第 3 章　关系数据库标准语言——SQL

3.1　SQL 概述

SQL 是 Structured Query Language(结构化查询语言)的缩写。查询是 SQL 的重要组成部分,但不是全部,SQL 还包含数据定义、数据操纵和数据控制功能。SQL 是一个功能强大的关系数据库语言,SQL 语句已经成为关系数据库的标准语言,所以现在所有的关系数据库管理系统都支持 SQL。

3.1.1　SQL 的产生与发展

结构化查询语言是 1974 年由 Chamberilin 和 Boyce 提出的,美国 IBM 公司经过修改,将它用于自己的原型关系数据库系统 System R 中。在此基础上,IBM 公司将其商品化,并把它命名为 SQL。

核心 SQL 主要包括 4 个部分。

① 数据定义语言(DDL):用于定义 SQL 模式、基本表、视图、索引等结构。

② 数据操纵语言(DML):数据操纵分成数据查询和数据更新两类,其中数据更新又分成插入、删除和修改三种操作。

③ 数据控制语言(DCL):这一部分包括对基本表和视图的授权、完整性规则的描述、事务控制等内容。

④ 嵌入式 SQL 语言:涉及 SQL 语句嵌入在宿主语言程序中的规则。

3.1.2　SQL 的特点

SQL 语言具有如下主要特点。

① SQL 是一种一体化的语言,它包含了数据定义、数据查询、数据操纵和数据控制等方面的功能,可以完成数据库活动中的全部工作。

② SQL 语言是一种高度非过程化的语言,它没有必要告诉计算机“如何”去做,而只需要描述清楚用户要“做什么”,SQL 语言就可以将要求反馈给系统,自动完成全部工作。

③ SQL 语言非常简洁。虽然 SQL 功能很强,但它只有为数不多的几条命令,而且

SQL 语句的语法也非常简单，它很接近自然语言，因此容易学习和掌握。

④ SQL 语句可以直接以命令方式交互使用，也可以嵌入到程序语言中以程序方式使用。

⑤ SQL 语言采用集合操作方式，不仅查找结果可以是元组的集合，而且一次插入、删除、更新操作的对象也可以是元组的集合。

表 3.1 为 SQL 语言的功能。

表 3.1　SQL 语言的功能

SQL 功能	实现语句	SQL 功能	实现语句
数据定义	CREATE，DROP，ALTER	数据操纵	INSERT，UPDATE，DELETE
数据查询	SELECT	数据控制	GRANT，REVOKE

3.2　数据的定义

标准 SQL 的数据定义功能非常广泛，一般包括数据库的定义、表的定义、视图的定义、存储过程的定义、规则定义等，本节主要介绍模式、表的定义和视图的定义。表 3.2 为 SQL 的数据定义语句。

表 3.2　SQL 的数据定义语句

操作对象	操作方式		
	创建	删除	修改
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	
索引	CREATE INDEX	DROP INDEX	

3.2.1　模式的定义与删除

模式本身就是数据库中的一个对象。它可以使用 CREATE SCHEMA 语句（或者使用控制中心）显式地创建，并且将一个用户确定为模式的所有者。模式名称（Schema Name）会用来作为由两部分组成的对象名称的高端部分。

所有对象的名称都有模式部分和对象部分，但是并不总会全部显示这两个部分。显式使用了两个部分的对象名称会得到充分限定（Fully Qualifed），只使用一个部分的对象名称会由模式名称隐式限定（Implicitly Qualified）。当没有为数据库对象提供显式模式的时候，就会使用隐式限定。

1. 定义模式

SQL 模式可用 CREATE SCHEMA 语句定义，其基本句法如下。

```
CREATE SCHEMA <模式名> AUTHORIZATION <用户名>;
```

