第3章 CHAPTER 3

结构化查询语言

扫一扫

视频讲解

3.1 SQL 概述

结构化查询语言(SQL)是关系数据库的标准语言,是一种数据库查询和程序设计语言,用于存取数据以及查询、更新和管理关系数据库系统。

3.1.1 SQL 的发展

SQL 是 1974 年由 Boyce 和 Chamberlin 提出的,并在 IBM 公司的关系数据库系统 System R 上实现。由于它功能丰富、简单易学、使用方便,所以深受用户和计算机工业界的欢迎,被众多数据库厂商所采用。

1986 年 10 月,美国国家标准局(American National Standard Institute,ANSI)的数据库委员会 X3H2 批准了 SQL 作为关系数据库语言的美国标准,同年公布了 SQL 标准(简称 SQL-86)。1987 年,国际标准化组织(International Organization for Standardization,ISO)也通过了这一标准。随着数据库技术的发展,ANSI 也不断修改和完善 SQL 标准,并于 1989 年公布了 SQL-89 标准,于 1992 年公布了 SQL-92 标准,于 1999 年公布了 SQL-99(SQL 3)标准,之后又公布了 SQL:2003、SQL:2008、SQL:2011、SQL:2016 以及 SQL:2019 标准,而且 SQL 标准的内容也越来越多。

自 SQL 成为国际标准语言以来,各个数据库厂家纷纷推出各自的 SQL 软件或与 SQL 的接口软件。这就使大多数数据库均使用 SQL 作为共同的数据存放语言和标准接口,使不同数据库系统之间的互相操作有了共同的基础。此外,SQL 成为国际标准,对数据库以外的领域也产生了很大影响,有不少软件产品将 SQL 的数据查询功能与图形功能、软件工程工具、软件开发工具、人工智能程序结合起来。SQL 已成为数据库领域中的一门主流语言,被广泛应用在商用系统中,现已成为数据开发的标准语言。

3.1.2 SQL 的特点

SQL 是一种通用的、功能强大同时又简单易学的关系数据库语言,集数据查询(Data Query)、数据操纵(Data Manipulation)、数据定义(Data Definition)和数据控制(Data Control)四大功能于一体,主要特点如下。

1. 综合统一

数据库系统的主要功能是通过数据库支持的数据语言来实现的。

非关系模型的数据语言一般都分为模式数据定义语言(Schema Data Definition Language,模式 DDL)、外模式数据定义语言(Subschema Data Definition Language,外模式 DDL 或子模式 DDL)、与数据存储有关的描述语言(Data Storage Description Language,DSDL)及数据操纵语言(DML),分别用于定义模式、外模式、内模式和进行数据的存取与处置。其缺点是当用户数据库投入运行后,如果需要修改模式,必须停止现有数据库的运行,转储数据,修改模式,编译后再重新装载数据库,非常麻烦。

SQL 集数据定义语言(DDL)、数据操纵语言(DML)、数据控制语言(DCL)的功能于一体,语言风格统一,可以独立完成数据库生命周期中的全部活动,包括定义关系模式、录入数据以及建立数据库、查询、更新、维护、数据库重构、数据库安全性控制等一系列操作要求,这就为数据库应用系统开发提供了良好的环境。例如,用户在数据库投入运行后还可根据需要修改模式,并且不影响数据库的运行,从而使系统具有良好的可扩充性。

2. 高度非过程化

非关系数据模型的数据操纵语言是面向过程的语言,若要完成某项请求,必须指定正确的存储路径,而 SQL 是高度非过程化语言,当进行数据操作时,只要指出“做什么”,无须指出“怎么做”,存储路径对用户来说是透明的。因此,用户无须了解存储路径,存储路径的选择以及 SQL 语句的操作过程由系统自动完成,减轻了用户的负担,有利于提高数据独立性。

3. 面向集合的操作方式

非关系数据模型采用的是面向记录的操作方式,操作对象是一条记录。例如,查询所有库存量小于 10 的配电物资,用户必须一条一条地把满足条件的记录找出来(通常要说明具体处理过程,即按照哪条路径、如何循环等),而 SQL 采用集合操作方式,不仅操作对象、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。

4. 用同一种语法结构提供两种使用方式

SQL 有两种使用方式,一种是作为独立的自含式语言,它能够独立地用于联机交互,用户可以在终端键盘上直接输入 SQL 命令对数据库进行操作;另一种是作为嵌入式语言,SQL 语句能够嵌入高级语言(如 C、C++、Java)程序中供程序员设计程序时使用,而在两种不同的使用方式下,SQL 的语法结构基本上是一致的。这种以统一的语法结构提供两种不同的使用方式的做法为用户提供了极大的灵活性与方便性。

5. 语言简洁,易学易用

SQL 功能极强,语言十分简洁,其核心功能只用了 9 个动词(见表 3.1),接近英语口语,因此易学易用。

表 3.1　SQL 功能的实现动词

SQL 功能	实 现 动 词	SQL 功能	实 现 动 词
数据查询	SELECT	数据操纵	INSERT、UPDATE、DELETE
数据定义	CREATE、DROP、ALTER	数据控制	GRANT、REVOKE

3.1.3　SQL 的基本概念

SQL 支持关系数据库三级模式结构,如图 3.1 所示。其中,外模式对应于视图(View)

和部分基本表(Base Table)；模式对应于基本表；内模式对应于存储文件(Stored File)。

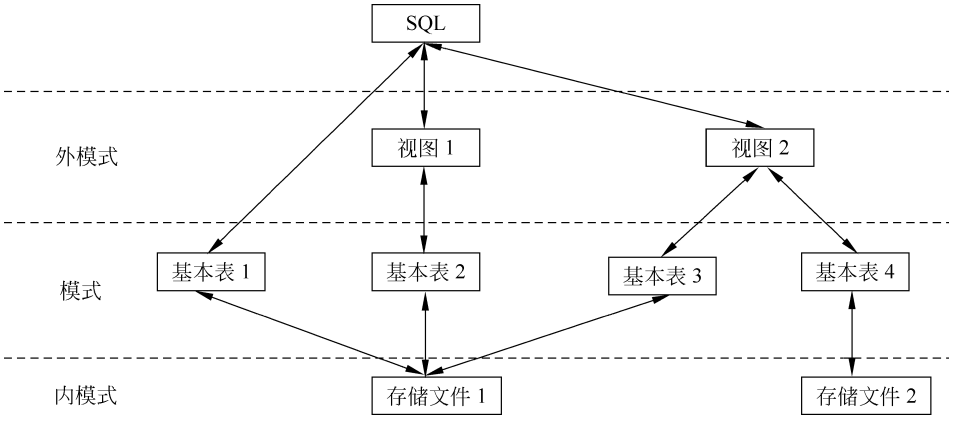


图 3.1　SQL 对关系数据库模式的支持

(1) 可以用 SQL 语句对视图和基本表进行查询等操作。在用户看来，视图和基本表是一样的，都是关系(即表)。

(2) 基本表是本身独立存在的表，是实际存储在数据库中的表。在 SQL 中，一个关系对应一张表。

(3) 视图是从基本表或其他视图中导出来的表，它本身不独立存储在数据库中。也就是说，数据库中只存放视图的定义，不存放视图的数据，这些数据仍存放在导出视图的基本表中，因此视图是一个虚表。

(4) 存储文件的逻辑结构组成了关系数据库的内模式，存储文件的物理结构是任意的，对用户是透明的。

扫一扫

视频讲解

3.2 数据定义语句

关系数据库系统支持三级模式结构，其模式、外模式和内模式中的基本对象有表、视图和索引。因此，SQL 的数据定义功能包括定义表、定义视图和定义索引，如表 3.2 所示。

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	ALTER VIEW
索引	CREATE INDEX	DROP INDEX	ALTER INDEX

本节只介绍如何定义基本表，视图的概念及其定义方法将在 3.5 节中详细阐述，索引的概念及其定义方法将在后续章节详细阐述。

3.2.1 基本表的定义

建立数据库最基本、最重要的一步就是定义一些基本表。SQL 使用 CREATE TABLE 语句定义基本表，一般语法格式如下。

```
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>]);
```

