

本节主要描述数据库的标准语言 SQL,包括 SQL 的功能和特点、常用语句的形式及多种风格的查询,介绍递归查询以及视图的概念与使用,讨论数据库访问,包括嵌入式 SQL 与数据库接口技术。

3.1 SQL 简介

3.1.1 SQL 的特征

SQL 是基于关系代数与关系演算的综合语言。它不仅具有丰富的查询功能,而且具有数据定义和数据控制功能,是集多种功能于一体的关系数据库标准语言。

1. SQL 的主要标准

SQL/86: 1986 年由美国国家标准化组织(ANSI)公布的 SQL 的第一个标准。

SQL/89(SQL1): 1989 年公布了 SQL 改进标准,该标准增强了完整性的语言特征。

SQL/92(SQL2): 该标准增加了许多新特征,如支持对远程数据库的访问,扩充了数据类型、操作类型、模式操作语言、动态 SQL 等。

SQL/99(SQL3): 该标准支持对象关系数据模型,扩展了递归查询、触发器,支持用户自定义数据类型、程序与流程控制等许多新的特征。

SQL/2003: 该标准扩展了 SQL 语法,增加了对互联网数据表示和交换的标准 XML、联机分析处理、取样等功能的支持。

SQL/2006: 该标准主要定义了 SQL 与 XML 间如何交互与应用。如何在 SQL 数据库中导入、存储 XML 数据,以及如何操纵、发布数据。增强了对 XML 数据处理的能力。

SQL 标准的制定与实施,屏蔽了不同数据库产品之间的差异,方便了用户使用,也为异构数据库互联互通奠定了基础。SQL 是数据库领域以至信息

领域中数据处理的主流语言。由于各关系数据库产品在实现标准 SQL 时各有差异,实际使用时,注意参见有关数据库产品的手册。本节主要介绍 SQL 的基本功能、常用语句形式。有些基于对象的特性及复杂类型将在 7.3 节介绍。

2. SQL 的功能特点

SQL 是一个综合的、通用的、功能极强,又简洁易学的语言。其主要特点如下:

(1) 综合统一。SQL 集数据定义、数据操纵、数据控制语言的功能于一体,风格统一,可以独立完成数据库生命周期中的全部操作,为使用数据库系统提供了良好的语言环境。

(2) 高度非过程化。SQL 语言是非过程化的、集合式的数据操作,用户只需提出“做什么”,而不必具体指明“怎么做”。存取路径的选择以及 SQL 语句的操作过程由系统自动完成。

(3) 灵活的使用方式。SQL 使用方式有命令式、程序式及嵌入式。它既能独立地用于联机交互方式,又扩展实现了过程化的程序控制与编程,还能够嵌入到高级语言程序中。

(4) 简洁、通用、功能强。SQL 吸取了关系代数、关系演算语言两者的特点和长处,故语言功能极强。而且 SQL 语法简单、易学易用,它已成为关系数据库的公共语言。由于它设计巧妙,语言十分简洁。

SQL 的数据查询、定义、操纵、控制的核心功能只用了 9 个动词,如表 3-1 所示。

表 3-1 SQL 的核心功能与动词

SQL 功能	核心动词
数据查询	SELECT
数据定义	CREATE, DROP, ALTER
数据操纵	INSERT, UPDATE, DELETE
数据控制	GRANT, REVOKE

3. SQL 数据库层次结构

数据库中的关系可分为 3 种类型。

(1) 基本表。实际存储的原始数据表(即实表)。

(2) 查询表。查询所得到的结果数据表。

(3) 视图。由基本表抽取出的表结构(即虚表)。

视图是数据库的一个重要概念。视图是从基本表或其他视图中导出的表,它本身不独立存储在数据库中,即数据库中只存放视图的结构定义而不存放视图对应的数据,因此视图是一个虚表。表与视图的联系如图 3-1 所示。

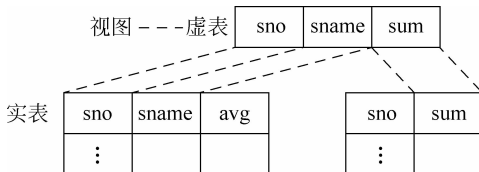


图 3-1 表与视图的联系

SQL 支持关系数据库的 3 级结构。其中,子模式对应于视图和部分查询表,模式对应于基本表,内模式对应于存储文件,SQL 数据库的 3 级结构如图 3-2 所示。

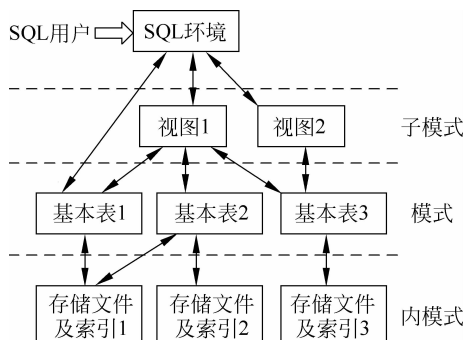


图 3-2 SQL 数据库的 3 级结构

在 SQL 中,数据库模式(简称模式或方案)不同于关系模式,它被定义为所拥有的表、视图、角色等对象的集合。一个数据库模式通常对应一个数据库用户。

SQL 环境(SQL Environment)是用户操作特定数据库的基本设施。SQL 环境包括设置或默认的目录和数据库模式、用户授权身份等。所有 SQL 语句,都是在某个 SQL 环境下对授权数据库进行操作的。

DBMS 为关系提供了一个 3 级命名方式。顶级由目录组成,目录是 SQL 环境中所有数据库模式的集合。每个目录中可包含若干数据库模式,而一个数据库模式里又可包含若干 SQL 的对象(如视图和关系等)。故要唯一定位一个关系,需要用 3 个部分组成,即

<目录名>.<模式名>.<表名>

为了在数据库中进行一次操作,用户或程序必须先连接到数据库。对应该连接,就会建立一个默认的目录和模式。这有点像刚登录到操作系统时当前目录被设置为用户主目录一样。若采用连接时默认的目录及默认的模式,则可以仅用表名来存取一个关系。

有了多目录和多模式,不同的应用程序和不同的用户可不必担心命名冲突的问题而独立工作。而且,便于一个应用程序的不同版本(成品版本和多个测试版本)在同一个数据库系统上运行。

3.1.2 SQL 数据类型

1. SQL 语句的分类

SQL 语言由 3 大部分组成:数据定义语言(Data Definition Language,DDL)用于定义数据库对象及其属性;数据操纵语言(Data Manipulation Language,DML)用于操作数据库对象中的数据;数据控制语言(Data Control Language,DCL)用于操控数据库对象的权限。按照 SQL 的功能,其语句可分为 4 大类。

(1) 模式定义语句。其主要功能是创建、更新、撤销各种数据库的对象,包括数据库模式、表、视图、索引、域、触发器、自定义类型等。

(2) 数据操纵语句。其主要功能是完成数据库的查询和更新操作。查询操作指对已存在的数据库中的数据按照指定的组合、条件表达式或次序进行各种查询检索。更新操作指

对数据库中的数据进行插入、修改和删除操作。

(3) 事务与控制语句。其主要功能是完成数据库授权、事务管理以及控制 SQL 语句集的运行。用来授予或回收访问数据库的某种特权、控制数据操纵事务的发生时间及效果及对数据库进行监视等。

(4) 会话及诊断语句。SQL 通过嵌入式和动态 SQL 规则规定了 SQL 语句在高级语言程序中的使用规范。会话规则可使应用程序连接到一个 SQL 服务器,并与之交互。这类语句主要功能是建立数据库连接、为 SQL 会话设置参数及获取诊断等。

2. SQL 的数据类型

在 SQL 中规定了 3 类数据类型。

(1) 预定义数据类型。

(2) 构造数据类型。

(3) 用户定义数据类型。

SQL 的数据类型说明及其分类如表 3-2 所示。

表 3-2 SQL 的数据类型及其分类表

分类	类型	类型名	说 明
预定义数据类型	数值型	INT	整数类型(也可写成 INTEGER)
		SMALLINT	短整数类型
		REAL	浮点数类型
		DOUBLE PRECISION	双精度浮点数类型
		FLOAT(n)	浮点数类型,精度至少为 n 位数字
		NUMERIC(p, d)	定点数类型,共有 p 位数字(不包括符号、小数点),小数点后面有 d 位数字
	字符串型	CHAR(n)	长度为 n 的定长字符串类型
		VARCHAR(n)	具有最大长度为 n 的变长字符串类型
	位串型	BIT(n)	长度为 n 的二进制位串类型
		BIT VARYING(n)	最大长度为 n 的变长二进制位串类型
	时间型	DATE	日期类型:年-月-日(形如 YYYY-MM-DD)
		TIME	时间类型:时:分:秒(形如 HH:MM:SS)
		TIMESTAMP	时间戳类型(DATE 加 TIME)
	布尔型	BOOLEAN	值为 TRUE(真)、FALSE(假)、UNKNOWN(未知)
	大对象	CLOB 与 BLOB	字符型大对象和二进制大对象数据类型值为大型文件、视频、音频等多媒体数据
构造数据类型	由特定的保留字和预定义数据类型构造而成,如用“ARRAY”定义的聚合类型,用“ROW”定义的行类型,用“REF”定义的引用类型等		
自定义数据类型	是一个对象类型,是由用户按照一定的规则用预定义数据类型组合定义的自己专用的数据类型		

说明:许多数据库产品还扩充了其他一些数据类型,如 TEXT(文本)、MONEY(货币)、GRAPHIC(图形)、IMAGE(图像)、GENERAL(通用)、MEMO(备注)等。

3.2 SQL 的数据定义

数据库中的关系集合必须使用数据定义语言向系统说明并创建。SQL 的数据定义语言不仅允许定义一组关系,还可以说明每个关系的相关信息。SQL 的数据定义包括对数据库模式(Schema)、关系(Table,基本表)、视图(View)、索引(Index)等的定义和撤销操作。

3.2.1 表的建立与删改

1. 表的建立

SQL 定义表结构的一般格式如下:

```
CREATE TABLE[模式名.]<表名>(<列名><数据类型>[列级完整性约束]
[,<列名><数据类型>[列级完整性约束]]...[,<表级完整性约束>])
```

其中,<表名>是所要定义的关系的名字,它可以由一个或多个属性列组成。每一个列的数据类型可以是预定义数据类型,也可以是用户定义数据类型。

建表的同时通常还可以定义与该表有关的完整性约束条件。这些完整性约束条件被存入系统的数据字典中,当用户操作表中数据时,由 DBMS 自动检查该操作是否违背这些完整性约束条件。

【例 3.1】 对于表 2-5 至表 2-7 所示的学生——选课数据库中的 3 个表结构,用 SQL 语言定义之。

学生表: student(sno,sname,sex,age,dept, place)

课程表: course(cno,cname,credit,cpno)

学生选课表: s_c(sno,cno,grade)

用 SQL 语句创建一个表应指出它放在哪个模式中,为简单起见,这里省略了模式名,即在默认的目录及模式下,student 表可定义如下:

```
CREATE TABLE student
(sno      NUMERIC(6),
sname    VARCHAR(8) NOT NULL,
age       NUMERIC(2)
sex       VARCHAR(2),
dept     VARCHAR(20),
place    VARCHAR(20),
PRIMARY KEY(sno));
```

这里用到了表完整性约束的主键子句: PRIMARY KEY(<列名>)。被定义为主键的列强制满足非空和唯一性条件。凡带有 NOT NULL 的列,表示不允许出现空值;反之,可出现空值。当首次用 CREATE TABLE 定义一个新表后,只是建立了一个无值的表结构。

类似地,可以定义学生——选课数据库中的 course 表如下:

```
CREATE TABLE course
(cno      VARCHAR(3),
```

```
cname    VARCHAR(20) NOT NULL,  
credit   NUMERIC(2),  
pcno     VARCHAR(3)  
PRIMARY KEY(cno));
```

对于学生—选课数据库中的 s_c 表,可定义如下:

```
CREATE TABLE s_c  
( sno      NUMERIC(6),  
  cno      VARCHAR(3),  
  grade    NUMERIC(3),  
  PRIMARY KEY(sno, cno) );
```