如果没有指定＜模式名＞，则＜模式名＞隐含为＜用户名＞。需要注意的是，要创建模式，调用该命令的用户必须拥有 DBA 权限，或者获得了 DBA 授予的 CREATE

SCHEMA 的权限。

【例 3.1】 定义一个商品-销售模式 SP_XS。

```
CREATE SCHEMA "SP_XS" AUTHORIZATION dbo;
```

注意：在 SQL Server 2005 中，给＜模式名＞加双引号和不加双引号都是可以的。

定义好的模式，可在数据库中“安全性”下面的“架构”中看到，如图 3.1 所示。

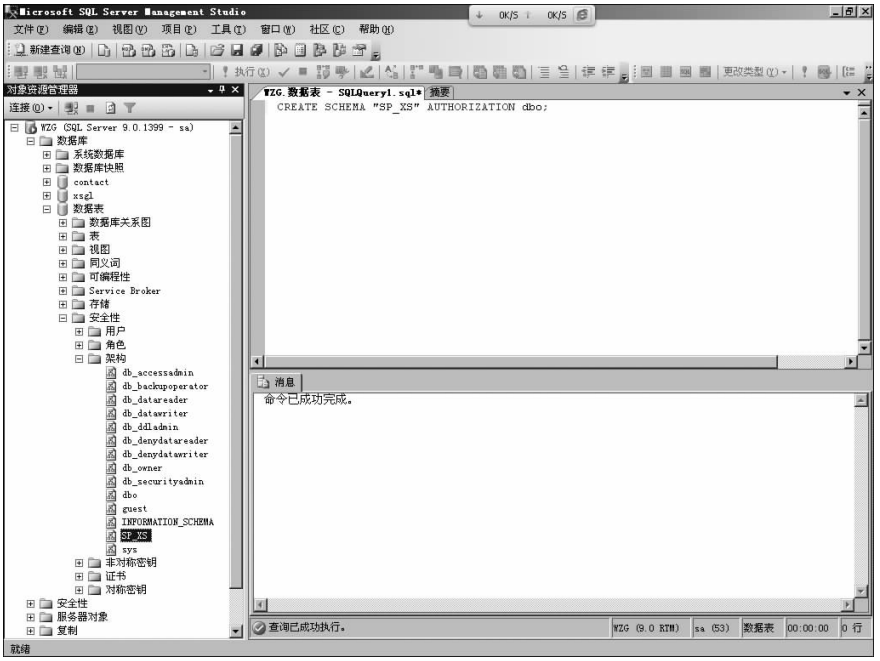

图 3.1 定义模式 SP_XS

模式实际上是一个命名空间，定义一个模式实际上也就是定义了一个命名空间，在这个命名空间中可以进一步定义该模式包含的数据库对象，例如基本表、视图、索引等。

2. 删除模式

删除模式使用 DROP SCHEMA 语句，DROP SCHEMA 语句的句法如下。

```
DROP SCHEMA<模式名>;
```

【例 3.2】 删除模式 SP_XS。

```
DROP SCHEMA SP_XS;
```

该语句将会删除模式 SP_XS。在 Microsoft SQL Server 2005 中要求被删除的架构不能包含任何对象，如果架构包含对象，则 DROP SCHEMA 语句将失败。

3.2.2 基本表的定义、删除与修改

1. 数据类型

数据类型是数据的一种属性，表示数据所表示信息的类型。任何一种计算机语言都

定义了自己的数据类型。不同的程序语言都具有不同的特点，所定义的数据类型的种类和名称都或多或少有些不同。以 SQL Server 为例，常用的数据类型如表 3.3 所示。

表 3.3　常用数据类型

数据类型	描　　述
Char(*n*)	长度为 *n* 的字符串
Varchar(*n*)	最大长度为 *n* 的可变长度字符串
Nchar(*n*)	长度为 *n* 的 Unicode 数据
Nvarchar(*n*)	最大长度为 *n* 的可变长度 Unicode 数据
Int	长整数
Numeric(*p*,*s*)	定点数，*p* 参数指示可以存储的最大位数，*s* 参数指示小数点右侧存储的最大位数
Smallmoney	短货币数据
Money	长货币数据
Float	浮动精度数字数据
bit	允许 0、1 或 NULL
Date	仅存储日期数据
Time	仅存储时间数据
Timestamp	存储唯一的数字，每当创建或修改某行时，该数字会更新
Timestamp	基于内部时钟，不对应真实时间
XML	存储 XML 格式化数据
Table	存储结果集，供稍后处理