其中,<表名>是所要定义的基本表的名称,它可以由一个或多个属性(列)组成;括号中是该表的各个属性列,此时需要说明各属性列的数据类型;在创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,当用户操作表中的数据时,由 DBMS 自动检查该操作是否违背了这些完整性约束条件。如果完整性约束条件涉及该表的多个属性列,则必须定义在表级,否则既可以定义在列级,也可以定义在表级。

本章的所有示例均根据电力抢修工程数据库进行讲解,该数据库的结构见第 2 章的例 2.1,假设已经创建了 sampledb 数据库。

【例 3.1】　建立一个抢修工程计划表(Salvaging),它由工程项目编号(prj_num)、工程项目名称(prj_name)、工程开始日期(start_date)、工程结束日期(end_date)、是否按期完成(prj_status)共 5 个属性组成。

```
use sampledb;
CREATE TABLE Salvaging
(     prj_num char(8) PRIMARY KEY,                -- 列级完整性约束,prj_num 是主键
      prj_name varchar(50),
      start_date datetime,
      end_date datetime,
      prj_status bit
);
```

系统执行上述 CREATE TABLE 语句后,在数据库中建立一个新的空表 Salvaging,并将有关 Salvaging 表的定义及约束条件存放在数据字典中。

【例 3.2】　建立一个配电物资库存记录表(Stock)。

```
use sampledb;
CREATE TABLE Stock
(    mat_num char(4)   PRIMARY KEY,          -- 列级完整性约束,mat_num 是主键
     mat_name varchar(50)   NOT NULL,        -- mat_name 不允许取空值
     speci varchar(20)   NOT NULL,           -- speci 不允许取空值
     warehouse char(20),
     amount int,
     unit decimal(18,2),
     CHECK(mat_num like'[m][0-9][0-9][0-9]')
                              -- mat_num 属性列的 CHECK 约束,要求第 1 位为字符 m,后 3 位为数字
);
```

【例 3.3】　建立一个配电物资领料出库表(Out_stock)。

```
use sampledb;
CREATE TABLE Out_stock
(    prj_num char(8),
     mat_num char(4),
     amount int,
     get_date datetime   default now(),    -- now()为系统时间,SQL Server 中该函数为 getdate()
     department char(20),
     PRIMARY KEY(prj_num,mat_num),         -- 主键由两个属性构成,必须作为表级完整性约束
```

```
    FOREIGN KEY(prj_num) REFERENCES salvaging(prj_num),
                    -- 表级完整性约束条件,prj_num 是外键,被参照表是 Salvaging
    FOREIGN KEY(mat_num) REFERENCES stock(mat_num)
                    -- 表级完整性约束条件,mat_num 是外键,被参照表是 Stock
);
```

在定义表的各个属性时,需要指明其数据类型及长度。需要注意,不同的 RDBMS 支持的数据类型不完全相同。本书附录 A 中详细列举了 MySQL 8.0 提供的一些系统数据类型。

3.2.2 基本表的修改

在基本表建立之后,用户可以根据实际需要对基本表的结构进行修改。SQL 用 ALTER TABLE 语句修改基本表,一般语法格式为

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

其中,<表名>是要修改的基本表名称；ADD 子句用于增加新列和新的完整性约束条件；DROP 子句用于删除指定列和指定的完整性约束条件；MODIFY 子句用于修改原先的列名和数据类型。

说明：SQL Server 中,修改列名的关键字是 ALTER。

【例 3.4】 向抢修工程计划表(Salvaging)中增加“工程项目负责人”列(prj_director),数据类型为字符型。

```
ALTER TABLE Salvaging ADD prj_director varchar(10);
```

注意：无论基本表中原来是否已有数据,新增加的列一律为空值。

【例 3.5】 删除抢修工程计划表(Salvaging)中“工程项目负责人”属性列(prj_director)。

```
ALTER TABLE Salvaging DROP COLUMN prj_director;
```

【例 3.6】 将配电物资领料出库表(Out_stock)中领取部门(department)的数据类型改为可变长度字符类型。

```
ALTER TABLE Out_stock
    MODIFY COLUMN department varchar(20) NOT NULL;
```

说明：SQL Server 中,该语句为

```
ALTER TABLE Out_stock
    ALTER COLUMN department varchar(20) NOT NULL;
```

注意：修改原有的列定义有可能会破坏已有数据。

3.2.3 基本表的删除

当不再需要某个基本表时,可以使用 DROP TABLE 语句将其删除,一般语法格式为

```
DROP TABLE <表名>
```

【例 3.7】 删除配电物资领料出库表(Out_stock)。

```
DROP TABLE Out_stock;
```

注意：基本表的删除是有限制条件的，要删除的基本表不能被其他表的约束(CHECK、FOREIGN KEY 等约束)所引用。如果存在这些依赖该表的对象，则此表不能被删除。

例如，执行 DROP TABLE Stock 语句，系统会给出如图 3.2 所示的提示信息。

图 3.2　删除 Stock 表的提示信息

一旦删除基本表，不仅表中的数据和此表的定义将被删除，而且在该表上建立的索引、视图、触发器等有关对象也将被删除，因此用户执行删除基本表的操作时一定要格外小心。

3.2.4　约束的添加和删除

基本表的约束有主键约束、外键约束、非空约束、唯一性约束、用户自定义的约束等，可以在创建表时创建约束(列级约束还可以在修改表的结构时创建)，也可以在表已经创建完成后再添加或删除约束。一般语法格式为

```
ALTER TABLE <表名> [ADD CONSTRAINT  <约束名>  <约束表达式>]
                   [DROP CONSTRAINT  <约束名>];
```

【例 3.8】　假设创建 Out_stock 表时没有同时创建外键，可以再添加外键约束。

```
ALTER TABLE  Out_stock
ADD  CONSTRAINT  FK_Salvaging_Out_stock
FOREIGN KEY (prj_num)    REFERENCES  Salvaging (prj_num);
```

【例 3.9】　删除 Out_stock 表中的外键约束。

```
ALTER TABLE  Out_stock
DROP  CONSTRAINT  FK_Salvaging_Out_stock;
```

或

```
ALTER TABLE  Out_stock
DROP  FOREIGN KEY  FK_Salvaging_Out_stock;
```

【例 3.10】　给 Stock 表添加一个 CHECK 约束：amount > 0。

```
ALTER TABLE Stock  ADD  CONSTRAINT  CK_amount CHECK(amount > 0);
```

【例 3.11】　给 Salvaging 表的 prj_name 列添加一个唯一性约束。

```
ALTER TABLE Salvaging ADD CONSTRAINT  un_prj_name  UNIQUE(prj_name);
```

3.3　查询

数据库查询是数据库的核心操作。SQL 提供了 SELECT 语句进行数据库的查询，该语句具有灵活的使用方式和丰富的功能。一般语法格式为

```
SELECT [ALL|DISTINCT]<目标列表达式>[,<目标列表达式>]…      -- 需要哪些列
FROM <表名或视图名>[,<表名或视图名>]…                      -- 来自哪些表
[WHERE <条件表达式>]                                       -- 根据什么条件
[GROUP BY <列名 1 >[HAVING <条件表达式>]]
[ORDER BY <列名 2 >[ASC|DESC]];
```

整个 SELECT 语句的含义是根据 WHERE 子句给出的条件表达式从 FROM 子句指定的基本表或视图中找出满足条件的元组，再按 SELECT 子句中的目标列表达式选出元组中的属性值形成结果表。如果有 GROUP BY 子句，则将结果按<列名 1 >的值进行分组，该属性列值相等的元组为一组。如果 GROUP BY 子句带 HAVING 短语，则只有满足指定条件的组才能够输出。如果有 ORDER BY 子句，则结果表还要按<列名 2 >的值升序或降序排序。