2. 表的删除

删除表的一般格式如下:

```
DROP TABLE <表名> [ CASCADE | RESTRICT]
```

当选用了任选项 CASCADE,在删除表时,该表中的数据、表本身以及在该表上所建的索引和视图将全部随之消失;当选用了任选项 RESTRICT 时,只有在先清除了表中的全部记录行数据,以及在该表上所建的索引和视图后,才能删除一个空表,否则拒绝删除表。

3. 表的扩充和修改

随着应用环境和应用需求的变化,有时需要修改已建立好的表,包括增加新列、修改原有的列定义或增加新的、删除已有的完整性约束条件等。

(1) 表中加新列。SQL 修改基本表的一般格式为:

```
ALTER TABLE <表名>  
ADD (<列名> <数据类型>, ... )
```

(2) 删除列。删除已存在的某个列的语句格式为:

```
ALTER TABLE <表名>  
DROP <列名> [ CASCADE | RESTRICT]
```

其中,CASCADE 表示在基表中删除某列时,所有引用该列的视图和约束也自动删除;RESTRICT 在没有视图或约束引用该属性时,才能被删除。

(3) 修改列类型。修改已有列类型的语句格式为:

```
ALTER TABLE <表名>  
MODIFY <列名> <类型>;
```

需要注意的是:新增加的列一律为空值;修改原有的列定义可能会破坏已有数据。

【例 3.2】 用 SQL 表示一组增、删、改操作。

① 设有建立的已退学学生表 st-quit,删除该表。

```
DROP TABLE st-quit CASCADE;
```

该表一旦被删除,表中的数据、此表上建立的索引和视图都将自动被删除。

② 在学生表 student 中增加“专业”、“地址”列。

```
ALTER TABLE student
ADD (subject VARCHAR(20),addr VARCHAR(20));
```

③ 将学生表 student 中所增加的“专业”列长度修改为 8。

```
ALTER TABLE student
MODIFY subject VARCHAR(8);
```

④ 把 student 表中的 subject、addr 列删除。

```
ALTER TABLE student DROP addr;
ALTER TABLE student DROP subject;
```

4. 模式与域类型的定义

模式由模式名或模式拥有者的用户名来确定,并包含模式中每一个元素(表、视图、索引等)的定义。定义了一个模式,就是定义了一个存储空间。在该存储空间的数据库对象全体,构成该模式对应的 SQL 数据库。该模式定义语句的格式为:

```
CREATE SCHEMA <模式名> AUTHORIZATION <用户名>
[<CREATE DOMAIN 子句>|<CREATE TABLE 子句>|<CREATE VIEW >|... ... ]
```

其中:创建模式需拥有 DBA 权限或获得 DBA 授予建模式的权限;默认的方括号内是在该模式中要创建的域、表和视图等子句,模式中的表、视图等也可在以后根据需要随时创建;若不指定<模式名>,则隐含为<用户名>。如:

```
CREATE SCHEMA Teaching_db AUTHORIZATION Hang
//表示创建一个教师数据库的模式 Teaching_db,属主为 Hang.
```

SQL 允许用户自定义属性值的范围,为其定义域名,并在不同的表定义中使用该域名,这就是用户自定义的域类型。其定义格式为:

```
CREATE DOMAIN <域名> [AS] <数据类型>
[DEFAULT <默认值>][<域约束> ]
```

域类型是用于建立用户自定义属性的一个特定数据类型,它由带有约束的数据类型和默认值一起构成。

【例 3.3】 建立一个建筑公司的数据库模式,名为 company,属主是 yin。它包含 3 个表: worker(职工)、assignment(任务分配)和 building(建筑物)。

```
CREATE SCHEMA company AUTHORIZATION yin
CREATE DOMAIN item_id NUMERIC(4) DEFAULT 0 CHECK (VALUE IS NOT NULL)
```

该语句创建了一个模式 company,属主为 yin,定义了一个域类型,名为 item_id,允许值为 4 位数字,默认值为零并限制其值非空。可在多个表中使用该域类型,无需重复定义,若以后改变了域类型定义,则该变化可自动施加到使用该域类型的所有表中。

下面在已建数据库模式中建表:

```
CREATE TABLE worker (
    worker_id item_id,           //用自定义的域类型说明属性
    worker_name VARCHAR (12),
```

```

    skill_type VARCHAR (8),
    supv_id NUMERIC(4),
    PRIMARY KEY(worker_id))
CREATE TABLE building (
    bldg_id item_id,
    bldg_address VARCHAR (20),
    type VARCHAR (9) DEFAULT 'office'
    CHECK (type IN ('office', 'warehouse', 'residence')), qlty_level NUMERIC (1),
    status NUMERIC (1) DEFAULT 1 CHECK (status > 0 and status < 4),
    PRIMARY KEY(bldg_id))
CREATE TABLE assignment (
    worker_id item_id ,
    bldg_id item_id ,
    start_date DATE,
    num_days INT ,
    PRIMARY KEY ( worker_id, bldg_id ),
    FOREIGN KEY (worker_id) REFERENCES worker ON DELETE CASCADE,
    FOREIGN KEY (bldg_id) REFERENCES building ON DELETE CASCADE );

```

本例用到了完整性约束常用的 3 种子句形式。

- ① 主键子句: PRIMARY KEY(<列名>)
- ② 外键子句: FOREIGN KEY(<列名>) REFERENCES[<表名>][<约束选项>]
- ③ 检验子句: CHECK(<约束条件>)

CHECK 子句规定了一个规则,即一个表的指定列只能取这个规则定义的集合之内的值。默认值用默认子句表达: DEFAULT <默认值>。关于完整性约束的具体设计将在 6.2.2 小节介绍。

3.2.2 索引的建立与删除

在基本表上建立一个或多个索引,可以提供多种存取路径,加快查找速度。SQL 新标准不主张使用索引,而是以在创建表时直接定义主键,一般系统会自动在主键上建立索引。有特殊需要时,建立与删除索引由数据库管理员 DBA(或表的属主)负责完成。

在基本表上可建立一个或多个索引,目的是提供多种存取路径,加快查找速度。建立索引的一般格式为:

```

CREATE [UNIQUE] [CLUSTER] INDEX <索引名>
ON <表名>(<列名 1>[ASC|DESC],<列名 2>[ASC|DESC], ... )

```

ASC 表示升序(默认设置),DESC 表示降序。

解释如下:

- ① UNIQUE 表示唯一索引,即此索引的每一个索引值只对应唯一的数据记录。
- ② CLUSTER 表示聚簇索引,即索引项的顺序与表中记录的物理顺序一致的有序索引。
- ③ 一个基本表上最多只能建立一个聚簇索引。更新聚簇索引列数据时,会导致表中记录的物理顺序的变更,代价较大。因此,对于经常更新的列不宜建立聚簇索引。

【例 3.4】 为学生一选课数据库中的 student、course、s_c 表建立索引。其中 student

表按学号升序建唯一索引, course 表按课程号升序建唯一索引, s_c 表按学号升序和课程号降序建唯一索引。

```
CREATE UNIQUE INDEX sindex ON student(sno);
CREATE UNIQUE INDEX cindex ON course(cno);
CREATE UNIQUE INDEX s_cindex ON s_c(sno ASC, cno DESC);
```

删除索引的一般格式为:

```
DROP INDEX [ON <表名>]<索引名>
```

若要删除按课程号所建立的索引, 则使用语句:

```
DROP INDEX cindex;
```

删除索引时, 系统会同时从数据字典中删去有关该索引的描述。

3.3 SQL 的数据查询

3.3.1 单表查询

SQL 的 SELECT 语句用于查询与检索数据, 其基本结构是以下的查询块:

```
SELECT <列名表 A>
FROM <表或视图名集合 R>
WHERE <元组满足的条件 F>;
```

上述查询语句块的基本功能等价于关系代数式 $\Pi_A(\sigma_F(R))$, 但 SQL 查询语句的表示能力大大超过该关系代数式。查询语句的一般格式为:

```
SELECT [ALL|DISTINCT] <目标列表表达式> [, <目标列表表达式> ] ...
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]];
```

整个 SELECT 语句的含义是:

① 根据 WHERE 子句的条件表达式, 从 FROM 子句指定的基本表或视图中找出满足条件的元组。

② 再按 SELECT 子句中的目标列表表达式, 选出元组中的列值形成结果表。

③ 如果有 GROUP 子句, 则将行选择的结果按 <列名 1> 的值进行分组, 列值相等的元组为一个组, 每个组产生结果表中的一条记录。如果 GROUP 子句带 HAVING 短语, 则只有满足指定条件的组才予输出。

④ 如果有 ORDER 子句, 则最终结果表还要按 <列名 2> 值的升序或降序排序。

SELECT 语句既可以完成简单的单表查询, 也可以完成复杂的连接查询和嵌套查询。下面仍以表 2-5 至表 2-7 中的学生—选课数据库为例说明 SELECT 语句的多种用法。

1. 单表的查询

单表查询是指仅涉及一个表的查询。

(1) 查询指定的列。有时用户只对表中的一部分属性列感兴趣,这可通过在 SELECT 子句的<目标列表表达式>中指定要查询的列来实现。它对应关系代数中的投影运算。

(2) 查询表中行。带有 WHERE 子句的 SELECT 语句,执行结果只输出使查询条件为真的那些记录值。查询条件可用多种形式表示:

① 比较和逻辑运算。WHERE 之后的查询条件中允许出现比较运算符 =、>(大于)、>=(大于等于)、<(小于)、<=(小于等于)、<>(不等于)和逻辑运算符 AND(与)、NOT(非)、OR(或)等。

② 谓词 BETWEEN。用于判断某值是否属于一个指定的区间。

③ 谓词 IN。用来查找列值属于指定集合的元组。其否定是 NOT IN。

④ 字符匹配(LIKE)。用来进行字符串的匹配。其一般格式如下:

```
[NOT] LIKE '<匹配串>' [ESCAPE '<换码字符>',]
```

其含义是查找指定的属性列值与<匹配串>相匹配的元组。<匹配串>可以是一个完整的字符串,也可以含有通配符%和_。其中,%(百分号)代表任意长度(长度可为0)的字符串;_(下划线)代表任意单个字符。

(3) ORDER BY 子句。该子句后可以跟多个排序的变量名,第一个变量为主序,下面依次类推。每一个排序列名后可用限定词 ASC(升序——默认设置)或 DESC(降序)声明排序的方式。

【例 3.5】 用 SQL 表示一组简单查询操作。

① 查询学生表中全体学生的情况,即查整个表(*代表指定表的所有列),表示为:

```
SELECT *  
FROM student;
```

② 查选了 004 号课程且成绩在 85~95 之间的学生号。

```
SELECT sno  
FROM s_c  
WHERE cno = '004' AND grade BETWEEN 85 AND 95;
```

③ 找出年龄小于 20 岁、籍贯是湖南或湖北的学生的姓名和性别。

```
SELECT sname, sex  
FROM student  
WHERE age < 20 AND place IN ('湖南', '湖北');
```

④ 查所有姓刘或名字中第二个字为“晓”字的学生姓名、学号和性别。

```
SELECT sname, sno, sex  
FROM student  
WHERE sname LIKE '刘%' OR sname LIKE '_ _晓%';
```

需注意的是:一个汉字占两个字符位,所以表示一个汉字需要用两个“_”。

⑤ 查询全体男学生的学号、姓名,结果按所在的院系升序排列,同一院系中的学生按年龄降序排列。

```
SELECT sno,sname
FROM student
WHERE sex = '男'
ORDER BY dept,age DESC;
```

2. 函数与表达式

(1) 聚集函数(Build-In Function)。为方便用户,增强查询功能,SQL 提供了许多聚集函数,主要有:

COUNT([DISTINCT ALL] *)	//统计元组个数
COUNT([DISTINCT ALL]<列名> ,)	//统计一列中值的个数
SUM ([DISTINCT ALL]<列名>)	//计算一数值型列值的总和
AVG ([DISTINCT ALL]<列名>)	//计算一数值型列值的平均值
MAX ([DISTINCT ALL]<列名>)	//求一列值中的最大值
MIN ([DISTINCT ALL]<列名>)	//求一列值中的最小值