2. 基本表的定义

表的定义可用 CREATE TABLE 语句，语句基本格式如下。

```
CREATE TABLE <表名>(列名 1 数据类型(宽度),列名 2 数据类型(宽度),…)
```

定义表中列的同时，还可以定义有关的完整性约束条件，例如为某列设置验证规则、验证信息和默认值等，此时 CREATE TABLE 语句的格式如下。

```
CREATE TABLE <表名>(列名 1 数据类型(宽度) PRIMARY KEY,
                    列名 2 数据类型(宽度) CHECK <规则>
                                          ERROR <信息>
                                          DEFAULT <默认值>…)
```

【例 3.3】 建立供货商表。

```
CREATE TABLE ghs (
      [ghsbh] [Varchar](6) ,
      [ghsmc] [Varchar] (20),
      [szd] [Varchar] (10),
      [lxdh] [Varchar] (20));
```

【例 3.4】 建立员工表，并同时设置相关完整性约束条件。

```
CREATE TABLE yg(
```

```
    [ygbh] [Varchar](6) NOT NULL,
    [xb] [Varchar](2) NOT NULL DEFAULT ('男'),
    [jbgz] [Money] NOT NULL,
    [zw] [Varchar](4) NULL,
    [mm] [Varchar](6) NOT NULL,
  PRIMARY KEY CLUSTERED ([ygbh] ASC)
);
```

3. 基本表的修改

基本表定义好后,有时需要对其进行修改,SQL 语言用 ALTER TABLE 语句修改基本表,ALTER TABLE 语句的一般格式为

```
ALTER TABLE <表名>
    [ ADD <新列名><数据类型>[完整性约束] ]
    [ DROP COLUMN <列名>]
    [ DROP <完整性约束>]
    [ ALTER COLUMN <列名><数据类型>];
```

其中<表名>指明要修改的基本表的名字,ADD 子句可用于增加新列和新的完整性约束条件,DROP 子句用于删除列或者完整性约束条件,ALTER COLUMN 子句用于修改原有的列定义,包括修改列名和数据类型。

【例 3.5】 为员工(yg)表增加家庭住址列。

```
ALTER TABLE yg ADD jtzz Varchar(100);
```

【例 3.6】 删除员工(yg)表中的家庭住址列。

```
ALTER TABLE yg DROP COLUMN jtzz;
```

【例 3.7】 将员工(yg)表中职务列(zw)的宽度改为 10。

```
ALTER TABLE yg ALTER COLUMN zw Varchar(10);
```

【例 3.8】 为员工(yg)表性别列(xb)增加完整性约束条件 xbcheck:性别只能是男或者女。

```
ALTER TABLE yg ADD CONSTRAINT xbcheck
   CHECK (xb='男' or xb='女');
```