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。下面以电力抢修工程数据库（见图 3.3）为例说明 SELECT 语句的各种用法。

mat_num	mat_name	speci	warehouse	amount	unit
m001	护套绝缘电线	BVV-120	供电局1#仓库	220	89.80
m002	架空绝缘导线	10KV-150	供电局1#仓库	30	17.00
m003	护套绝缘电线	BVV-35	供电局2#仓库	80	22.80
m004	护套绝缘电线	BVV-50	供电局2#仓库	283	32.00
m005	护套绝缘电线	BVV-70	供电局2#仓库	130	40.00
m006	护套绝缘电线	BVV-150	供电局3#仓库	46	85.00
m007	架空绝缘导线	10KV-120	供电局3#仓库	85	14.08
m008	护套绝缘电线	BVV-95	供电局3#仓库	164	88.00
m009	交联聚乙烯绝缘电缆	YJV22—15KV	供电局4#仓库	45	719.80
m010	户外真空断路器	ZW12-12	供电局4#仓库	1	13600.00

(a) Stock表

prj_num	prj_name	start_date	end_date	prj_status
20100015	220KV清经线接地箱及接地线被盗抢修	2010-10-12 00:...	2010-10-13 00...	1
20100016	沙河站2#公变出线电缆老化烧毁抢修	2010-11-05 00:...	2010-11-06 00...	1
20110001	西丽站电缆短路烧毁抢修工程	2011-01-03 00:...	2011-01-04 00...	1
20110002	西丽站电缆接地抢修	2011-01-03 00:...	2011-01-05 00...	1
20110003	观澜站光缆抢修	2011-02-10 00:...	2011-02-15 00...	1
20110004	小径墩低压线被盗抢修	2011-02-12 00:...	2011-02-17 00...	1
20110005	明珠立交电缆沟盖板破损抢修	2011-03-02 00:...	2011-03-05 00...	0
20110006	朝阳围公变低压线被盗抢修	2011-03-08 00:...	2011-03-10 00...	0

(b) Salvaging表

prj_num	mat_num	amount	get_date	department
20100015	m001	2	2010-10-12 00:00:00	工程1部
20100015	m002	1	2010-10-12 00:00:00	工程1部
20100016	m001	3	2010-11-05 00:00:00	工程1部
20100016	m003	10	2010-11-05 00:00:00	工程1部
20110001	m001	2	2011-01-03 00:00:00	工程2部
20110002	m001	1	2011-01-03 00:00:00	工程2部
20110002	m009	1	2011-01-03 00:00:00	工程2部
20110003	m001	5	2011-02-11 00:00:00	工程3部
20110003	m010	1	2011-02-11 00:00:00	工程3部
20110004	m001	3	2011-02-15 00:00:00	工程3部
20110004	m004	20	2011-02-15 00:00:00	工程3部
20110005	m001	2	2011-03-02 00:00:00	工程2部
20110005	m003	10	2011-03-02 00:00:00	工程2部
20110005	m006	3	2011-03-02 00:00:00	工程2部

(c) Out_stock表

图 3.3　电力抢修工程数据库的数据库示例

扫一扫 视频讲解

3.3.1　单表查询

单表查询指仅涉及一张表的查询。

1. 查询表中的列

扫一扫 视频讲解

查询表中的全部列或部分列，这就是关系代数的投影运算。

1）查询指定的列

在很多情况下，用户只对表中的一部分属性列感兴趣，这时可以在 SELECT 子句的<目标列表达式>中指定要查询的属性列。

【例 3.12】　查询所有配电物资的物资编号、物资名称、规格。

```
SELECT mat_num, mat_name, speci
FROM Stock;
```

查询结果如图 3.4 所示。

【例 3.13】　查询所有配电物资的物资名称、物资编号、规格和所在仓库名称。

```
SELECT mat_name, mat_num, speci, warehouse
FROM Stock;
```

查询结果如图 3.5 所示。

mat_num	mat_name	speci
m001	护套绝缘电线	BVV-120
m002	架空绝缘导线	10KV-150
m003	护套绝缘电线	BVV-35
m004	护套绝缘电线	BVV-50
m005	护套绝缘电线	BVV-70
m006	护套绝缘电线	BVV-150
m007	架空绝缘导线	10KV-120
m008	护套绝缘电线	BVV-95
m009	交联聚乙烯绝缘电缆	YJV22—15KV
m010	户外真空断路器	ZW12-12

图 3.4　例 3.12 查询结果

mat_name	mat_num	speci	warehouse
护套绝缘电线	m001	BVV-120	供电局1#仓库
架空绝缘导线	m002	10KV-150	供电局1#仓库
护套绝缘电线	m003	BVV-35	供电局2#仓库
护套绝缘电线	m004	BVV-50	供电局2#仓库
护套绝缘电线	m005	BVV-70	供电局2#仓库
护套绝缘电线	m006	BVV-150	供电局3#仓库
架空绝缘导线	m007	10KV-120	供电局3#仓库
护套绝缘电线	m008	BVV-95	供电局3#仓库
交联聚乙烯绝缘电缆	m009	YJV22—15KV	供电局4#仓库
户外真空断路器	m010	ZW12-12	供电局4#仓库

图 3.5　例 3.13 查询结果

<目标列表达式>中各个列的先后顺序可以和表中的顺序不一致，用户可以根据应用的需要改变列的显示顺序。

2）查询全部列

如果要查询表中的所有属性列，有两种方法：一种是在<目标列表达式>中列出所有列名；另一种是如果列的显示顺序与其在表中定义的顺序相同，则可以简单地在<目标列表达式>中写星号（*）。

【例 3.14】　查询所有配电物资的记录。

```
SELECT *
FROM Stock;
```

等价于

```
SELECT mat_num, mat_name, speci, warehouse, amount, unit
FROM Stock;
```

3）查询经过计算的值

SELECT 子句中的<目标列表达式>可以是表中存在的属性列，也可以是表达式、字符串常量或函数。

【例 3.15】　查询所有抢修工程的抢修天数。

在 Salvaging 表中只记录了抢修工程的开始日期和结束日期，没有记录抢修天数，但可以通过计算得到，即调用 datediff()函数返回结束日期与开始日期的时间间隔。因此，实现此功能的查询语句为

```
SELECT prj_name, start_date, end_date, datediff(end_date,start_date)
FROM Salvaging;
```

查询结果如图 3.6 所示。

MySQL 8.0 提供了许多系统函数，如数学函数、字符串函数、日期时间函数等。

prj_name	start_date	end_date	datediff(end_date,start_date)
220KV清经线接地箱及接地线被盗抢修	2010-10-12 00:...	2010-10-13 00...	1
沙河站2#公变出线电缆老化烧毁抢修	2010-11-05 00:...	2010-11-06 00...	1
西丽站电缆短路烧毁抢修工程	2011-01-03 00:...	2011-01-04 00...	1
西丽站电缆接地抢修	2011-01-03 00:...	2011-01-05 00...	2
观澜站光缆抢修	2011-02-10 00:...	2011-02-15 00...	5
小径墩低压线被盗抢修	2011-02-12 00:...	2011-02-17 00...	5
明珠立交电缆沟盖板破损抢修	2011-03-02 00:...	2011-03-05 00...	3
朝阳围公变低压线被盗抢修	2011-03-08 00:...	2011-03-10 00...	2

图 3.6　例 3.15 查询结果

MySQL 常用的时间和日期函数如表 3.3 所示。

表 3.3　MySQL 常用的时间和日期函数

函　数	功　能
now()	返回系统当前的日期和时间
year(d)	返回日期 d 中的年份
month(d)	返回日期 d 中的月份
dayofmonth(d)	计算日期 d 是本月的第几天
datediff(d1,d2)	返回 d1 和 d2 之间相隔的天数
adddate(d,n)	返回起始日期 d 加上 n 天的日期
subdate(d,n)	返回起始日期 d 减去 n 天的日期

SQL Server 中常用的时间和日期函数如表 3.4 所示。

表 3.4　SQL Server 中常用的时间和日期函数

函　数	功　能
getdate()	返回系统当前的日期和时间
year(date)	返回一个整数，表示指定日期 date 中的年份
month(date)	返回一个整数，表示指定日期 date 中的月份
day(date)	返回一个整数，表示指定日期 date 中的天数
datediff(datepart,date1,date2)	返回 date1 和 date2 的时间间隔，其单位由 datepart 参数指定