SQL 对查询的结果不会自动去除重复值,如果指定 DISTINCT 短语,则表示在计算时要取消输出列中的重复值。ALL 为默认设置,表示不取消重复值。聚集函数统计或计算时一般均忽略空值,即不统计空值。

(2) 算术表达式。查询目标列中允许使用算术表达式。算术表达式由算术运算符+、-、*、/与列名或数值常量及函数所组成。常见函数有算术函数 INTEGER(取整)、SQRT(求平方根)、三角函数(SIN、COS)、字符串函数 SUBSRING(取子串)、UPPER(大写字符)以及日期型函数 MONTHS_BETWEEN(月份差)等。

(3) 分组与组筛选。GROUP BY 子句将查询结果表按某相同的列值来分组,然后再对每组数据进行规定的操作。对查询结果分组的目的是为了细化聚集函数的作用对象。如果未对查询结果分组,聚集函数将作用于整个查询结果。

分组与组筛选语句的一般格式:

```
<SELECT 查询块>
GROUP BY <列名>
HAVING <条件>
```

解释如下:

① GROUP BY 子句对查询结果分组,即将查询结果表按某列(或多列)值分组,值相等的为一组,再对每组数据进行统计或计算等操作。GROUP BY 子句总是跟在 WHERE 子句之后(若无 WHERE 子句,则跟在 FROM 子句之后)。

② HAVING 短语常用于在计算出聚集函数值之后对查询结果进行控制,在各分组中选择满足条件的小组予以输出,即进行小组筛选。

比较: HAVING 短语是在各组中选择满足条件的小组;而 WHERE 子句是在表中选择满足条件的元组。

【例 3.6】 函数与表达式的使用示例。

① 用聚集函数查询选修了课程的学生人数。

```
SELECT COUNT(DISTINCT sno)
```

FROM s_c;

学生每选修一门课,在 s_c 中都有一条相应的记录。一个学生可选修多门课程,为避免重复计算学生人数,必须在 COUNT 函数中用 DISTINCT 短语。

② 用聚集函数查选 001 号课并及格学生的总人数及最高分、最低分。

```
SELECT COUNT( * ),MAX(Grade),MIN(Grade)
FROM s_c
WHERE cno = '001'and grade >= 60;
```

③ 设一个表 tab(a,b),表的列值均为整数,使用算术表达式的查询语句如下:

```
SELECT a, b, a * b, SQRT(b)
FROM tab;
```

输出的结果是: tab 表的 a 列、b 列、a 与 b 的乘积及 b 的平方根。

④ 按学号求每个学生所选课程的平均成绩。

```
SELECT sno,AVG(grade) avg_grade      // 为 AVG(grade)指定别名 avg_grade
FROM s_c
GROUP BY sno ;
```

SQL 提供了为属性指定一个别名的方式,这对表达式的显示非常有用。用户可通过指定别名来改变查询结果的列标题。

分组情况及查询结果示意图如图 3-3 所示。若将平均成绩超过 90 分的输出,则只需在 GROUP BY sno 子句后加 HAVING avg_grade>90 短语即可。

	sno	cno	grade	查询结果： <table><tr><td>sno</td><td>avg_grade</td></tr><tr><td>000101</td><td>85</td></tr><tr><td>010101</td><td>80</td></tr><tr><td>.....</td><td></td></tr></table>	sno	avg_grade	000101	85	010101	80		
sno	avg_grade											
000101	85											
010101	80											
.....												
1组	000101	c1	90									
	000101	c2	85									
	000101	c3	80									
2组	010101	c1	85									
	010101	c2	75									
3组	020101	c3	70									
	⋮											

图 3-3 分组情况及查询结果示意图

⑤ 求学生关系中湖北籍男生的每一年龄组(不少于 30 人)共有多少人,要求查询结果按人数升序排列,人数相同时按年龄降序排列。

```
SELECT age,COUNT(sno) number
FROM student
WHERE sex = '男'AND place = '湖北'
GROUP BY age
HAVING number > 30
ORDER BY number,age DESC;
```

3.3.2 多表查询

1. 嵌套查询

在 SQL 语言中, WHERE 子句可以包含另一个称为子查询的查询,即在 SELECT 语句中先用子查询查出某个(些)表的值,主查询根据这些值再去查另一个(些)表的内容。子查询总是括在圆括号中,作为表达式的可选部分出现在条件比较运算符的右边,并且可有选择地跟在 IN、SOME(ANY)、ALL 和 EXIST 等谓词后面。采用子查询的查询称为嵌套查询。

【例 3.7】 嵌套查询示例。

① 找出年龄超过平均年龄的学生姓名。

```
SELECT sname                                //外层查询或父查询
FROM student
WHERE age >
      ( SELECT AVG(age)                      //内层查询或子查询
        FROM student);
```

说明: 子查询是嵌套在父查询的 WHERE 条件中的。子查询中不能使用 ORDER BY 子句,因为 ORDER BY 子句只能对最终查询结果排序。

嵌套查询一般的求解方法是先求解子查询,其结果用于建立父查询的查找条件。SQL 语言允许多层嵌套查询,使得人们可以用多个简单查询构成复杂的查询,从而增强 SQL 的查询能力。层层嵌套的方式构造程序正是 SQL 语言结构化的含义所在。

嵌套查询中,谓词 ALL、SOME 的使用一般格式如下:

<标量表达式><比较运算符> ALL | SOME(<表子查询>)

其中: $X > \text{SOME}(\text{子查询})$, 表示 X 大于子查询结果中的某个值; $X < \text{ALL}(\text{子查询})$, 表示 X 小于子查询结果中的所有值。此外, $= \text{SOME}$ 等价于 IN; $< > \text{ALL}$ 等价于 NOT IN。

② 找出平均成绩最高的学生号。

```
SELECT sno
FROM s_c
GROUP BY sno
HAVING AVG(grade) >= ALL
      ( SELECT AVG(grade)
        FROM s_c
        GROUP BY sno);
```

2. 条件连接查询

通过连接使查询的数据从多个表中取得。查询中用来连接两个表的条件称为连接条件,其一般格式如下:

[<表名 1>·<列名 1> <比较运算符> [<表名 2>·<列名 2>]

连接条件中的列名也称为连接字段。连接条件中的各连接列的类型必须是可比的,但不必是相同的。当连接条件中比较的两个列名相同时,必须在其列名前加上所属表的名字

和一个圆点“.”以示区别。表的连接除=外,还可使用比较运算符<>、>、>=、<、<=以及 BETWEEN、LIKE、IN 等谓词。当连接运算符为=时,称为等值连接。

【例 3.8】 条件连接查询示例。

① 查询全部学生的学生号、学生名和所选修课程号及成绩。

```
SELECT student.sno, sname, cno, grade
FROM student, s_c
WHERE student.sno = s_c.sno;
```

② 查询选修了数据库原理课程的学生学号、成绩。用嵌套查询表示:

```
SELECT sno, grade
FROM s_c
WHERE cno IN
    ( SELECT cno
      FROM course
      WHERE cname = '数据库原理' );
```

上述语句可以用下面的条件连接查询语句表示:

```
SELECT sno, grade
FROM s_c, course
WHERE s_c.cno = course.cno AND cname = '数据库原理';
```

若上例还要查询输出课程名、学分,用条件连接查询语句很容易实现,而用嵌套查询则难以实现,但一般嵌套查询效率较高。

③ 按平均成绩的降序给出所有课程都及格的学生(号、名)及其平均成绩,其中成绩统计时不包括 008 号考查课。

```
SELECT student.sno, sname, AVG( grade ) avg_g
FROM student, s_c
WHERE student.sno = s_c.sno AND cno <> '008'
GROUP BY sno
HAVING MIN( grade ) >= 60
ORDER BY avg_g DESC ;
```

连接操作不仅可以在两个表之间进行,也可以是一个表与其自己进行连接,这种连接称为自身连接,涉及的查询称自身连接查询。

④ 找出年龄比“王迎”同学大的学生的姓名及年龄。

```
SELECT s1.sname, s1.age
FROM student s1, student s2          //s1、s1 在此用作设置的元组变量
WHERE s1.age > s2.age AND s2.sname = '王迎';
```

3. 相关子查询

当子查询的判断条件涉及外层父查询的属性时,称为相关子查询。相关子查询要用到存在谓词 EXISTS 和 NOT EXISTS,或者 ALL、SOME 等。

【例 3.9】 相关子查询示例。

① 查询所有选修了 005 号课程的学生姓名和学号。

在带存在谓词的子查询中,产生逻辑值谓词 EXISTS 作用是若内层查询结果非空,则外 WHERE 子句返回真;否则返回假。由 EXISTS 引出的子查询,其目标列表达式通常用“*”表示,因为带 EXISTS 的子查询只返回真值或假值,给出列名无实际意义。

相关子查询的一般处理过程是：首先取外层查询表中的第一个元组，根据它与内层查询相关的属性值一一进行判断，若 WHERE 子句返回值为真，则取此元组放入结果表；然后再取外层查询表的下一个元组；重复这一过程，直至外层表全部检查完为止。

与 EXISTS 谓词相对应的是 NOT EXISTS 谓词,其含义是:若内层查询结果为空,则外层的 WHERE 子句返回真值,否则返回假值。

② 查询没选修 001 号课程的学生学号及姓名。

方法 1:

```
SELECT sno,sname
FROM student
WHERE NOT EXISTS
( SELECT *
FROM s_c
WHERE s_c.sno = student.sno AND cno = '001');
```

方法 2:

```
SELECT sno, sname
FROM student
WHERE sno <> ALL
      ( SELECT sno
        FROM s_c
        WHERE cno = '001' );
```

一些带 EXISTS 或 NOT EXISTS 谓词的子查询不能被其他形式的子查询等价替换, 但所有带 IN 谓词、比较运算符、SOME 和 ALL 谓词的子查询都能用带 EXISTS 谓词的子查询等价替换。

③ 找出选修了全部课程的学生姓名。

在 SQL 中只有存在测试谓词 EXISTS 和 NOT EXISTS。然而任何一个带全称量词的谓词可以转换为等价的带存在量词的谓词： $(\forall x)p \equiv \neg(\exists x(\neg p))$ 。

可以用双嵌套 NOT EXISTS 来实现带全称量词的查询。本查询可以改为：查询这样一些学生，没有一门课程是他不选修的。

[illegible]

```

WHERE NOT EXISTS          //否定某学生选某课
  (SELECT *               //某学生选某课
   FROM s_c
   WHERE s_c.sno = student.sno AND s_c.cno = course.cno));

```

4. 集合运算

在 SQL 中提供了集合运算谓词：UNION、INTERSECT、EXCEPT。

【例 3.10】 设有某学院学生的 3 张表：一般研究生表 st1、在职研究生表 st2、全体研究生中的学生干部表 st3，其关系模式同 student 表。

① 查询全部研究生情况。

```

(SELECT * FROM st1)
UNION
(SELECT * FROM st2);

```

② 查询非干部的在职研究生的学生情况。

```

(SELECT * FROM st2)
EXCEPT
(SELECT * FROM st3);

```

③ 求选修了 001 或 002 号而没有选修 003 号课程的学生号。

```

(SELECT sno
 FROM s_c
 WHERE cno = '001' OR cno = '002')
EXCEPT
  (SELECT sno
   FROM s_c
   WHERE cno = '003');

```

3.3.3 连接查询

在 SQL 中扩展了关系代数的连接操作概念，提供了一组连接运算谓词，可以实现自然连接(Natural Join)、内连接(Inner Join，即等值的条件连接)、外连接(Outer Join)、合并连接(Union Join)、交叉连接(Cross Join，即笛卡儿积)等。这里仅介绍常用的自然连接与外连接。

1. 自然连接

SQL 的自然连接与关系代数中的自然连接功能是一样的。

【例 3.11】 查询在计算机学院且课程成绩 90 分以上的学生档案及其成绩情况。

可用自然连接实现该例。

```

SELECT *
FROM student NATURAL JOIN s_c
WHERE dept = '计算机学院' AND grade >= 90;

```

2. 外连接

外连接在结果中加上了每个关系中没有和另外一个关系中的元组相连的剩余元组，称之为悬浮元组。外连接生成的关系模式是两个关系模式的并集，允许在结果表中保留不能连接的悬浮元组，空缺部分填以 NULL。外连接的作用是在做连接操作时避免丢失信息。