增加的完整性约束条件 xbcheck,可在 yg 表的“约束”中看到如图 3.2 所示。

4. 基本表的删除

可以使用 DROP TABLE 语句删除已不需要的基本表,DROP TABLE 语句的一般格式为

```
DROP TABLE <表名>;
```

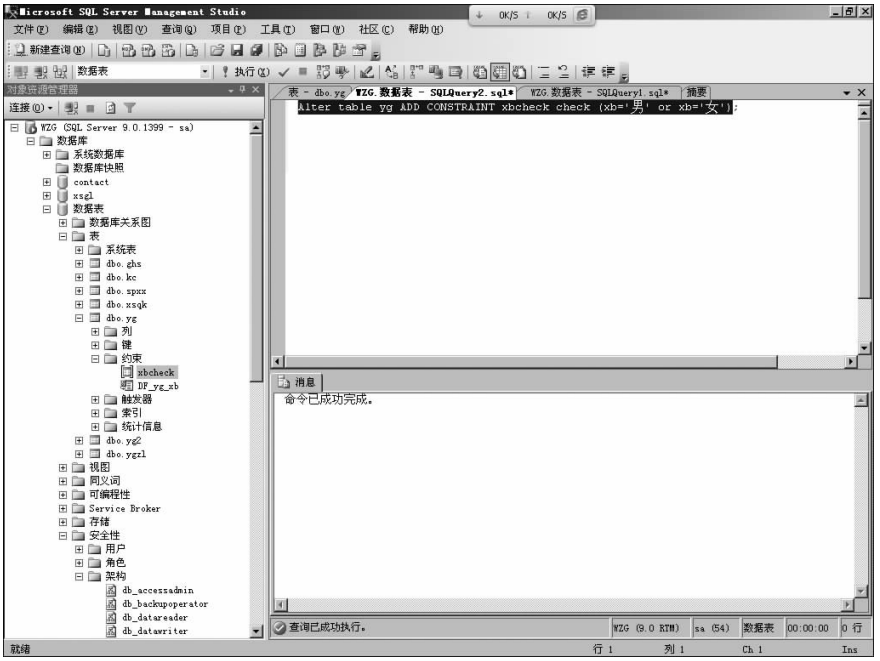

图 3.2　增加完整性约束条件 xbcheck

【例 3.9】 删除员工(yg)表。

```
DROP TABLE yg;
```

表被删除后,表的定义和表中的数据都将不复存在,因此,执行删除基本表的操作时需要慎重。

3.2.3　索引的定义与删除

索引是行位置的列表,按一个或多个指定列的内容来排序。索引通常用于加速对表的存取。索引数据可以与表数据存储在相同的表空间中,或存储在包含索引数据的单独表空间中。

索引的作用包括加快查询速度,这也是创建索引最主要的原因;通过创建唯一性索引,可以保证数据库表中每一行数据的唯一性。

1. 创建索引

在 SQL 语言中,建立索引使用 CREATE INDEX 语句,其一般格式为

```
CREATE [UNIQUE] [CLUSTER] INDEX <索引名>
    ON <基本表名>(<列名>[<次序>],[,<列名>[<次序>]]…);
```

其中,UNIQUE 表明要建立的是唯一索引,即每个索引值只对应唯一的数据记录。CLUSTER 表示建立的是聚集索引,聚集索引是指索引项的顺序与表中记录的物理顺序一致的索引组织,也就是说该索引中键值的逻辑顺序就是表中相应行的物理顺序。由于

聚集索引规定了数据在表中的物理存储顺序，因此一个表只能包含一个聚集索引。用户可以在最近常要查询的列上建立聚集索引，这样可以提高查询的效率。索引可以建立在一列或者多列之上，建立在多个列之上的索引可称为组合索引，各列名之间用逗号分隔。每个＜列名＞后面还可以用＜次序＞来指定索引值的升降排列次序，次序有 ASC(升序)或 DESC(降序)两种，缺省值为 ASC。

【例 3.10】 在商品信息(spxx)表的商品编号(spbh)列上创建一个唯一索引 PK_spxx。

```
CREATE UNIQUE INDEX PK_spxx ON spxx(spbh ASC);
```

【例 3.11】 在商品信息(spxx)表中基于商品名称(spmc)升序和销售单价(xsdj)降序建立组合索引 mc_dj，类型为唯一索引。

```
CREATE UNIQUE INDEX mc_dj ON spxx(spmc ASC, xsdj DESC);
```

需要注意的是，对于已含重复值的属性列不能建 UNIQUE 索引，对某个列建立 UNIQUE 索引后，插入新记录时数据库管理系统会自动检查新记录在该列上是否取了重复值，这相当于增加了一个 UNIQUE 约束。

2. 删除索引

索引建立好后，如果基本表中的数据发生变化，不需要用户的干预，数据库系统会自动维护索引，以保证索引的正确性。建立索引的目标是为了提高查询的效率，减少查询操作所需要的时间，但如果数据增加、删除、修改等操作很频繁，则系统需要花费很多时间来维护索引，此时可以删除一些不必要的索引，以减少系统维护索引所需要的开销。