可以看到，经过计算的列、函数的列的显示结果都没有合适的列标题，用户可以通过指定别名改变查询结果的列标题，这对于含算术表达式、函数名的目标列尤其有用。

改变列标题的语法格式为

```
列名|表达式 [AS] 列标题
```

或

```
列标题 = 列名|表达式
```

【例 3.16】　查询所有抢修工程的抢修天数，对各个属性列赋予别名，并在实际抢修天数列前加入一列，此列的每行数据均为“抢修天数”常量值。

```
SELECT prj_name 项目名称, start_date 开始日期, end_date 结束日期,'抢修天数', datediff(end_
date,start_date) 抢修天数
FROM Salvaging;
```

查询结果如图 3.7 所示。

2. 查询表中的若干元组

前面介绍的例子都是查询表中的全部记录，没有对表中的记录行进行任何有条件的筛

项目名称	开始日期	结束日期	抢修天数	抢修天数
220KV青经线接地箱及接地线被盗抢修	2010-10-12 00:00:00	2010-10-13 00:00:00	抢修天数	1
沙河站2#公变出线电缆老化烧毁抢修	2010-11-05 00:00:00	2010-11-06 00:00:00	抢修天数	1
西丽站电缆短路烧毁抢修工程	2011-01-03 00:00:00	2011-01-04 00:00:00	抢修天数	1
西丽站电缆接地抢修	2011-01-03 00:00:00	2011-01-05 00:00:00	抢修天数	2
观澜站光缆抢修	2011-02-10 00:00:00	2011-02-15 00:00:00	抢修天数	5
小径墩低压线被盗抢修	2011-02-12 00:00:00	2011-02-17 00:00:00	抢修天数	5
明珠立交电缆沟盖板破损抢修	2011-03-02 00:00:00	2011-03-05 00:00:00	抢修天数	3
朝阳围公变低压线被盗抢修	2011-03-08 00:00:00	2011-03-10 00:00:00	抢修天数	2

图 3.7　例 3.16 查询结果

选。实际上，在查询过程中，除了可以选择列之外，还可以对行进行选择，使查询的结果更加满足用户的要求。

1）消除取值相同的行

本来在数据库表中不存在取值全部相同的元组，但进行了对列的选择后，查询结果中就有可能出现取值完全相同的行了，而取值相同的行在结果中是没有意义的，因此应消除。

【例 3.17】　在配电物资库存记录表中查询出所有仓库名称。

```
SELECT warehouse
FROM Stock;
```

查询结果如图 3.8 所示。

在这个结果中有许多重复的行（实际上一个仓库存放了多少种物资，其仓库名称就在结果中重复多次）。如果想去掉结果表中的重复行，必须指定 DISTINCT 关键字：

```
SELECT DISTINCT warehouse
FROM Stock;
```

查询结果如图 3.9 所示。

warehouse
供电局1#仓库
供电局1#仓库
供电局2#仓库
供电局2#仓库
供电局2#仓库
供电局3#仓库
供电局3#仓库
供电局3#仓库
供电局4#仓库
供电局4#仓库

图 3.8　例 3.17 查询结果（1）

warehouse
供电局1#仓库
供电局2#仓库
供电局3#仓库
供电局4#仓库

图 3.9　例 3.17 查询结果（2）

DISTINCT 关键字在 SELECT 之后，目标列表达式之前。如果没有指定 DISTINCT 关键字，则默认为 ALL，即保留结果表中取值重复的行。

```
SELECT warehouse
FROM Stock;
```

等价于

```
SELECT ALL warehouse
FROM Stock;
```

2）查询满足条件的元组

查询满足条件的元组是通过 WHERE 子句实现的。WHERE 子句常用的查询条件如表 3.5 所示。

表 3.5　WHERE 子句常用的查询条件

查 询 条 件	谓　　词
比较(比较运算符)	=、>、<、>=、<=、!=、<>、!>、!<；NOT+上述比较运算符
确定范围	BETWEEN AND、NOT BETWEEN AND
确定集合	IN、NOT IN
字符匹配	LIKE、NOT LIKE
空值	IS NULL、IS NOT NULL
多重条件(逻辑谓词)	AND、OR

(1) 比较大小的查询。

【例 3.18】　查询供电局 1＃仓库存放的所有物资编号、物资名称、规格以及数量。

```
SELECT mat_num,mat_name,speci,amount
FROM Stock
WHERE warehouse = '供电局 1＃仓库';
```

查询结果如图 3.10 所示。

mat_num	mat_name	speci	amount
m001	护套绝缘电线	BVV-120	220
m002	架空绝缘导线	10KV-150	30

图 3.10　例 3.18 查询结果

RDBMS 执行该查询的一种可能过程是对 Stock 表进行全表扫描，取出一个元组，检查该元组在 warehouse 列上的值是否等于“供电局 1＃仓库”。如果相等，则取出 mat_num、mat_name、speci、amount 列的值形成一个新的元组输出；否则跳过该元组，取下一个元组。

如果 Stock 表中有数万种配电物资，供电局 1＃仓库存放的物资种类是所有物资的 10%左右，可以在 warehouse 列上建立索引，系统会利用该索引找出 warehouse ='供电局 1＃仓库'的元组，从中取出 mat_num、mat_name、speci、amount 列的值形成结果关系。这就避免了对 Stock 表的全表扫描，可以加快查询速度。但需要注意的是，如果物资种类较少，索引查找则不一定能提高查询效率，系统仍会使用全表扫描。这由查询优化器按照某些规则或估计执行代价进行选择。

【例 3.19】　查询所有单价小于 80 元的物资名称、数量及其单价。

```
SELECT mat_name,amount,unit
FROM Stock
WHERE unit < 80;
```

或

```
SELECT mat_name,amount,unit
FROM Stock
WHERE NOT unit > = 80;
```

查询结果如图 3.11 所示。

(2) 确定范围的查询。BETWEEN…AND 和 NOT BETWEEN…AND 是一个逻辑运算符，可以用来查找属性值在或不在指定范围内的元组，其中 BETWEEN 后指定范围的下限(即低值)，AND 后指定范围的上限(即高值)。

【例 3.20】　查询单价为 50～100 元的物资名称、数量及其单价。

```
SELECT mat_name, amount, unit
FROM Stock
WHERE unit BETWEEN 50 AND 100;
```

等价于

```
SELECT mat_name, amount, unit
FROM Stock
WHERE unit >= 50 AND unit <= 100;
```

查询结果如图 3.12 所示。

mat_name	amount	unit
架空绝缘导线	30	17.00
护套绝缘电线	80	22.80
护套绝缘电线	283	32.00
护套绝缘电线	130	40.00
架空绝缘导线	85	14.08

图 3.11　例 3.19 查询结果

mat_name	amount	unit
护套绝缘电线	220	89.80
护套绝缘电线	46	85.00
护套绝缘电线	164	88.00

图 3.12　例 3.20 查询结果

【例 3.21】　查询单价不在 50～100 元的物资名称、数量及其单价。

```
SELECT mat_name, amount, unit
FROM Stock
WHERE unit NOT BETWEEN 50 AND 100;
```

等价于

```
SELECT mat_name, amount, unit
FROM Stock
WHERE unit < 50 OR unit > 100;
```

（3）确定集合的查询。IN 是一个逻辑运算符，可以用来查找属性值属于指定集合的元组。

【例 3.22】　查询存放在供电局 1＃仓库和供电局 2＃仓库的物资名称、规格、数量及仓库。

```
SELECT mat_name, speci, amount, warehouse
FROM Stock
WHERE warehouse IN ('供电局 1＃仓库','供电局 2＃仓库');
```

等价于

```
SELECT mat_name, speci, amount, warehouse
FROM Stock
WHERE warehouse = '供电局 1＃仓库'OR warehouse = '供电局 2＃仓库';
```

【例 3.23】　查询既没有存放在供电局 1＃仓库，也没有存放在供电局 2＃仓库的物资名称、规格、数量及仓库。

```
SELECT mat_name, speci, amount, warehouse
FROM Stock
WHERE warehouse NOT IN ('供电局 1＃仓库','供电局 2＃仓库');
```

等价于

```
SELECT mat_name, speci, amount, warehouse
FROM Stock
WHERE warehouse != '供电局 1＃仓库' AND warehouse != '供电局 2＃仓库';
```