外连接的连接类型定义了关系之间连接条件不匹配元组的处理方式,有以下 3 类:

- (1) 左外连接(LEFT [OUTER] JOIN),其结果表中保留左关系的所有元组。
- (2) 右外连接(RIGHT [OUTER] JOIN),其结果表中保留右关系的所有元组。
- (3) 全外连接(FULL [OUTER] JOIN),其结果表中保留左右两关系的所有元组。

【例 3.12】 设有教师和任职两个关系表如表 3-3 所示,其关系模式如下:

teacher(教师号,姓名,所属大学,职称)

post(编号,姓名,职务)

表 3-3 两个关系表

teacher 表				post 表		
教师号	姓名	所属大学	职称	编号	姓名	职务
20064	赵雨林	武汉大学	教授	020	赵雨林	副市长
21089	李岗	华中科大	教授	103	李岗	科委主任
22089	韩英星	华中师大	副教授	204	吴倩	计委书记
20076	张万象	地质大学	讲师	005	刘长江	副局长

若执行操作:

```
SELECT *
FROM teacher FULL OUTER JOIN post      //全外连接
ON teacher. 姓名 = post. 姓名;
```

则该运算的结果包括 3 种类型的元组:

第一种元组描述的是既是教师又兼有职务的人,即两个关系的自然连接的结果集。

第二种元组描述的是未兼职的教师。这些元组是那些在 teacher 表的所有属性上有对应值,而只属于 post 表的编号,职务属性的对应值为 null。

第三种元组描述非教师的任职人员。这些元组是那些在 post 的所有属性上有对应值,而只属于 teacher 表的教师号,所属大学,职称属性的对应值为 null。

有关执行结果以表 3-4 所示。

若执行以下操作:

```
SELECT *
FROM teacher LEFT OUTER JOIN post      //左外连接
ON teacher. 姓名 = post. 姓名;
```

则该运算结果只包括上述第一、第二种类型的元组。

表 3-4 teacher 表和 post 表的全外连接

	姓名	教师号	所属大学	职称	编号	职务
第一种 {	赵雨林	20064	武汉大学	教授	020	副市长
	李岗	21089	华中科大	教授	103	科委主任
第二种 {	韩英星	22089	华中师大	副教授	NULL	NULL
	张万象	20076	地质大学	讲师	NULL	NULL
第三种 {	吴倩	NULL	NULL	NULL	204	计委书记
	刘长江	NULL	NULL	NULL	005	副局长

3.3.4 递归查询

1. 递归合并查询

在实际应用中,会涉及许多递归问题,如求组成零件的子零件、求树的子树等问题。这类问题可用递归合并查询来解决。递归合并查询是 SQL 的一个重要新特性(但非核心部分)。这里仅介绍线性递归,即最多有一个递归子目标。

在 SQL 中,可以由关键字 WITH 引导的递归合并查询语句来定义新的关系,然后就可以在 WITH 语句内部使用这些定义。

递归合并查询语句的简单形式如下:

```
WITH RECURSIVE <临时表 R> AS
<R 的定义>
UNION
<涉及 R 的查询>
<递归结果查询>
```

这就是说,可以定义一个名称为 R 的临时关系,然后再在某个查询中使用 R。

定义中的任何一个关系都可以是递归的,它们之间也可以是相互递归的,也就是说,每个关系都可以根据某个其他关系(包括它本身在内)来定义。

【例 3.13】 设有一个航班表 flight 记录了成都、西安、北京、广州、上海、武汉之间的航班信息,如表 3-5 所示。现要求求出:能从一个城市飞到另一个城市的城市对集合(含直接到达和间接中转到达)。

表 3-5 flight 表

airline	from	to	depart	arrive
南航	武汉	成都	8:00	9:40
国航	成都	西安	9:30	12:30
东航	成都	北京	9:00	14:30
东航	北京	广州	15:30	19:30
东航	北京	上海	15:00	17:30
国航	广州	上海	18:30	21:30

设:以下式中的航线 a、起飞时间 d、到达时间 r 以及表示起点或终点的 f、t、t1 都是列变量。定义要求的结果集为关系 reach(f,t),以下表达式描述了结果关系:

```
①reach(f,t) = flight(a,f,t,d,r)           //f,t 在表的同一个元组中
②reach(f,t) = flight(a,f,t1,d,r) AND reach(t1,t)
//f,t 不在表的同一个元组中
```

上述的定义是线性右递归的。式子①表明 reach 包含有能从第一个城市直达另一个城市的城市对。式子②表明,如果能从 f 城市直达 t1 城市,并能从 t1 城市到达 t 城市,则就能从 f 城市到达 t 城市。

上述问题用 SQL 写出的相应查询语句如下:

```
WITH RECURSIVE reach(from,to) AS
```

```

        (SELECT from,to                                //选出直接到达城市对
        FROM flight)
    UNION
    (SELECT flight.from,reach.to                        //选出间接到达城市对
    FROM flight,reach
    WHERE flight.to = reach.from)
SELECT * FROM reach;
```

2. 空值处理

在许多情况下,必须对元组的分量赋值,但又不能给出具体的值。例如,在一个学生档案表中,有奖惩属性,但是有些学生尚没有任何奖惩,这该怎么办呢? 这时可将奖惩属性标以空值(NULL)。

SQL 用空值标识某属性值的信息缺失。空值不同于空白或零值,可以有多种不同的解释,如值暂时未知、值需隐瞒、无合适的值等。

空值会在多种情况下产生。例如,当向关系中插入元组时,如果所用的命令只提供元组的某些分量而不是所有的分量,那么,对那些没有指定值的分量若没有默认值,则就会是空值;当进行外部连接运算时,也会产生大量的空值。

测试属性的值是否为空值的方法是使用关键字 IS NULL 和 IS NOT NULL。在 SQL 中只有一个特殊的地方——WHERE 查询条件中允许查询空值,形式如下:

```
WHERE <列名> IS [NOT] NULL
```

在 SQL 中不能使用条件表示: <列名>=NULL。

【例 3.14】 某些学生选修课程后没有参加考试,所以有选课记录,但无考试成绩。查询缺少选课成绩的学生号和相应的课程号。

```

SELECT sno,cno
FROM s_c
WHERE grade IS NULL;
```

说明: 空值不是常量。不能显式地将空值作为操作数使用;空不能用来表示默认值、不可用值;尽量少使用空值,大部分聚集函数忽略空值。

3.4 SQL 的数据更新

SQL 中数据更新包括插入数据、修改数据和删除数据。向数据库表插入新行用 INSERT 语句。

3.4.1 插入数据

1. 用子句向表中插入数据

用 VALUES 子句向表中插入数据的语句格式如下:

```

INSERT INTO <表名> [(<列名 1>[,<列名 2>] ... ...)]
VALUES(<常量 1>[,<常量 2>] ... ...)
```

其功能是将新元组插入指定表中。其中新元组属性列 1 的值为常量 1, 属性列 2 的值为常量 2……如果某些属性列在 INTO 子句中没有出现, 则新元组在这些列上将取空值。

如果 INTO 子句中没有指明任何列名, 则新插入的记录必须在每个属性列上均有值。

2. 用子查询向表中插入数据

用子查询向表中插入数据的语句格式如下:

```
INSERT INTO <表名> [(<列名 1>[, <列名 2>] ...)]  
<查询语句>;
```

其功能是以批量插入的方式, 一次将查询的结果全部插入指定表中。

【例 3.15】 分别用子句和子查询向表中插入数据。

① 把新转学来的学生王晓雪的记录加入到学生表 student 中。

```
INSERT INTO student  
VALUES(200510, '王晓雪', 20, '女', '英语系', '湖北省');
```

② 设已建研究生复试表 st1_grade, 平均成绩优秀的学生将免试成为研究生, 将这些学生的学号、姓名及平均成绩插入研究生复试表。

```
INSERT INTO st1_grade(gno, name, avg)  
SELECT student.sno, sname, AVG(grade) avg_good  
FROM student, s_c  
WHERE student.sno = s_c.sno  
GROUP BY sno  
HAVING avg_good >= 90;
```

3.4.2 删改数据

1. 用子句修改表中数据

修改表中的记录用 UPDATE 语句。用 SET 子句修改表中数据的语句格式如下:

```
UPDATE <表名>  
SET <列名> = <表达式> [, <列名> = <表达式>]  
[WHERE <条件表达式>]
```

其功能是修改指定表中满足 WHERE 条件的元组。其中, SET 子句用于指定修改值, 即用表达式的值取代相应的属性列值。如果省略 WHERE 子句则表示要修改表中的所有元组。

2. 用子查询修改表中数据

子查询可以嵌套在修改语句中, 用以构造执行修改操作的条件。

【例 3.16】 分别用子句、子查询修改表中数据。

① 把计算机学院所有学生的年龄增加 1 岁。

```
UPDATE student  
SET age = age + 1  
WHERE dept = '计算机学院';
```

② 学生“左彼”在 001 号课程考试中作弊,该课成绩应作零分计。

```
UPDATE s_c
SET grade = 0
WHERE cno = '001' AND sno =
    (SELECT sno
     FROM student
     WHERE sname = '左彼');
```

③ 当某学生 008 号课程的成绩低于该门课程的平均成绩时,提高 5%。

```
UPDATE s_c
SET grade = grade * 1.05
WHERE cno = '008' AND grade <
    (SELECT avg(grade)
     FROM s_c
     WHERE cno = '008');
```

这里在内层子句中使用了外层 UPDATE 子句中出现的表名 s_c,但这两次使用是不相关的,即这个修改语句执行时,先执行内层 SELECT 语句,然后再对查找到的元组执行修改操作,而不是边找元组边修改。在插入语句和删除语句遇到类似情况时,也是如此处理。

3. 删除数据

删除数据用 DELETE 语句,其语句一般格式如下:

```
DELETE
FROM <表名>
[WHERE <条件表达式>]
```

功能是从指定表中删除满足 WHERE 条件的所有元组。如果省略 WHERE 子句,表示删除表中的全部元组。子查询同样也可以嵌套在删除语句中,用以构造执行删除操作的条件。

【例 3.17】 删除学号为 990081 的学生信息。

```
DELETE
FROM student
WHERE sno = '990081';
```

需注意的是:若表 s_c 未定义外键及其约束选项,此操作执行后,学生表 student 中学号为“990081”的学生信息不存在了,但在学习表 s_c 中,该学生的相关学习信息仍在,破坏了数据的完整性,故需进行相关数据的显式删除。

说明: 插入数据时,表定义中说明了 NOT NULL 的列不能取空值;若已定义主键、外键及其约束选项,则向参照表中插入、删除元组时,关系 DBMS 会自动保证实体完整性、参照完整性;否则,删除时需用户维护数据一致性。

3.5 SQL 中的视图

3.5.1 视图的概念

1. 视图与表的关系

(1) 视图(View)是从一个或几个基本表(或视图)导出的表。视图是以现存表的全部或部分内容建立起来的一个表。视图非物理存在,它不包含真正存储的数据,其内容不占存储空间。因此视图也被称为虚表,而真正物理存在的表称为基本表或实表。

(2) 视图是用户用来查看数据的窗口。任何对基表中所映射的数据更新,通过该窗口在视图可见的范围内都可以自动和实时地再现。

(3) 对视图的一切操作最终将转换为对基本表的操作,即任何对视图允许的更新也将自动和实时地在相应基表所映射的数据上实现。

(4) 更新视图有限制。视图与表一样可被查询、删除,即可以像一般的表那样操作,但对视图的更新操作有一定的限制。

2. 视图的作用

视图最终是定义在基本表之上的,对视图的一切操作最终也要转换为对基本表的操作。既然如此,为什么还要定义视图呢?这是因为视图有以下作用:

(1) 简化结构及复杂操作。视图机制使用户可以将注意力集中在他所关心的数据上。使用户眼中的数据库结构简单、清晰,并且可以简化用户的复杂查询操作。

(2) 多角度地、更灵活地共享。视图机制能使不同的用户从多种角度以不同的方式看待同一数据。当许多不同种类的用户使用同一个数据库时,这种灵活性是非常重要的。

(3) 提高逻辑独立性。由于有了视图机制,所以当数据库重构时,有些表结构的变化,如增加新的关系、结构的分解或对原有关系增加新的属性等,用户和用户程序不会受影响。视图对重构数据库提供了一定程度的逻辑独立性。

(4) 提供安全保护。有了视图机制,就可以在设计数据库应用系统时对不同的用户定义不同的视图,使机密数据不出现在不应看到这些数据的用户视图上,这样就由视图机制自动提供了对机密数据的安全保护功能。

3.5.2 创建与使用视图

1. 创建视图

SQL 语言用 CREATE VIEW 命令建立视图,其一般格式为:

```
CREATE VIEW <视图名> [(<列名 1>[,<列名 2>] ...)]  
AS <子查询>  
[WITH CHECK OPTION]
```

其中,子查询可以是任意复杂的 SELECT 语句。

【例 3.18】 创建视图示例。

(1) 建立一个只包括女学生学号、姓名和年龄的视图 s_st。

```
CREATE VIEW s_st(no,name,age)
```

```
AS SELECT sno, sname, age
FROM student
WHERE sex = '女';
```

系统执行 CREATE VIEW 语句的结果只是把视图的定义存入数据字典,并不执行其中的 SELECT 语句。只是在对视图查询时,才按视图的定义从基本表中将数据查出。因基本表随着更新操作在不断变化,所以视图对应的内容是实时、最新的内容,并非总是视图定义时的内容。

WITH CHECK OPTION 子句表示对视图进行 UPDATE、INSERT 和 DELETE 操作时要保证更新、插入或删除的行满足视图定义中子查询中的条件。

(2) 建立信息院学生的视图,并要求进行修改和插入操作时仍需保证该视图只有信息院的学生记录。

```
CREATE VIEW d_student
AS SELECT *
FROM student
WHERE dept = '信息院'
WITH CHECK OPTION;
```

由于在定义 d_student 视图时加上了 WITH CHECK OPTION 子句,以后对该视图进行插入、修改和删除操作时,DBMS 会自动加上 dept='信息院'的条件。

如果 CREATE VIEW 语句仅指定了视图名,省略了组成视图的各个属性列名,则表示该视图由子查询中 SELECT 子句目标列中的诸属性列组成。但在下列 3 种情况下必须明确指定组成视图的所有列名:

- ① 某个目标列不是单一的属性名,而是集函数或列表表达式。
- ② 多表连接时选出了几个同名列作为视图的属性。
- ③ 需要在视图中为某个列启用新的更合适的名字。

视图创建总是包括一个查询语句 SELECT。可以利用 SELECT 语句从一个表中选取所需要的行或列构成视图,也可以从几个表中选取所需要的行或列(使用子查询和连接查询方式)构成视图。

【例 3.19】 从学生表、课程表和选课表中产生一个视图 s_grade,它包括学生号、姓名、课程名和成绩。

```
CREATE VIEW s_grade
AS SELECT student.sno, sname, cname, grade
FROM student, course, s_c
WHERE student.sno = s_c.sno AND course.cno = s_c.cno;
```

若需再建立一个选修了课程且成绩在 90 分以上的学生的视图。因为已定义了一个关于学生选课情况的视图 s_grade,故可在该视图基础上再建新视图。

```
CREATE VIEW s_good
AS SELECT *
FROM s_grade
WHERE grade >= 90;
```

以下是一些特殊视图：

(1) 行列子集视图是指视图是从单个基本表导出,并且只是去掉了基本表的某些行和某些列,但保留了主键的视图。

(2) 分组视图是指带有集函数和 GROUP BY 子句查询所定义的视图。

(3) 带虚拟列的视图是指设置了一些基本表中并不存在的虚拟列(派生列)的视图,这种视图也称为带表达式的视图。

定义基本表时,为了减少数据库中的冗余数据,表中只存放基本数据,由基本数据经过各种计算派生出的数据一般是不存储的。由于视图中的数据并不实际存储,所以定义视图时可以根据应用的需要,设置一些派生属性列。

【例 3.20】 将学生的学号及其选课的平均成绩定义为一个视图。

```
CREATE VIEW s_avg(sno,gavg)    //gavg 是基本表中不存在的派生属性列
AS SELECT sno,AVG(grade)
FROM s_c
GROUP BY sno ;
```

例 3.20 是一个带虚拟列的分组视图。视图不仅可以建立在一个或多个基本表上,也可以建立在一个或多个已定义好的视图上,或建立在基本表与视图上。

有了视图机制,数据独立性大为提高,不管基本表是扩充还是分解均不影响对概念数据库的认识。只需重新定义视图内容,而不改变面向用户的视图形式,因而保持了关系数据库逻辑上的独立性。

【例 3.21】 设有学生关系 stud(sno,sname,sex,age,dept,...,degree),根据需要分为了两个表:

简表: st1(sno,sname)

简历表: st2(sno,sex,age,dept,...,degree)

此时原基本表 stud 是 st1 和 st2 自然连接的结果,这时可建立一个视图 st-view 以得到原基本表的效果:

```
CREATE VIEW st-view
AS SELECT st1.sname,st2.*
FROM st1,st2
WHERE st1.sno = st2.sno;
```

这时,虽然数据库的逻辑结构发生了改变,由一个表变成了两个表,但由于定义了一个和原表逻辑结构一样的视图,使用户的子模式没有发生改变。因此,用户的应用程序不必修改,仍可通过视图来查询数据,但更新操作会受到影响。

2. 删除视图

视图建好后,若导出此视图的基本表被删除了,该视图将失效,但视图定义一般不会被自动删除(除非指定了基本表的级联删除 CASCADE),故要用语句进行显式删除,该语句的一般格式为:

```
DROP VIEW <视图名>{ CASCADE | RESTRICT }
```

其中,RESTRICT 确保只有无相关对象(如其他视图、约束等)涉及的视图才能被撤销。

【例 3.22】 删除视图 s_good。

```
DROP VIEW s_good RESTRICT;
```

3. 查询视图

一旦视图定义好后,用户就可以像对基本表一样对视图进行查询了。也就是说,在前面介绍的对表的各种查询操作都可以作用于视图。

DBMS 执行对视图的查询时,首先进行有效性检查,检查查询涉及的表、视图等是否存在数据库中,如果存在,则从数据字典中取出查询涉及的视图的定义,把定义中的子查询和用户对视图的查询结合起来,转换成对基本表的查询,然后再执行这个经过修正的查询。这种将对视图的查询转换为对基本表的查询的过程称为视图消解(View Resolution)。

3.5.3 更新视图

目前,大部分数据库产品都能够正确地对视图进行数据查询的操作,但还不能对视图作任意的更新操作,因此视图的更新操作还不能实现逻辑上的数据独立性。下面介绍更新视图的一些限制。

更新视图包括插入(INSERT)、删除(DELETE)和修改(UPDATE)3 类操作。由于视图是不实际存储数据的虚表,因此对视图的更新最终要转换为对表的更新,即通过对表的更新来实现视图更新。

为防止用户通过视图对数据进行增、删、改时,无意或故意操作不属于视图范围内的表的数据,可在定义视图时加上 WITH CHECK OPTION 子句,这样在视图上增、删、改数据时,DBMS 会进一步检查视图定义中的条件,若不满足条件,则拒绝执行该操作。

【例 3.23】 删除例 3.18 ②信息院学生视图 d_student 中学号为 200029 的学生记录。

```
DELETE  
FROM d_student  
WHERE sno = '200029';
```

转换为对基本表的更新:

```
DELETE  
FROM student  
WHERE sno = '200029' AND dept = '信息院';
```

这里系统自动将 dept='信息院'放入子句中。

在关系数据库中,并非所有的视图都是可更新的,因为有些视图的更新不能唯一地有意义地转换成对相应基本表的更新。

如例 3.20 中定义的视图 s_avg 是由学号和平均成绩两个属性列组成的,其中平均成绩一项是由 s_c 表中对元组分组后计算平均值得来的。若想把视图中某学生的平均成绩改成 90 分,则对该视图的更新是无法转换成对基本表 s_c 的更新的。因为系统无法修改该生各科成绩。以使之平均成绩成为 90 分。所以该视图是不可更新的。

不能更新的视图有两种:

(1) 不可更新视图。理论上已证明是不可更新的视图。

(2) 不允许更新视图。理论上允许,但实际系统中不支持其更新。

有些视图从理论上讲是不可更新的,还有些视图在理论上是可更新的,但它们的确切特征还尚待研究。目前各关系 DBMS 一般都允许对行列子集视图进行更新,各系统对视图的更新均有自己的规定,由于系统实现方法上的差异,这些规定也不尽相同。

3.6 数据库访问技术

3.6.1 数据库访问概述

在设计数据库应用程序时,不可避免地会碰到如何访问数据库,如何使用不同数据库里的数据等问题。

应用程序与 SQL 数据库联系的基本方式如下:

(1) 嵌入式 SQL(Embedded SQL,ESQL)。该方式将 SQL 作为一种数据子语言嵌入到某些高级语言中,借助高级语言访问数据库以及实现过程化控制和复杂计算。

(2) 数据库接口。该方式通过应用程序编程接口(API)函数来访问数据库。利用它可屏蔽不同系统之间的差异,提供应用系统的可移植性和可扩展性。

(3) SQL 模块。该方式利用过程化 SQL(标准 SQL 的过程化扩展)编程,通过调用程序模块访问、操作数据库。

标准 SQL 语言作为一种独立的命令式语言,用户在交互环境下使用极为便利。然而在描述一些业务流程时,就有些无能为力了。过程化 SQL (Procedural Language/SQL,PL/SQL)正是为了解决这一问题而引入的。过程化 SQL 将第 4 代语言——SQL 语言的强大灵活性与第 3 代语言的过程性结构融为一体,集成了 SQL 数据操作能力和过程化的流程控制能力,提供了功能强大的数据库编程语言。因为不少数据库公司已将过程化 SQL 整合到数据库服务器及其开发工具中,所以在客户端环境和服务器中都可以编写过程化的 SQL 程序。过程化 SQL 语法和功能因 DBMS 不同而不同。有关内容将在 8.2 节介绍。本章仅介绍前两种方式。

3.6.2 嵌入式 SQL

1. 嵌入式 SQL 的引入

SQL 是一种强有力的非过程性查询语言,实现相同的查询比用通用编程语言要简单得多,那么为什么还要用高级语言访问数据库呢?这是因为以下几点:

(1) 有些复杂事务性处理单纯使用 SQL 难以完成。存在着不少应用是高度过程化的,还伴随着复杂的计算。虽然 SQL 在逐渐增强自己的表达能力,但一方面扩充的非过程化与计算能力有限,另一方面太多的扩展会导致优化能力及执行效率的降低。

(2) 还有许多应用不仅需要查出数据,还必须对数据进行随机处理。实际的应用系统是非常复杂的,数据库操作只是其中一部分。有些功能如与用户交互、图形化显示、复杂数据的处理等需借助于其他语言或软件实现。

为了解决这些问题,将 SQL 作为一种数据子语言嵌入到高级语言中,利用高级语言的过程性结构与强大的功能来弥补 SQL 语言实现复杂应用的不足。接受 SQL 嵌入的高级语言,称为主语言(或宿主语言)。标准 SQL 语言可以嵌入到 C/C++、Java、C#、FORTRAN、

Ada、PASCAL、COBOL 等主语言中使用。

对于主语言中的嵌入式 SQL 语句,通常采用两种方式处理:

- (1) 预编译方式。
- (2) 扩充主语言使之能够处理 SQL 的方式。

目前主要采用第一种方式。其主要过程为:先由 DBMS 的预处理程序对源程序进行扫描,识别出 SQL 语句,然后将它们转换为主语言的调用语句,使主语言能够识别它们,最后由主语言编译程序将整个源程序编译成可执行的目标代码。

嵌入式 SQL 语句根据其作用的不同,可分为可执行语句(数据定义、数据控制、数据操纵)和说明性语句。

说明:在主语言程序中,任何允许出现可执行高级语言语句的地方,都可以写可执行 SQL 语句。任何允许出现说明性高级语言语句的地方,都可以写说明性 SQL 语句。

2. 嵌入式 SQL 需解决的问题

使用嵌入式 SQL 需要面对下面 3 个主要问题:

