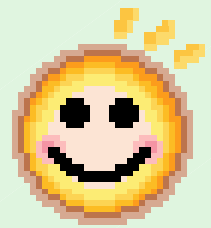
语法：

Delete from <表名>

[Where <条件>]；

例如：

Delete from employee where emp_name=‘陈红’;



数据查询语句

语法：

Select [all|distinct] <目标表达式>[,<目标表达式>]

from <表名或查询名> [,<表名或查询名>]

[Where <条件>]

[group by <列名> [having <条件表达式>]]

[order by <列名>[ASC | DESC]]

COUNT函数

语法：

```
Count[* | distinct 参数名| all 参数名]
```

例如：

```
Select count(*) from T_EMPLOYEE;
Select count(emp_phone) from
T_EMPLOYEE

Select count(distinct(emp_phone))
from T_EMPLOYEE
```

Sum 函数

语法：

Sum([distinct] 参数名)

例如：

```
Select sum(emp_salary) from
T_EMPLOYEE;
Select sum(distinct(emp_salary)) from
T_EMPLOYEE
```

AVG 函数

语法：

Avg([distinct]参数名)

例如：

```
Select avg(emp_salary) from
T_EMPLOYEE;
Select avg(distinct(emp_salary)) from
T_EMPLOYEE
```

Max 函数

语法：

max([distinct]参数名)

例如：

```
Select max(emp_salary) from
T_EMPLOYEE;
Select max(distinct(emp_salary))
from T_EMPLOYEE
```

Min 函数

语法：

min([distinct]参数名)

例如：

```
Select  min(emp_salary)  from
T_EMPLOYEE;
Select  min(distinct(emp_salary))
from T_EMPLOYEE
```