（4）字符匹配的查询。LIKE 运算符用于查找指定列名与匹配串常量匹配的元组。匹配串是一种特殊的字符串，其特殊之处在于它不仅可以包含普通字符，还可以包含通配符。通配符用于表示任意的字符或字符串。在实际应用中，如果需要从数据库中检索一批记录，但又不能给出精确的字符查询条件，这时就可以使用 LIKE 运算符和通配符实现模糊查询。

在 LIKE 运算符前也可以使用 NOT 运算符，表示对结果取反。一般语法格式如下。

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

其含义是查找指定的属性列值与<匹配符>相匹配的元组。<匹配符>可以是一个完整的空符串，也可以含有通配符“%”和“_”。

- %（百分号）代表任意长度（长度可以为 0）的字符串。例如，a%b 表示以 a 开头，以 b 结尾的任意长度的字符串，如 acb、addgb、ab 等都满足该匹配串。
- _（下横线）代表任意单个字符。例如，a_b 表示以 a 开头，以 b 结尾的长度为 3 的任意字符串，如 acb、afb 等都满足该匹配串。

【例 3.24】 查询存放在供电局 1＃仓库的物资的详细情况。

```
SELECT *
FROM Stock
WHERE warehouse LIKE '供电局 1＃仓库';
```

等价于

```
SELECT *
FROM Stock
WHERE warehouse = '供电局 1＃仓库';
```

如果 LIKE 后的匹配串中不含通配符，则可以用“=”（等于）运算符取代 LIKE 谓词，用“!=”或“< >”（不等于）运算符取代 NOT LIKE 谓词。

【例 3.25】 查询物资名称中含有“绝缘电线”的物资编号、名称和规格。

```
SELECT mat_num, mat_name, speci
FROM Stock
WHERE mat_name LIKE '%绝缘电线';
```

【例 3.26】 查询物资名称中第 3、4 个字为“绝缘”的物资编号、名称和规格。

```
SELECT mat_num, mat_name, speci
FROM Stock
WHERE mat_name LIKE '__绝缘%';
```

查询结果如图 3.13 所示。

mat_num	mat_name	speci
m001	护套绝缘电线	BVV-120
m002	架空绝缘电线	10KV-150
m003	护套绝缘电线	BVV-35
m004	护套绝缘电线	BVV-50
m005	护套绝缘电线	BVV-70
m006	护套绝缘电线	BVV-150
m007	架空绝缘电线	10KV-120
m008	护套绝缘电线	BVV-95

图 3.13 例 3.26 查询结果

【例 3.27】 查询所有物资名称不带“绝缘”两个字的物资编号、名称和规格。

```
SELECT mat_num, mat_name, speci
FROM Stock
WHERE mat_name NOT LIKE '%绝缘%';
```

如果用户要查询的字符串本身就含有“%”或“_”，这时就要使用 ESCAPE'<换码字符>'对通配符进行转义了。

【例 3.28】 查询物资名称中含有“户外_真空”字样的物资信息。

```
SELECT *
FROM Stock
WHERE mat_name LIKE '%户外/_真空%' ESCAPE '/';
```

ESCAPE'/'短语表示“/”为换码字符，这样匹配串中紧跟在“/”后面的字符“_”不再具有通配符的含义，转义为普通的“_”字符。

(5) 涉及空值的查询。空值(NULL)在数据库中有特殊的含义,它表示不确定的值。例如,某些物资现没有单价,因此单价为空值。判断某个值是否为 NULL,不能使用普通的比较运算符(=、!=等),只能使用专门的判断空值的子句来完成。

判断取值为空的语法格式:列名 IS NULL。

判断取值不为空的语法格式:列名 IS NOT NULL。

【例 3.29】　查询无库存单价的物资编号及其名称。

```
SELECT mat_num, mat_name
FROM Stock
WHERE unit IS NULL;
```

注意:这里的 IS 不能用等号(=)代替。

(6) 多重条件查询。在 WHERE 子句中可以使用逻辑运算符 AND 和 OR 组成多条件查询。用 AND 连接的条件表示必须全部满足所有条件的结果才为真,用 OR 连接的条件表示只要满足其中一个条件结果即为真。AND 的优先级高于 OR,但用户可以用括号改变优先级。

【例 3.30】　查询规格为 BVV-120 的护套绝缘电线的物资编号、库存数量及库存地点。

```
SELECT mat_num,warehouse,amount
FROM Stock
WHERE mat_name = '护套绝缘电线'  AND speci = 'BVV - 120';
```

3. 对查询结果进行排序

有时希望查询结果能按一定的顺序显示出来,如按单价从高到低排列库存物资。SQL 语句支持将查询的结果按用户指定的列进行排序的功能,而且查询结果可以按一个列排序,也可以按多个列排序;排序可以是从小到大(升序),也可以是从大到小(降序)。排序子句的语法格式为

```
ORDER BY <列名> [ASC|DESC][,...n]
```

其中,<列名>为排序的依据列,可以是列名或列的别名;ASC 表示对列进行升序排列;DESC 表示对列进行降序排列。如果没有指定排序方式,则默认为升序排序。

如果在 ORDER BY 子句中使用多个列进行排序,则这些列在该子句中出现的顺序决定了对结果集进行排序的方式。当指定多个排序依据列时,首先安排在最前面的列进行排序,如果排序后存在两个或两个以上列值相同的记录,则对这些值相同的记录再依据排在第 2 位的列进行排序,以此类推。

【例 3.31】　查询护套绝缘电线的物资名称及其单价,查询结果按单价降序排列。

```
SELECT mat_name,unit
FROM Stock
WHERE mat_name = '护套绝缘电线' ORDER BY unit DESC;
```

【例 3.32】　查询所有物资的信息,查询结果按所在仓库名降序排列,同一仓库的物资按库存量升序排列。

```
SELECT *
FROM Stock
ORDER BY warehouse DESC, amount;
```

查询结果如图 3.14 所示。

mat_num	mat_name	speci	warehouse	amount	unit
m010	户外真空断路器	ZW12-12	供电局4#仓库	1	13600.00
m009	交联聚乙烯绝缘电缆	YJV22—15KV	供电局4#仓库	45	719.80
m006	护套绝缘电线	BVV-150	供电局3#仓库	46	85.00
m007	架空绝缘导线	10KV-120	供电局3#仓库	85	14.08
m008	护套绝缘电线	BVV-95	供电局3#仓库	164	88.00
m003	护套绝缘电线	BVV-35	供电局2#仓库	80	22.80
m005	护套绝缘电线	BVV-70	供电局2#仓库	130	40.00
m004	护套绝缘电线	BVV-50	供电局2#仓库	283	32.00
m002	架空绝缘导线	10KV-150	供电局1#仓库	30	17.00
m001	护套绝缘电线	BVV-120	供电局1#仓库	220	89.80

图 3.14　例 3.32 查询结果

空值被认为是最小的，因此，若按升序排列，含空值的元组将最先显示；若按降序排列，含空值的元组将最后显示。

4. 限制查询结果

如果 SELECT 语句查询结果数据过多，可以使用 LIMIT 关键字限制 SELECT 查询返回的记录总数，有以下两种使用方式。

（1）LIMIT 显示记录数量：从第 1 条记录开始，显示指定条数的记录。

（2）LIMIT 初始位置，显示记录数量：从指定的初始位置开始显示指定条数的记录。

【例 3.33】　显示 Stock 表中库存量最大的两条记录。

```
SELECT *
FROM Stock
ORDER BY amount DESC
LIMIT 2;
```

查询结果如图 3.15 所示。

mat_num	mat_name	speci	warehouse	amount	unit
m004	护套绝缘电线	BVV-50	供电局2#仓库	283	32.00
m001	护套绝缘电线	BVV-120	供电局1#仓库	220	89.80

图 3.15　例 3.33 查询结果

说明：SQL Server 中，实现该功能的关键字为 TOP，本例在 SQL Server 中的实现语句为

```
SELECT top 2 *
FROM Stock
ORDER BY amount DESC
```