- (1) 程序中既有主语言的语句又有 SQL 的语句,如何识别这两种不同的语句。
- (2) 程序中既有主语言变量又有 SQL 列变量,如何区分这两种不同的变量。
- (3) 主语言变量一般为标量,而 SQL 中的列变量一般均为集合量,如何实现由集合量到标量的转换。

讨论处理这些问题的技术手段之前先引入一些术语。

(1) 状态变量(SQLSTATE)。它是 5 个字符的变量,其作用是存储描述系统当前状态和运行情况的数据。状态代码由数字和 A~Z 之间的大写字母构成。前两个数字或字母表示:类(一般类别);后面的 3 个数字或字母表示:子类(特殊子类,如果有的话),状态变量的格式如图 3-4 所示。

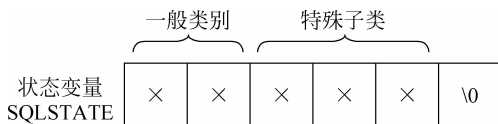


图 3-4 状态变量的格式

其中重要的类别特征是:00 类表明所有的事情都正常;01 类表明警示;02 类表明无数据;所有的其他类型表明 SQL 语句或执行有错误。

在每一个语句执行之后,DBMS 将描述当前工作状态和运行环境的一个状态值放入 SQLSTATE 中,应用程序根据 SQLSTATE 中的信息决定下一步要执行的语句。如果 SQLSTATE 含有任 00、01、02 之外的信息,就必须进行一些纠错操作或者甚至放弃工作。

(2) 主变量(Host Variable)。嵌入式 SQL 语句中可以使用主语言的程序变量来输入或输出数据。把这种 SQL 语句中使用的主语言程序变量简称为主变量。主变量根据其作用的不同,分为输入主变量和输出主变量。

① 输入主变量。由应用程序对其赋值,SQL 语句引用。利用输入主变量作为传递参数,可指定向数据库中插入的数据、指定修改数据的值,可以指定执行的操作、指定 WHERE 子句或 HAVING 子句中的条件等。

② 输出主变量。由 SQL 语句对其赋值或设置状态信息,返回给应用程序。利用输出主变量,可以得到 SQL 语句的结果数据和状态。

一个主变量可既是输入主变量又是输出主变量。

(3) 游标(Cursor)。SQL 语言与主语言具有不同的数据处理方式。SQL 语言是面向集合的,一条 SQL 语句可以产生或处理多条记录。而主语言是面向记录的,一组主变量一次只能存放一条记录。所以仅用主变量难以满足 SQL 语句向应用程序输出数据的要求,为此嵌入式 SQL 引入了游标的概念,用游标来协调这两种不同的处理方式。

游标是系统为用户开设的一个数据缓冲区,存放 SQL 语句的执行结果。每个游标区都有一个名字。游标的作用是用户可以通过游标指针及相关语句逐一获取记录,并赋给主变量,交给主语言处理,从而把对集合的操作转换为对单个记录的处理。游标工作示意图如图 3-5 所示。

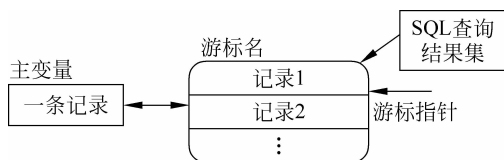


图 3-5 游标工作示意图

状态变量、主变量、游标是嵌入式 SQL 与主语言的接口,利用它们可实现两种不同风格语言的交互。

使用嵌入式 SQL 必须解决前面提到的 3 个主要问题,即

(1) 程序中如何区分主语言语句与 SQL 语句?

SQL 语句在嵌入主语言程序时要加标识。一般前缀标识为 EXEC SQL,结束标识为 END-EXEC(或用分号)。确切格式依赖于主语言。

例如,嵌入到 C 程序中用 EXEC SQL <嵌入的 SQL 语句>。

在 Java 中嵌入式 SQL 的格式为: #SQL{<嵌入的 SQL 语句>}。

(2) 程序中如何区分主语言变量与 SQL 的列变量?

SQL 语句中使用的主语言程序变量即主变量是主语言程序与 SQL 语句的共享变量。程序中区分主语言变量与 SQL 的列变量的方法是在主变量前加冒号,如“: name”。

【例 3.24】 根据主变量 sno 的值检索学生名、年龄和性别:

```

...                                     //主变量 sn、sa、ss 说明
EXEC SQL SELECT sname, age, sex INTO :sn, :sa, :ss //输出主变量放结果
FROM student
WHERE sno = :sno;                       //输入主变量设有初值

```

(3) 主语言变量与 SQL 的列变量如何交互与转换?

由于 SQL 的列变量是集合量,这就需要有一种机制:能将 SQL 变量中的集合量逐个取出送入主变量内,供主程序使用。方法是在嵌入式 SQL 中引入游标机制。

3. 嵌入式 SQL 的工作过程

将 SQL 嵌入到高级语言中混合编程,SQL 语句负责操纵数据库,高级语言语句负责控

制程序流程。这时程序中会含有两种不同计算模型的语句,一种是描述性的面向集合的 SQL 语句,另一种是过程性的记录式高级语言,它们之间应该如何通信呢?

数据库工作单元与源程序工作单元之间的通信主要包括以下几个步骤:

- (1) 主语言通过主变量向 SQL 语句提供查询等执行参数。
- (2) 系统通过状态变量 SQLSTATE 向主语言传递 SQL 语句的执行情况及结果状态,使主语言程序能据此控制程序流程。

- (3) 系统通过游标和主变量将 SQL 语句查询数据库的结果转换并交给主语言处理。

总之,SQL 语句与高级语言语句之间是通过主变量与游标等进行通信,嵌入式 SQL 的工作过程如图 3-6 所示。

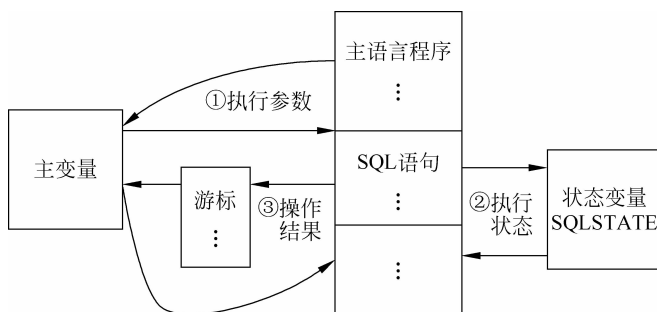


图 3-6 嵌入式 SQL 的工作过程

4. 嵌入式 SQL 程序

每一个内嵌 SQL 语句的 C 程序包括两大部分:应用程序首部和应用程序体。嵌入 C 程序中的 SQL 语句前须加前缀 EXEC SQL,以便与主语言的语句相区别。下面通过一个例子说明嵌入式 SQL 的程序组成。

【例 3.25】 在数据库 emp_db 中建立一个表: emp(eno,ename,sal,deptno),建表后可进行相关的操作。

```

#include <stdio.h>
EXEC SQL BEGIN DECLARE SECTION                                //主变量说明
VARCHAR SQLSTATE[6];
...
EXEC SQL END DECLARE SECTION;
//其他要用到的主变量

void main( )
{ EXEC SQL CONNECT TO emp_db USER <用户名>;
  if (<SQLSTATE 类码不为"00"和"01">)
    printf("Connection failed. SQLSTATE = %s.\n", SQLSTATE);
  EXEC SQL CREATE TABLE emp
    (eno INT,
     ename VARCHAR(8) NOT NULL,
     sal FLOAT,
     deptno INT,
     PRIMARY KEY(eno));
  if (<SQLSTATE 类码不为"00"和"01">)
    printf("Fail to create table. SQLSTATE = %s.\n", SQLSTATE);
}
  
```

```

else
    {printf("Table emp created");
    ...
    }
EXEC SQL COMMIT WORK RELEASE;
EXEC SQL DISCONNECT CURRENT;
exit(0);
}

```

程序的主要构成部分说明如下：

(1) DECLARE 说明段。用于定义主变量(又称共享变量),主变量既可在 C 语句中使用,也可在 SQL 语句中使用,应用程序可以通过主变量与数据库相互传递数据。该说明段用下列语句作为开始和结束：

```

EXEC SQL BEGIN DECLARE SECTION;
...
EXEC SQL END DECLARE SECTION;

```

所有主变量必须用 SQL 说明性语句进行说明。说明之后它们可以在 SQL 语句中任何一个能够使用表达式的地方出现,而在 SQL 语句之外,主变量可以直接引用,不必加冒号。

为了在主语言中检测可执行 SQL 语句的执行结果状态,SQL 规定应用程序必须使用一个特殊的主变量——共享状态变量 SQLSTATE,它应在 DECLARE 说明段中定义。

(2) CONNECT 语句。一般形式为：

```
EXEC SQL CONNECT TO <SQL 服务器或数据库名(含路径)> USER <用户名>
```

为执行 SQL 语句,操纵 SQL 建立的数据库。DBMS 要通过 CONNECT 语句建立 SQL 连接,这样就可以在用户环境下访问一个服务器上的 SQL 数据库。若不显式建立 SQL 连接,则 DBMS 执行默认的 SQL 连接。

(3) 应用程序体。其包含若干可执行 SQL 语句及主语言语句。当对数据库的操作完成后,应该提交和退出数据库,可使用简单的命令 COMMIT WORK RELEASE 实现。

一个含有嵌入式 SQL 语句的 C 语言程序的一般实施步骤如下：

- (1) 编写含有 SQL 语句的 C 程序。
- (2) 使用预编译器,产生预编译输出文件。
- (3) 使用 C 编译器对预编译输出文件进行编译,产生目标程序。
- (4) 连接目标程序,产生可执行程序。
- (5) 运行程序。

5. 动态 SQL 方法

在嵌入式 SQL 语句中,主变量的个数与数据类型在预编译时都是确定的,只有主变量的值在程序运行过程中可动态输入。这类嵌入式 SQL 语句称为静态 SQL 语句。

在许多情况下,静态 SQL 语句提供的编程灵活性显得不足。例如,需要查询选修了某些课程的选课学生的学号及其成绩;需要查询某类学生选修的课程情况等。这里的问题表现为查询条件或查询的属性列都是不确定的,仅用静态 SQL 语句难以实现。

如果在嵌入使用 SQL 的过程中,SQL 语句不预先确定,而根据需要在程序运行中动态

指定,这就是动态 SQL。

动态 SQL 的实质是允许在程序运行过程中临时“组装”语句。这些临时组装的 SQL 语句主要有 3 种基本类型。

(1) 具有条件可变的 SQL 语句。指 SQL 语句中的条件子句具有一定的可变性。例如,对于查询语句来说,SELECT 子句是确定的,即语句的输出是确定的,其他子句如 WHERE 子句和 HAVING 具有一定的可变性。

(2) 数据库对象、条件均可变的 SQL 语句。例如,对于查询语句,SELECT 子句中属性列名,FROM 子句中的基表名或视图名,WHERE 子句和 HAVING 短语中的条件均可由用户临时构造,即语句的输入和输出可能都是不确定的。

(3) 具有结构可变的 SQL 语句。在程序运行时临时输入完整的 SQL 语句。

在目前实现中,动态 SQL 分为直接执行、带动态参数和动态查询 3 种类型。

(1) 直接执行。直接执行的具体做法是由实际应用定义一个字符串主变量,用它存放要执行的 SQL 语句。在 SQL 语句中分固定部分和可变部分,其中固定部分由应用程序直接赋值,可变部分则由应用程序提示,在程序执行时用户输入。此后即用 EXEC SQL EXECUTE IMMEDIATE 语句执行字符串主变量中的 SQL 语句。

(2) 带动态参数。在此类 SQL 语句中含有未定义的变量,这些变量仅起占位符的作用。在此类语句执行前,应用程序提示输入相应参数以取代这些变量。

(3) 动态查询。动态查询 SQL 语句是一类针对查询结果为集合且需返回查询结果,但往往不能在编程时予以确定的 SQL 语句。此类查询方法是预先由应用定义一个字符串主变量,用它存放动态的 SQL 语句。应用程序在执行时提示输入相应的 SQL 语句。此类 SQL 查询一般需要用游标。