在 SQL 中，用于删除索引的语句为 DROP INDEX，语句格式为

```
DROP INDEX <索引名>;
```

【例 3.12】 删除商品信息(spxx)表中的索引 mc_dj。

```
DROP INDEX spxx.mc_dj;
```

需要注意的是，DROP INDEX 语句格式中没有单独的＜表名＞，所以需要在索引名前加上表名，以指定是对哪个表进行操作。删除索引时，系统会从数据字典中删去有关该索引的描述。

3.3 数据查询

SQL 语言的核心是查询。SQL 语言的查询语句为 SELECT，它的基本形式由 SELECT…FROM…WHERE 查询块组成，多个查询块可以嵌套执行。SELECT 语句格式如下。

```
SELECT [ALL|DISTINCT] <目标列表达式>[,<目标列表达式>] …
       FROM <表名或视图名>[, <表名或视图名>] …
       [WHERE <条件表达式>]
       [GROUP BY <列名 1>[HAVING <条件表达式>]]
       [ORDER BY <列名 2>[ASC|DESC]];
```

SELECT ＜目标列表达式＞子句。指定要显示的属性列，ALL 和 DISTINCT 为二选一参数，ALL 表示输出所有满足条件的元组也就是行，DISTINCT 表示去掉重复的元组，默认为 ALL。

FROM ＜表名或视图名＞子句。指定查询对象（基本表或视图），也就是指定原始数据的来源，指明从哪个或者哪些对象中去查询满足条件的元组。

WHERE ＜条件表达式＞子句。指定查询条件。

GROUP BY ＜列名 1＞［HAVING ＜条件表达式＞］子句。对查询结果按指定列的不同取值分组，该属性列值相等的元组为一个组。进行分组的目的通常是为了在每组中使用聚集函数。HAVING 短语可用于筛选出满足指定条件的组。

ORDER BY ＜列名 2＞［ASC|DESC］子句。对查询结果按指定列值的升序或降序排序。

SELECT 语句用法非常灵活，可以实现一些简单的查询，也可以完成非常复杂的查询功能。下面我们从简单到复杂，逐步来学习 SELECT 语句的使用。

3.3.1　单表查询

所谓单表查询，是指查询的数据来源只有一张表或者视图，也就是 FROM ＜表名或视图名＞子句中指定的表名或视图名只有一个。

1. 指定要显示的属性列

指定查询结果中要显示的属性列，有两种情况，一是直接从原始数据中选择列，这相当于关系代数中的投影运算，二是通过表达式来得出需要显示的列。

1）查询选定列

表或视图中的属性列可以有许多个，很多情况下用户只需要查看其中的一部分列，这时可以通过在 SELECT 子句的＜目标列表达式＞中指定要显示的列名。

【例 3.13】 查询员工（yg）表中所有员工的编号、职务。

```
SELECT ygbh,zw FROM yg;
```

【例 3.14】 查询库存表（kc）表中所有库存商品的商品编号、进货价和库存数量。

```
SELECT spbh,jhj,kcsl FROM kc;
```

2）查询全部列

如果需要查询表或视图中的全部列，可使用 * 号指代，而不必逐一列出所有的列名。

【例 3.15】 查询员工（yg）表中所有列。

```
SELECT * FROM yg;
```

【例 3.16】 查询库存表（kc）表中所有列。

```
SELECT * FROM kc;
```

3）使用列别名改变查询结果的列标题

如果需要改变查询结果的列标题，可用 AS 关键字指定列标题。

【例 3.17】 查询员工(yg)表中所有员工的编号、职务,并且显示汉字列标题。

```
SELECT ygbh AS 员工编号, zw AS 职务 FROM yg;
```

4) 查询需要经过计算得出的列

SELECT 语句不仅具有我们通常所理解的从原始数据中按条件抽取数据得到查询结果的直接查询能力,而且还有对原始数据进行计算得到查询结果的计算查询能力,比如查询商品的平均价格、某种商品的最高单价等统计性数据,这些信息数据在表或视图中并没有直接存储,但可以通过表达式计算来实现查询输出。