SQL Server 还可以按百分比显示表中的记录，如以下语句显示 Stock 表中排序前 10% 的记录。

```
SELECT top 10 PERCENT *
FROM Stock
ORDER BY amount DESC
```

【例 3.34】　显示 Stock 表中的 5 条记录，指定从第 3 条记录开始显示。

```
SELECT *
FROM Stock
LIMIT 3,5;
```

5. 聚集函数

为了进一步方便用户，增强检索功能，SQL 提供了许多聚集函数（Aggregate Functions），如表 3.6 所示。聚集函数的作用是对一组值进行计算并返回一个单值。

表 3.6　聚集函数

函　数　名	功　　能
COUNT(*)	统计表中元组的个数
COUNT([DISTINCT|ALL]<列名>)	统计一列中值的个数
SUM([DISTINCT|ALL]<列名>)	计算一列中值的总和(此列必须是数值型)
AVG([DISTINCT|ALL]<列名>)	计算一列中值的平均值(此列必须是数值型)
MAX([DISTINCT|ALL]<列名>)	求一列中值的最大值
MIN([DISTINCT|ALL]<列名>)	求一列中值的最小值

如果指定 DISTINCT 短语,则表示在计算时要消除指定列中的重复值;如果不指定 DISTINCT 短语或指定 ALL 短语(ALL 为默认值),则表示不消除重复值。

【例 3.35】　统计领取了物资的抢修工程项目数。

```
SELECT COUNT(DISTINCT prj_num)
FROM Out_stock;
```

抢修工程每领取一种物资,在 Out_stock 表中都有一条相应的记录。一个抢修工程要领取多种物资,为避免重复计算抢修工程项目数,必须在 COUNT()函数中使用 DISTINCT 短语。

【例 3.36】　查询使用 m001 号物资的抢修工程项目数,以及物资最大领取数量、最小领取数量以及平均领取数量。

```
SELECT COUNT( * ),MAX(amount), MIN(amount), AVG(amount)
FROM  Out_stock
WHERE mat_num = 'm001';
```

注意:聚集函数在计算过程中忽略空值;WHERE 子句不能使用聚集函数作为条件表达式。

6. 开窗函数

数据分析和统计中经常需要用到开窗函数,通常和聚集函数联合使用。

【例 3.37】　查询每种物资按时间顺序的累计订单金额。

```
SELECT * ,SUM(amount) OVER(PARTITION BY mat_num ORDER BY get_date) sum_amount
FROM Out_stock;
```

查询结果的最后一列是按照物资编号的时间顺序累计 amount 的总量,如图 3.16 所示。

prj_num	mat_num	amount	get_date	department	sum_amount
20100015	m001	2	2010-10-12 00:00:00	工程1部	2
20100016	m001	3	2010-11-05 00:00:00	工程1部	5
20110001	m001	2	2011-01-03 00:00:00	工程2部	8
20110002	m001	1	2011-01-03 00:00:00	工程2部	8
20110003	m001	5	2011-02-11 00:00:00	工程3部	13
20110004	m001	3	2011-02-15 00:00:00	工程3部	16
20110005	m001	2	2011-03-02 00:00:00	工程2部	18
20100015	m002	1	2010-10-12 00:00:00	工程1部	1
20100016	m003	10	2010-11-05 00:00:00	工程1部	10
20110005	m003	10	2011-03-02 00:00:00	工程2部	20
20110004	m004	20	2011-02-15 00:00:00	工程3部	20
20110005	m006	3	2011-03-02 00:00:00	工程2部	3
20110002	m009	1	2011-01-03 00:00:00	工程2部	1
20110003	m010	1	2011-02-11 00:00:00	工程3部	1

图 3.16　例 3.37 查询结果

什么是开窗呢?可以理解为记录集合,开窗函数也就是在满足某种条件的记录集合上执行的特殊函数。对于每条记录,都要在此窗口内执行函数,有的函数记录不同,窗口大小都是固定的,这种属于静态窗口;有的函数则相反,不同的记录对应着不同的窗口,这种动态变化的窗口叫作滑动窗口。开窗函数的本质还是聚合运算,只不过它更具灵活性,它对数据的每行都使用与该行相关的行进行计算并返回计算结果。

聚合函数是将多条记录聚合为一条;而开窗函数是每条记录都会执行,有几条记录,执行完还是几条。具体的开窗函数如表 3.7 所示。

表 3.7　开窗函数

函　数　名	功　　能
CUME_DIST()	计算一组值中一个值的累积分布
DENSE_RANK()	根据 ORDER BY 子句为分区中的每行分配一个等级，它将相同的等级分配给具有相等值的行。如果两行或更多行具有相同的等级，则等级序列中将没有间隙
FIRST_VALUE()	返回相对于窗口框架第 1 行的指定表达式的值
LAG()	返回分区中当前行之前的第 *N* 行的值。如果不存在前一行，则返回 NULL
LAST_VALUE()	返回相对于窗口框架最后一行的指定表达式的值
LEAD()	返回分区中当前行之后的第 *N* 行的值。如果不存在后续行，则返回 NULL
NTH_VALUE()	从窗口框架的第 *N* 行返回参数的值
NTILE()	将每个窗口分区的行分配到指定数量的排名组中
PERCENT_RANK()	计算分区或结果集中行的百分数等级
RANK()	与 DENSE_RANK()函数相似，不同之处在于当两行或多行具有相同的等级时，等级序列中存在间隙
ROW_NUMBER()	为分区中的每行分配一个顺序整数

【例 3.38】　查询每种物资使用数量最高的前两条出库情况。

```
SELECT *
FROM (
     SELECT *,
     ROW_NUMBER() OVER(PARTITION BY mat_num ORDER BY amount desc) AS row_num
     FROM Out_stock) AS t
WHERE row_num <= 2;
```

查询结果如图 3.17 所示。

prj_num	mat_num	amount	get_date	department	row_num
20110003	m001	5	2011-02-11 00:00:00	工程3部	1
20100016	m001	3	2010-11-05 00:00:00	工程1部	2
20100015	m002	1	2010-10-12 00:00:00	工程1部	1
20100016	m003	10	2010-11-05 00:00:00	工程1部	1
20110005	m003	10	2011-03-02 00:00:00	工程2部	2
20110004	m004	20	2011-02-15 00:00:00	工程3部	1
20110005	m006	3	2011-03-02 00:00:00	工程2部	1
20110002	m009	1	2011-01-03 00:00:00	工程2部	1
20110003	m010	1	2011-02-11 00:00:00	工程3部	1

图 3.17　例 3.38 查询结果

7. 对查询结果进行分组

有时并不是对全表进行计算，而是根据需要对数据进行分组，然后再对每个组进行计算。例如，统计每个抢修工程使用的物资种类，这时就需要用到 GROUP BY 分组子句。GROUP BY 子句将查询结果按某一列或多列的值分组，值相等的为一组。分组的目的是细化聚集函数的作用对象。需要注意的是，如果使用了分组子句，则查询列表中的每列必须是分组依据列（在 GROUP BY 后的列）或聚集函数。

使用 GROUP BY 子句时，如果在 SELECT 的查询列表中包含聚集函数，则是针对每个组计算出一个汇总值，从而实现对查询结果的分组统计。

分组语句跟在 WHERE 子句的后面，一般语法格式如下。

```
GROUP BY <分组依据列>[,...n]
[HAVING <组提取条件>]
```

HAVING 子句用于对分组后的结果进行过滤，它的功能有点像 WHERE 子句，但它用于组，而不是用于单个记录。在 HAVING 子句中可以使用聚集函数，但在 WHERE 子句中则不能。HAVING 子句通常与 GROUP BY 子句一起使用。

【例 3.39】　查询每个抢修工程项目号及使用的物资种类。

```
SELECT prj_num 项目号, COUNT( * ) 物资种类
FROM Out_stock
GROUP BY prj_num;
```

首先对查询结果按 prj_num 的值分组，所有具有相同 prj_num 值的元组归为一组，然后再对每组使用 COUNT()函数进行计算。查询结果如图 3.18 所示。

【例 3.40】　查询使用两种及两种以上物资的抢修工程项目号。

```
SELECT prj_num 项目号
FROM Out_stock
GROUP BY prj_num
HAVING COUNT( * )>=2;
```