动态 SQL 语句不是直接嵌入在主语言中的,而是在程序中设置一个字符串变量,程序运行时交互式地输入或从某个文件读取动态 SQL 语句,接收并存入到该字符串变量中去。

动态 SQL 执行过程如下:

(1) 执行准备语句。

格式: PREPARE <语句名> FROM :<主变量>

功能: 接收含有 SQL 语句的主变量,将其送到 DBMS 编译并生成执行计划。

(2) 执行该计划。动态 SQL 的语句有 EXECUTE(执行已准备好的 SQL 语句)、EXECUTE IMMEDIATE(立即执行除 SELECT 以外的 SQL 语句)等。

【例 3.26】 在学生选课表 s_c 中修改某课程的成绩,使其提高 10%。

```
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR prep[100]; VARCHAR cno[3];          //主变量说明
EXEC SQL END DECLARE SECTION;
strcpy (prep, "UPDATE s_c SET grade = grade * 1.1 WHERE cno = ? ");
                                                    //?为参数标志
EXEC SQL PREPARE prep_stat FROM :prep;
scanf ( "%s", cno);
EXEC SQL EXECUTE prep_stat USING :cno;          //:cno 的值代替参数?"
```

嵌入式/动态 SQL 方法有较高的运行效率,适用于嵌入式数据库等应用,但不够灵活、编程复杂。在网络广泛使用的客户端和服务端环境下,常使用数据库接口方式。

3.6.3 数据库标准接口

1. 数据库接口简介

数据库接口是指为支持数据库应用开发而提供的各种应用程序编程接口。数据库接口标准涉及的技术内容非常广泛,几种典型的数据库接口如下:

(1) ODBC(OpenData Base Connectivity,开放式数据库互联)提供了一组对数据库访问的标准接口,使用户通过 SQL 可访问不同的关系数据库,适合数据库底层开发。

(2) JDBC(JavaData Base Connectivity,Java 数据库连接)是基于 Java 语言的数据访问标准接口,具有平台无关性。它由一组类和接口组成,可统一访问多种关系数据库。JDBC 应用程序使用简便、灵活,更适用于 Internet 上异构环境的数据库应用。

(3) ADO(ActiveX Data Objects).NET 是在微软 Web 服务平台使用的数据库对象访问接口。它是一组用于和数据源进行交互的面向对象类库,提供了平台互用性和可伸缩的数据访问,功能强、易用、高效。

(4) SQL/CLI(Call Level Interface,SQL 调用级接口)是 SQL 的应用编程接口,它等于核心 ODBC API。它定义了一套公共函数,从一个主语言应用程序可以调用这些函数,连接到数据库,并利用 SQL 语句存取和更新数据。

(5) OLE DB(Object Linking and Embedding DataBase,对象连接与嵌入数据库)建立于 ODBC 之上,并将此技术扩展为提供更高级数据访问接口的组件结构。它有较好的健壮性、灵活性和很强的错误处理能力,能够同非关系数据源进行通信。

(6) OTL 是 Oracle、ODBC 和 DB2-CLI Template Library 的缩写,是一个 C++ 编译中操控关系数据库的模板库。具有跨平台、运行效率高、使用简洁、部署容易等优点。

本节介绍两种典型的标准数据库接口,即 ODBC、JDBC。

2. 开放式接口——ODBC

ODBC 是微软推出的一种工业标准,一种开放的独立于厂商的应用程序接口,可以跨平台访问各种个人计算机、小型机及主机系统,方便异构数据库之间进行数据共享。ODBC 作为一个工业标准,绝大多数数据库厂商、大多数应用软件和工具软件厂商都为自己的产品提供了 ODBC 接口或提供了 ODBC 支持。

ODBC 是依靠分层结构来实现的,这样可以保证其标准性和开放性。ODBC 的体系结构可以分为 4 层:数据库应用程序、驱动程序管理器、数据库驱动程序和数据源,ODBC 的体系结构如图 3-7 所示。解释如下:

(1) 数据库应用程序主要处理业务逻辑,是 ODBC 的使用者。它调用 ODBC 函数,递交 SQL 语句给 DBMS,从而实现与数据库的交互,并对数据库的数据进行处理。它可以用 Object Pascal、C++、VB、Java 等高级语言编写。

(2) 驱动程序管理器是一个名为 ODBC.DLL 的动态链接库。它为不同的数据库驱动程序(如 Oracle、DB2、SQL Server 等驱动程序)提供统一的函数调用接口,并管理这些不同的驱动程序,实现各种驱动程序的正确调度。

(3) 数据库驱动程序也是一个动态链接库,由各个数据库厂商提供。它提供对具体数据库操作接口,因此它是应用程序对各种数据源发出操作的实际执行者。

(4) 数据源(Data Source Name,DSN)是驱动程序与数据库连接的桥梁,用于表示一个

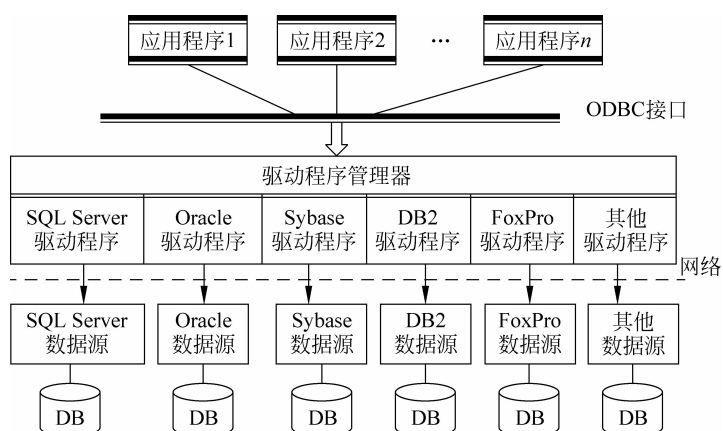


图 3-7 ODBC 的体系结构

特殊连接的命名,它隐藏了诸如数据库文件名、所在目录、数据库驱动程序、用户标识、密码等细节。当建立一个连接时,用户不用考虑数据库文件名、地址等,只要给出其数据源名即可。

有了 ODBC,当应用程序要访问数据库时,就可以使用 ODBC 的驱动程序管理器的操作接口,而不用直接存取数据库。驱动程序管理器将操作请求提交给具体的 DBMS 驱动程序,由具体的驱动程序实现对数据源的各种操作,数据库的操作结果也通过驱动程序管理器返回给应用程序。

ODBC 的基本特性是独立于特定数据库。使用相同源代码的应用程序通过动态加载不同的驱动程序,可以访问不同的数据库。支持在应用程序中同时建立多个数据库连接,以同时访问多个异构的数据库。可在 Windows 的“ODBC 数据源管理器”中建立与各种数据库连接的数据源(具体操作参见 ODBC 的联机帮助),以后各种编程语言就可以通过这个数据源访问数据库。ODBC 的最大优点是能以统一的方式处理所有的关系数据库。

3. 调用级接口——JDBC

JDBC 是由 Sun 公司开发的,属 Java 语言的一部分。JDBC 是一个为数据库管理提供的 API,它提供了调用级接口,以便从一个 Java 语言程序中执行 SQL 语句。它同动态 SQL 一样,可以在运行时构造 SQL 语句,能够通过 Internet 访问数据库,且比动态 SQL 移植性更好。

(1) JDBC 的特性。JDBC 保持了 ODBC 的基本特性,此外,更具有对硬件平台、操作系统异构性的支持。这主要是因为 JDBC 使用的是 Java 语言。利用 Java 的平台无关性,JDBC 应用程序可以自然地实现跨平台特性,因而更适合于 Internet 上异构环境的数据库应用。还有,JDBC 驱动程序管理器是内置的,驱动程序本身也可通过 Web 浏览器自动下载,无须安装、配置;而 ODBC 驱动程序管理器和 ODBC 驱动程序必须在每台客户机上分别安装、配置。

(2) JDBC 的主要工作。如图 3-8 所示,应用通过 JDBC 连接并访问数据库。应用程序通过一个被称为驱动管理器的 JDBC 模块与 DBMS 通信。JDBC 为最常用的 DBMS 保留一个单独的驱动。当将要执行一个 SQL 语句时,应用程序将语句的一个字符串表达式发送

给合适的驱动,由它执行任何必要的格式化,然后将语句发送给 DBMS,进行准备和执行。

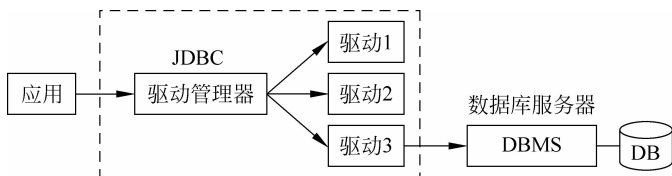


图 3-8 应用通过 JDBC 连接并访问数据库

JDBC 完成的主要工作是:建立与数据库的连接;发送 SQL 语句;返回数据结果给用户。JDBC 提供了一些其他的特性,如它可以通过在数据库关系上的查询建立一个可更新的结果集,对结果集中元组的一次更新会导致数据库关系中相应元组的更新。

(3) JDBC 的编程要点。使用数据库进行 JDBC 编程时,Java 程序中通常应包含下述几部分内容(具体编程参见 Java 程序设计有关书籍):

- ① 在程序的首部用 import 语句将 java. sql 包引入程序。
- ② 使用 Class. forName()方法加载相应数据库的 JDBC 驱动程序。
- ③ 定义 JDBC 连接目标 DBMS 的连接对象,并连接到数据库。
- ④ 创建操作语句对象。
- ⑤ 使用 SQL 语句对数据库进行操作。
- ⑥ 使用 close()方法解除 Java 与数据库的连接,并关闭数据库。

3.6.4 对象访问接口

1. ADO.NET 简介

ADO. NET 是在 Windows 中 Web 服务平台使用的数据库访问接口。它是一组用于和数据源进行交互的面向对象类库,ADO. NET 类库中的类提供了众多对象,分别完成与数据库的连接、查询、插入、删除和更新记录等操作。提供了 Web 应用平台的互用性和可伸缩的数据访问,功能强、易用、高效。

ADO. NET 允许和不同类型的数据源及数据库进行交互,增强了对非连接编程模式的支持,任何能够读取 XML 格式的应用程序都可以进行数据处理。ADO. NET 提供与数据源进行交互的公共方法,但是对于不同的数据源采用不同的类库,并且通常是以与之交互的协议和数据源的类型来命名的,称为数据提供者(Data Providers),它负责把 .NET 应用程序连接到数据源,有关于 Oracle、SQL Server 等不同的 .NET 数据提供者,每种数据提供者各自命名空间中。数据提供者虽然针对的数据源不一样,但是其对象的架构都一样,只要针对数据源种类来选择即可,如使用 Oracle 数据提供者的组件对象只需加前缀 Oracle。

2. ADO.NET 对象模型

ADO. NET 对象模型如图 3-9 所示。这些组件中负责建立联机和数据操作的部分称之为数据提供者,由 Connection 对象、Command 对象、DataAdapter 对象及 DataReader 对象组成。数据提供者最主要是充当 DataSet 对象及数据源之间的桥梁,负责将数据源中的数据取出后植入 DataSet 对象中,以及将数据存回数据源的工作。