【例 3.18】 查询商品信息(spxx)表中商品在国庆九五折促销活动中的促销价格,输出商品编号、商品名称和促销价格。

```
SELECT spbh,spmc,xsdj * 0.95 AS 促销价格 FROM spxx;
```

在表达式中除可以使用加减乘除等运算之外,还可以使用聚集函数,为增强查询功能,SQL 提供了许多聚集函数,主要有以下几种。

① 计数。

```
count([DISTINCT|ALL] * )          统计元组的个数
count([DISTINCT|ALL] <列名>)      统计某一列中不同值的个数
```

② 求和。

```
sum([DISTINCT|ALL] <列名>)        统计某一列中值的总和
```

③ 计算平均值。

```
avg([DISTINCT|ALL] <列名>)        统计某一列中值的均值
```

④ 求最大值。

```
max([DISTINCT|ALL] <列名>)        求某一列中的最大值
```

⑤ 求最小值。

```
min([DISTINCT|ALL] <列名>)        求某一列中的最小值
```

如果带 DISTINCT 短语,则在计算时要取消指定列中的重复值;带 ALL 短语,则不取消重复值。ALL 为缺省值。

【例 3.19】 查询员工(yg)表中的员工人数。

```
SELECT count(*) AS 员工人数 FROM yg;
```

【例 3.20】 统计销售情况表中的总销售金额。

```
SELECT sum (zje) AS 总销售金额 FROM xsqk;
```

2. 筛选元组

表或视图中的元组(行)可以有很多,有时我们并不需要查看所有的元组,此时可以通过筛选来选定需要查询输出的元组。

1）消除重复行

基本表中一般而言是不会出现重复行的，但本来并不完全相同的元组，经过列筛选（投影运算）后可能就变得相同了，此时可用 DISTINCT 来消除重复值。

【例 3.21】 查询商品信息（spxx）表中所有的商品名称品种。

```
SELECT DISTINCT spmc FROM spxx;
```

2）查询满足条件的元组

查询满足条件的元组，可以通过 SELECT 语句中的 WHERE ＜条件表达式＞ 子句来实现，WHERE 子句可包括各种条件运算符。

① 比较运算符（大小比较）。＞、＞＝、＝、＜、＜＝、＜＞、!＞、!＜。

② 范围运算符（表达式值是否在指定的范围）。

BETWEEN…AND…

NOT BETWEEN…AND…

如 Age BETWEEN 10 AND 30 相当于 Age＞＝10 AND Age＜＝30。

③ 列表运算符（判断表达式是否为列表中的指定项）。

IN（项 1，项 2，…）

NOT IN（项 1，项 2，…）

如 Country IN（'Germany'，'China'）。

④ 模式匹配符（判断值是否与指定的字符通配格式相符）。

LIKE、NOT LIKE

⑤ 空值判断符（判断表达式是否为空）。

IS NULL、NOT IS NULL

如 Age IS NULL。

⑥ 逻辑运算符（用于多条件的逻辑连接）。

NOT、AND、OR

逻辑运算符的优先级次序为 NOT、AND、OR。

模式匹配常用于模糊查找，它判断列值是否与指定的字符串格式相匹配，可用于 Char、Varchar、Text、Ntext、Datetime 和 Smalldatetime 等类型查询。

模式匹配可使用以下通配字符。

百分号％。可匹配任意类型和长度的字符，如果是中文，请使用两个百分号即％％。

下划线_。匹配单个任意字符，它常用来限制表达式的字符长度。

例如：

限制以“牙膏”结尾，使用 LIKE '％牙膏'。

限制以“王”开头，使用 LIKE '王％'。

限制以“王”开头除外，使用 NOT LIKE '王％'。

【例 3.22】 查询员工资料（ygzl）表中的刘姓员工人数。

```
SELECT count(*) AS 刘姓员工人数 FROM ygzl WHERE xm LIKE '刘%';
```

3. 排序

在 SQL 的 SELECT 语句中，使用 ORDERY BY 子句对查询结果排序，并可以用