首先用 GROUP BY 子句对 prj_num 进行分组，然后用聚集函数 COUNT()分别对每组进行统计，最后挑选出统计结果满足大于或等于 2 的元组的 prj_num。查询结果如图 3.19 所示。

WHERE 子句与 HAVING 子句的区别在于作用对象不同。WHERE 子句作用于基本表或视图，从中选择满足条件的元组；而 HAVING 子句作用于组，从中选择满足条件的组。

GROUP BY 子句的分组字段也可以包含多个属性列名。

项目号	物资种类
20100015	2
20100016	2
20110001	1
20110002	2
20110003	2
20110004	2
20110005	3

图 3.18　例 3.39 查询结果

项目号
20100015
20100016
20110002
20110003
20110004
20110005

图 3.19　例 3.40 查询结果

【例 3.41】　按工程部门及物资编号统计其抢修的项目个数以及对应的领取总量。

```
SELECT department,mat_num,COUNT(DISTINCT prj_num) 项目个数 ,SUM(amount) 领取总量
FROM Out_stock
GROUP BY department,mat_num
```

首先按 GROUP BY 子句的第 1 个分组字段 department 进行分组，同一组内的再按第 2 个分组字段 mat_num 进行分组，然后使用统计函数分别对每组进行统计。查询结果如图 3.20 所示。

department	mat_num	项目个数	领取总量
工程1部	m001	2	5
工程1部	m002	1	1
工程1部	m003	1	10
工程2部	m001	3	5
工程2部	m003	1	10
工程2部	m006	1	3
工程2部	m009	1	1
工程3部	m001	2	8
工程3部	m004	1	20
工程3部	m010	1	1

图 3.20　例 3.41 查询结果

MySQL 和 SQL Server 都支持带 ROLLUP 关键字的分类汇总，这种用法在数据统计中经常用到，尤其是在制作报表时，ROLLUP 关键字按照分组顺序，先对第 1 个分组字段分组，在组内进行统计，最后给出合计。

【例 3.42】　按工程部门及物资编号统计其抢修的项目个数以及对应的领取总量，要求带 ROLLUP 关键字。

```
SELECT department,mat_num,COUNT(DISTINCT prj_num) 项目个数 ,SUM(amount) 领取总量
FROM Out_stock
GROUP BY department,mat_num
WITH ROLLUP
```

本例的处理过程在例 3.41 的基础上增加了对 GROUP BY 子句的第 1 个分组字段 department 的合计，即统计了每个工程部门抢修的项目总数以及物资领取总量，最后又给出了所有工程部门抢修的项目总数以及物资领取总量。查询结果如图 3.21 所示。

department	mat_num	项目个数	领取总里
工程1部	m001	2	5
工程1部	m002	1	1
工程1部	m003	1	10
工程1部	NULL	2	16
工程2部	m001	3	5
工程2部	m003	1	10
工程2部	m006	1	3
工程2部	m009	1	1
工程2部	NULL	3	19
工程3部	m001	2	8
工程3部	m004	1	20
工程3部	m010	1	1
工程3部	NULL	2	29
NULL	NULL	7	64

图 3.21　例 3.42 查询结果

8. 正则表达式查询

正则表达式通常用于检索或替换那些符合某个模式的文本内容，根据指定的匹配模式匹配文本中符合要求的特殊字符串。正则表达式的查询能力比通配字符的查询能力更强，而且更加灵活。在 MySQL 中可以使用 REGEXP 关键字指定正则表达式的字符串匹配模式，基本语法格式如下。

```
列名　REGEXP　'匹配方式'
```

其中，列名表示需要查询的字段名称；匹配方式表示以哪种方式进行匹配查询。

正则表达式的模式字符如表 3.8 所示。

表 3.8　正则表达式的模式字符

模 式 字 符	说　　明
^	匹配字符串开始的部分
$	匹配字符串结束的部分
.	代表字符串中的任意一个字符，包括回车和换行
[字符集合]	匹配字符集合中的任何一个字符
[^字符集合]	匹配除了字符集合以外的任何一个字符
S1|S2|S3	匹配 S1、S2、S3 中的任意一个字符串
*	匹配 0 个或多个在它前面的字符
+	匹配前面的字符 1 次或多次
字符串{N}	字符串出现 N 次
字符串{M,N}	字符串出现至少 M 次，最多 N 次

【例 3.43】　查询项目名称中包含“西丽”“明珠”的项目信息。

```
SELECT *
FROM Salvaging
WHERE prj_name REGEXP '西丽|明珠'
```

查询结果如图 3.22 所示。

prj_num	prj_name	start_date	end_date	prj_status
20110001	西丽站电缆短路烧毁抢修工程	2011-01-03 00:...	2011-01-04 00...	1
20110002	西丽站电缆接地抢修	2011-01-03 00:...	2011-01-05 00...	1
20110005	明珠立交电缆沟盖板破损抢修	2011-03-02 00:...	2011-03-05 00...	0

图 3.22　例 3.43 查询结果

【例 3.44】　查询项目名称中包含字符串“2”至少一次，最多两次的项目信息。

```
SELECT *
FROM Salvaging
WHERE prj_name REGEXP '2{1,2}'
```

说明：SQL Server 中，主要有 regexp_like、regexp_replace、regexp_substr、regexp_instr 这 4 个正则表达式函数。

扫一扫

视频讲解

3.3.2　连接查询

前面介绍的查询都是针对一张表进行的，但有时需要从多张表中获取信息，因此就会涉及多张表。若一个查询涉及两张或两张以上的表，则称为连接查询。连接查询是关系数据库中最主要的查询，主要包括等值连接、非等值连接、自然连接、内连接、外连接、复合条件连接、自身连接查询。

1. 等值与非等值连接查询

不同表之间的连接查询，重点是 WHERE 子句中的连接条件及两个表的属性列名。连接查询中用来连接两张表的条件称为连接条件或连接谓词，其连接条件一般语法格式为

```
[<表 1>.]<列名 1> <比较运算符> [<表 2>.]<列名 2>
```

其中，比较运算符主要有＝、>、<、>＝、<＝、!＝(或< >)。

此外，连接条件还可以使用以下形式。

```
[<表 1>.]<列名 1> BETWEEN [<表 2>.] <列名 2> AND [<表 2>.]<列名 3>
```

当连接的比较运算符为“＝”时，称为等值连接；其他为非等值连接。在连接条件中列名对应属性的类型必须是可比的，但不必是相同的。

从概念上讲，DBMS 执行连接操作的过程是首先在表 1 中找到第 1 个元组，然后从头开始扫描表 2，逐一查找满足连接条件的元组，找到后就将表 1 中的第 1 个元组与该元组拼接起来，形成结果表中的一个元组。表 2 全部查找完后，再找表 1 中的第 2 个元组，然后从头开始扫描表 2，逐一查找满足连接条件的元组，找到后将表 1 中的第 2 个元组与该元组拼接起来，形成结果表中的一个元组。重复上述操作，直到表 1 中的全部元组都处理完毕为止。

【例 3.45】　查询每个抢修工程及其领料出库的情况。

抢修工程情况存放在 Salvaging 表中，领料出库情况存放在 Out_stock 表中，所以查询实际上涉及 Salvaging 与 Out_stock 两张表，这两张表之间的联系是通过公共属性 prj_num 实现的。

```
SELECT Salvaging.*, Out_stock.*
FROM Salvaging, Out_stock
WHERE Salvaging.prj_num = Out_stock.prj_num
```