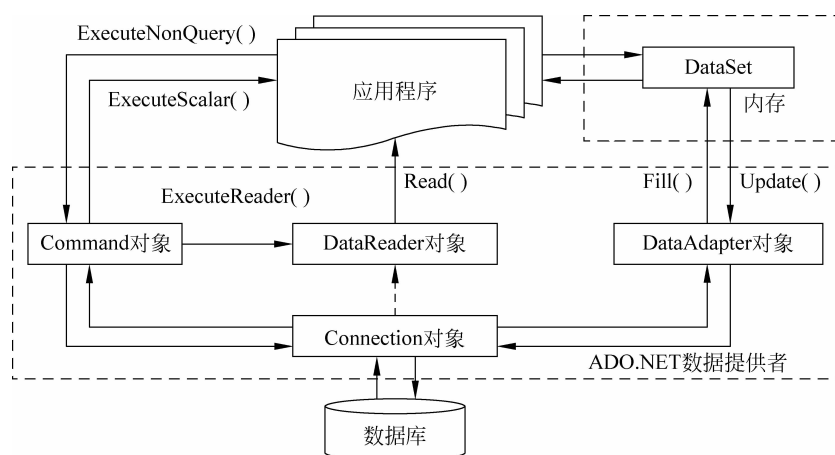


图 3-9 ADO.NET 对象模型

(1) ADO.NET 的数据提供者。它包含 4 个主要组件：

① Connection 对象。其主要用于开启程序和数据库之间的连接。不利用连接对象将数据库打开,是无法从数据库中取得数据的。这个对象在 ADO.NET 的最底层,用户可以自己创建这个对象,或由其他的对象自动创建。

② Command 对象。其主要可以用来对数据库发出一些指令,如可以对数据库下达查询、新增、修改、删除数据等指令,以及呼叫存在于数据库中的预存程序等。Command 对象是透过连接到数据源的 Connection 对象来下命令的,所以连接到哪个数据库,Command 对象的命令就下到哪里。

③ DataAdapter 对象。其主要是在数据源及 DataSet 之间进行数据传输的工作,它可以透过 Command 对象下达命令,并将取得的数据放入 DataSet 对象中。这个对象是架构在 Command 对象上的,并提供了许多配合 DataSet 使用的功能。

④ DataReader 对象。当只需要顺序地读取数据而不需要进行其他操作时,可以使用 DataReader 对象。因为 DataReader 对象在读取数据时限制每次只读取一笔,而且只能读取,所以使用起来不但节省资源而且效率很高。此外,因为不用把数据全部传回,故可以降低网络的负载。

(2) DataSet 对象。它可以把从数据库中查询到的数据保留起来,甚至可以将整个数据库显示出来。DataSet 的能力不只是可以储存多个表,还可以透过 DataAdapter 对象取得一些如主键等的数据表结构,并可以记录数据表间的关联。可将 DataAdapter 对象当作 DataSet 对象与数据源之间传输数据的桥梁。正是由于 DataSet 的存在,才使得程序员在编程时可以屏蔽数据库之间的差异,从而获得一致的编程模型。

3. ADO.NET 访问基本步骤

ADO.NET 提供了两种读取数据库的方式。

方式 1: 通过 DataReader 对象读取数据。这种方式只能读取数据库,如果只是想查询记录,这种方式的效率更高些。

方式 2: 通过数据集 DataSet 和 DataAdapter 对象访问和操作数据。这种方式更灵活,

可以对数据库进行各种操作。

ADO.NET 的最重要的概念之一是 DataSet 对象,它是不依赖于数据库的独立数据集合,即使断开数据链路或者关闭数据库,DataSet 依然是可用的。

以使用对象访问和操作数据方式 2 为例,ADO.NET 访问数据库的步骤如下:

- ① 使用 Connection 对象创建一个数据库链路。
- ② 使用 Command 和 DataAdapter 对象请求一个记录集合。
- ③ 把记录集合暂存到 DataSet 中。
- ④ 如果需要,返回第②步(DataSet 可以容纳多个数据集合)。
- ⑤ 关闭数据库链路。
- ⑥ 在 DataSet 上进行所需要的操作。

4. 通过 ADO.NET 访问数据库

首先需要准备数据。设创建数据库 mydb; 在创建的数据库中,创建 usr 表(no,name,pass); 再插入若干数据。下面以 Oracle 数据库为例来讲解。假定数据库的用户名为 sa,密码为 123456。

(1) 通过 DataReader 对象读取数据。它只能实现对数据的读取,查询结果形成一个只读只进的数据流存储在客户端。DataReader 对象的 Read 方法可以前进到下一条记录。在默认情况下,每执行一次 Read 方法只会在内存中存储一条记录,系统的开销非常少。

要使用 ADO.NET,必须先加载 System.Data 这个命名空间,因为该名称空间中包括大部分组成 ADO.NET 架构的基础对象类别,如 DataSet 对象、数据表、字段、关联等。

【例 3.27】 下面的程序完成的功能是访问数据。它从数据库中读取记录,并将查询结果通过控制台输出。程序如下:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;                                     //载入 ADO.NET 的命名空间
using System.Data.OracleClient;                       //使用 ORACLE 的数据提供者
namespace adoconsole.ado
{
    class test1
    {
        static void main(string[] args)
        {
            Console.WriteLine("ADO 测试");
            string myconn = "Data Source = localhost; Integrated Security = SSPI; Initial Catalog = mydb; User ID = sa; Password = 123456";
            string mysql = "SELECT no , name,pass FROM usr"; //需执行的 SQL 命令
            OracleConnection myconnect = new OracleConnection(myconn); //创建数据连接
            myconnect.Open(); //打开数据库连接
            OracleCommand mycomm = new OracleCommand(mysql, myconnect); //创建 SQL 命令对象并初始化
            OracleDataReader myread = mycomm.ExecuteReader();
            //构造阅读器,同时执行 SQL 命令
            while(myread.Read()) //读出每行数据并显示
            {
                Console.WriteLine(myread.GetInt32(0) + ", " + myread.GetString(1) + ", " + myread.GetString(2));
            }
            myread.Close(); //关闭阅读器
        }
    }
}
```

```

        myconnect.Close();           // 关闭连接
    }
}

```

这种方法仅适用于读取少量数据,特点是速度快。要读取大量数据并操作时,需用下面的方法。

(2) 通过数据集 DataSet 和 DataAdapter 对象访问和操作数据。

【例 3.28】 下面的程序完成的功能是访问数据并操作。示例程序如下:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;                                //载入 ADO.NET 的命名空间
using System.Data.OracleClient;                  //使用 SQL 数据提供者
namespace adoconsole.ado
{
    class test2
    {
        static void main(string[] args)
        {
            Console.WriteLine("ADO 测试类");
            OracleConnection sqlconn = new OracleConnection("Data Source = localhost;Integrated
            Security=SSPI;Initial Catalog= mydb; User ID= sa;Password = 123456"); //创建数据连接
            OracleCommand selectcomm = new OracleCommand("SELECT no, name,pass FROM usr", sqlconn
            ); //创建并初始化 OracleCommand 对象
            OracleDataAdapter sqldata = new OracleDataAdapter();
            sqldata.SelectCommand = selectcomm; //创建对象并查询数据
            sqlconn.Open(); //打开数据库连接
            DataSet data = new DataSet(); //用 OracleDataAdapter 的 Fill 方法填充 DataSet
            sqldata.Fill(data, "usr");
            Console.WriteLine(data.Tables[0].Rows[0]["no"] + " " + data.Tables[0].Rows[0]["
            name"] + " " + data.Tables[0].Rows[0]["pass"]); //取一行数并显示
            sqlconn.Close(); //关闭数据库连接
            OracleCommandBuilder sqlbuilder = new OracleCommandBuilder(sqldata);
            //以 sqldata 为参数来初始化对象 sqlbuilder
            data.Tables["usr"].Rows[0].Delete(); //删除 DataSet 中的第一行数据
            sqldata.Update(data, "usr");
            data.Tables["usr"].AcceptChanges();
            //调用 Update 方法,以 DataSet 中的数据更新表
            Console.WriteLine(data.Tables[0].Rows[0]["no"] + " " + data.Tables[0].Rows[0]["name"] + "
            " + data.Tables[0].Rows[0]["pass"]); //取一行数据并显示
        }
    }
}

```

总之,ADO.NET 是与数据源交互的数据库对象访问接口。其数据提供者允许应用程序与不同的数据源交流。其中数据提供者组件的主要功能如下:

- (1) OracleConnection 对象管理与数据源的连接。
- (2) OracleCommand 对象允许与数据源交流并发送命令。
- (3) 使用 OracleDataReader 进行快速的向前读取数据。

(4) 用 DataSet 对象以及用 OracleDataAdapter 实现读取或者操作数据。

3.7 本章小结

SQL 是一种结构化、非过程化且进行了过程化扩展的查询语言。数据库是表、视图等的汇集,它由一个或多个数据库模式来定义。用 SQL 可以定义和删除所需的模式、关系表、视图和索引;对数据库中数据进行各种查询和更新操作。其中,查询包括单表查询、多表连接、嵌套和递归等查询,并能对查询结果进行统计、计算和排序等。SQL 提供视图功能,视图是由若干个基本表或其他视图导出的表。视图可简化数据查询,保持数据独立性,提供数据安全性。

SQL 用户可以是终端用户,也可以是应用程序。嵌入式 SQL 是将 SQL 作为一种数据子语言嵌入到高级语言中,利用高级语言和其他专门软件来弥补 SQL 实现复杂应用的不足。动态 SQL 允许在执行一个应用程序中,根据不同的情况动态地定义和执行某些 SQL 语句,可实现应用中的灵活性。

ODBC 是一种开放的应用程序接口,可以跨平台访问各种数据库。JDBC 提供了一个公共的、网络透明的、与具体数据库无关的应用程序接口。ADO.NET 是与数据源交互的对象访问接口,Web 应用功能强、易用、高效。它们都能够通过因特网访问数据库,且比嵌入式和动态 SQL 的实现移植性更好。

习题 3

1. 名词解释与理解:SQL 环境、数据库模式、基本表、视图、相关子查询、递归合并查询、外部连接、过程化 SQL。

2. 简述 SQL 语言的功能、特点。

3. 简述 SQL 数据库层次结构及 SQL 语句的分类。

4. SQL 查询条件中 SOME、IN 和 EXISTS 有什么异同? SQL 查询中何处使用空值?

5. 简述视图与表有何不同? 视图有哪些用途?

6. 设学生——选课数据库同表 2-5 至表 2-7,试用 SQL 语句完成以下操作:

(1) 查询考试成绩不及格的学生的学号及成绩。

(2) 查询年龄在 19~25 岁(包括 19 和 25 岁)之间的学生的姓名、院系和年龄,并按年龄的降序排序。

(3) 查询姓名中含“国”字的学生档案情况。

(4) 按院系查询学生总人数。

(5) 计算选修了 008 号课程的学生的平均成绩和最高分及最低分。

(6) 求平均成绩在 85 分以上(含 85)的各门课程号及平均成绩。

7. 已知学生表 S、任课表 C 和选课表 SC,参见第 2 章习题 5,试用 SQL 语句实现下列查询:

(1) 查询“张林”老师所授课程号和课程名。

(2) 查询选修课程名为“C 语言”或者“数据库”的学生学号。

- (3) 查询“高灵”同学所选修课程的课程号及课程名。
- (4) 查询至少选修课程号为 c_3 和 c_4 的学生姓名。
- (5) 用 EXISTS 来查询学习课程号为 c_5 的学生学号与姓名。
- (6) 查询不学 c_2 课程的学生姓名与性别。

8. 设有以下关系:

职工: E (职工号, 姓名, 性别, 职务, 家庭地址, 部门号)

部门: D (部门号, 部门名称, 地址, 电话)

保健: B (保健号, 职工号, 检查日期, 健康状况)

用关系代数((1)~(4)题)、SQL 语言((1)~(6)题)完成下列功能:

- (1) 查找所有女科长的姓名和家庭地址。
 - (2) 查找部门名称为“办公室”的科长姓名和家庭地址。
 - (3) 查找部门名为“财务科”、健康状况为“良好”的职工姓名和家庭地址。
 - (4) 删除职工关系表中职工号为“3016”的记录。
 - (5) 将职工号为“3016”的职工的健康状况改为“一般”。
 - (6) 建立健康状况为“差”的职工情况的视图。
9. SQL 有哪几种连接? 它们之间有何不同?
10. 视图消解是指什么? 为什么对视图的更新有限制?
11. 为什么引入嵌入式 SQL? 其工作过程是什么?
12. 动态 SQL 的意义是什么? 如何实现动态 SQL?
13. ODBC 驱动程序的体系结构由哪些部分组成? 其中数据源名的类型与含义是什么?
14. 讨论嵌入式 SQL 与动态 SQL 的主要区别。
15. 讨论 ODBC 与 JDBC 之间的一些主要异同。