查询结果如图 3.23 所示。

prj_num	prj_name	start_date	end_date	prj_status	prj_num	mat_num	amount	get_date	department
20100015	220KV清经线接地箱及接地线被盗抢修	2010-10-12 00:...	2010-10-13 00...	1	20100015	m001	2	2010-10-12 00:00:00	工程1部
20100015	220KV清经线接地箱及接地线被盗抢修	2010-10-12 00:...	2010-10-13 00...	1	20100015	m002	1	2010-10-12 00:00:00	工程1部
20100016	沙河站2#公变出线电缆老化烧毁抢修	2010-11-05 00:...	2010-11-06 00...	1	20100016	m001	3	2010-11-05 00:00:00	工程1部
20100016	沙河站2#公变出线电缆老化烧毁抢修	2010-11-05 00:...	2010-11-06 00...	1	20100016	m003	10	2010-11-05 00:00:00	工程1部
20110001	西丽站电缆短路烧毁抢修工程	2011-01-03 00:...	2011-01-04 00...	1	20110001	m001	2	2011-01-03 00:00:00	工程2部
20110002	西丽站电缆接地抢修	2011-01-03 00:...	2011-01-05 00...	1	20110002	m001	1	2011-01-03 00:00:00	工程2部
20110002	西丽站电缆接地抢修	2011-01-03 00:...	2011-01-05 00...	1	20110002	m009	1	2011-01-03 00:00:00	工程2部
20110003	观澜站光缆抢修	2011-02-10 00:...	2011-02-15 00...	1	20110003	m001	5	2011-02-11 00:00:00	工程3部
20110003	观澜站光缆抢修	2011-02-10 00:...	2011-02-15 00...	1	20110003	m010	1	2011-02-11 00:00:00	工程3部
20110004	小径墩低压线被盗抢修	2011-02-12 00:...	2011-02-17 00...	1	20110004	m001	3	2011-02-15 00:00:00	工程3部
20110004	小径墩低压线被盗抢修	2011-02-12 00:...	2011-02-17 00...	1	20110004	m004	20	2011-02-15 00:00:00	工程3部
20110005	明珠立交电缆沟盖板破损抢修	2011-03-02 00:...	2011-03-05 00...	0	20110005	m001	2	2011-03-02 00:00:00	工程2部
20110005	明珠立交电缆沟盖板破损抢修	2011-03-02 00:...	2011-03-05 00...	0	20110005	m003	10	2011-03-02 00:00:00	工程2部
20110005	明珠立交电缆沟盖板破损抢修	2011-03-02 00:...	2011-03-05 00...	0	20110005	m006	3	2011-03-02 00:00:00	工程2部

图 3.23　例 3.45 查询结果

本例中，SELECT 子句与 WHERE 子句中的属性名前都加上了表名前缀，这是为了避免混淆。如果属性名在参加连接的各表中是唯一的，则可以省略表名前缀。

在连接运算中有两种特殊情况，一种为自然连接，另一种为广义笛卡儿积(连接)。

广义笛卡儿积是不带连接谓词的连接。两张表的广义笛卡儿积即是两表中元组的交叉乘积，其连接的结果会产生一些没有意义的元组，所以这种运算实际上很少使用。

若在等值连接中把目标列中重复的属性列去掉，则为自然连接。

【例 3.46】 对例 3.45 用自然连接完成查询。

```
SELECT Salvaging.prj_num,prj_name,start_date,end_date,prj_status, mat_num,
amount,get_date,department
FROM Salvaging, Out_stock
WHERE Salvaging.prj_num = Out_stock.prj_num
```

本例中，由于 prj_name、start_date、end_date、prj_status、mat_num、amount、get_date 和 department 属性列在 Salvaging 表与 Out_stock 表中是唯一的，因此引用时可以去掉表名前缀；而 prj_num 在两张表中都出现了，因此引用时必须加上表名前缀。

2. 内连接查询

使用 INNER JOIN 和 JOIN 连接均可，重点是要有查询条件，条件使用 ON 或 WHERE 引导查询，查询出的结果为两表都匹配的记录。

【例 3.47】 对例 3.45 用内连接完成查询。

```
SELECT Salvaging.prj_num,prj_name,start_date,end_date,prj_status, mat_num,
amount,get_date,department
FROM Salvaging INNER JOIN Out_stock ON Salvaging.prj_num = Out_stock.prj_num
```

例 3.46 中的 WHERE 方式为隐性连接，而例 3.47 中的 INNER JOIN 方式为显性连接。两者虽然在查询结果集上是等同的，但实际上在执行过程上却是不同的。其中，隐性连接在 FROM 过程中对所有表进行笛卡儿积，最终通过 WHERE 条件过滤；显性连接在每次表连接时通过 ON 过滤，筛选后的结果集再和下一张表作笛卡儿积，以此循环。

也就是说，显性连接最终得到笛卡儿积的数量可能会远远小于隐性连接所要扫描的数量，所以同样是等值连接，显性连接的效率更高。

3. 外连接查询

在通常的连接操作中，只有满足连接条件的元组才能作为结果输出，如例 3.45 的结果表中没有编号为 20110006 的工程项目信息，原因在于该工程项目没有领料，在 Out_stock 表中没有相应的元组。若以 Salvaging 表为主体列出每个工程的基本情况及其领料情况，没有领料的工程也希望输出其基本信息，这时就需要使用外连接(Outer Join)。外连接分为左外连接、右外连接和全外连接 3 种类型。

(1) 左外连接：LEFT OUTER JOIN，其结果是列出左边关系中所有元组，而不仅仅是连接属性所匹配的元组。

(2) 右外连接：RIGHT OUTER JOIN，其结果是列出右边关系中所有元组。

(3) 全外连接：FULL OUTER JOIN，其结果是列出左边关系和右边关系中的所有元组。

【例 3.48】 将例 3.45 中的等值连接改为左外连接。

```
SELECT Salvaging.prj_num,prj_name,start_date,end_date,prj_status, mat_num,
amount,get_date,department
FROM Salvaging LEFT OUTER JOIN Out_stock
ON Salvaging.prj_num = Out_stock.prj_num
```

查询结果如图 3.24 所示。请注意区分图 3.23 和图 3.24 的不同之处。

prj_num	prj_name	start_date	end_date	prj_status	mat_num	amount	get_date	department
20100015	220KV清经线接地箱及接地线被盗抢修	2010-10-12 00:...	2010-10-13 00...	1	m001	2	2010-10-12 00:00:00	工程1部
20100015	220KV清经线接地箱及接地线被盗抢修	2010-10-12 00:...	2010-10-13 00...	1	m002	1	2010-10-12 00:00:00	工程1部
20100016	沙河站2#公变出线电缆老化烧毁抢修	2010-11-05 00:...	2010-11-06 00...	1	m001	3	2010-11-05 00:00:00	工程1部
20100016	沙河站2#公变出线电缆老化烧毁抢修	2010-11-05 00:...	2010-11-06 00...	1	m003	10	2010-11-05 00:00:00	工程1部
20110001	西丽站电缆短路烧毁抢修工程	2011-01-03 00:...	2011-01-04 00...	1	m001	2	2011-01-03 00:00:00	工程2部
20110002	西丽站电缆接地抢修	2011-01-03 00:...	2011-01-05 00...	1	m001	1	2011-01-03 00:00:00	工程2部
20110002	西丽站电缆接地抢修	2011-01-03 00:...	2011-01-05 00...	1	m009	1	2011-01-03 00:00:00	工程2部
20110003	观澜站光缆抢修	2011-02-10 00:...	2011-02-15 00...	1	m001	5	2011-02-11 00:00:00	工程3部
20110003	观澜站光缆抢修	2011-02-10 00:...	2011-02-15 00...	1	m010	1	2011-02-11 00:00:00	工程3部
20110004	小径墩低压线被盗抢修	2011-02-12 00:...	2011-02-17 00...	1	m001	3	2011-02-15 00:00:00	工程3部
20110004	小径墩低压线被盗抢修	2011-02-12 00:...	2011-02-17 00...	1	m004	20	2011-02-15 00:00:00	工程3部
20110005	明珠立交电缆沟盖板破损抢修	2011-03-02 00:...	2011-03-05 00...	0	m001	2	2011-03-02 00:00:00	工程2部
20110005	明珠立交电缆沟盖板破损抢修	2011-03-02 00:...	2011-03-05 00...	0	m003	10	2011-03-02 00:00:00	工程2部
20110005	明珠立交电缆沟盖板破损抢修	2011-03-02 00:...	2011-03-05 00...	0	m006	3	2011-03-02 00:00:00	工程2部
20110006	朝阳围公变低压线被盗抢修	2011-03-08 00:...	2011-03-10 00...	0	NULL	NULL	NULL	NULL

图 3.24　例 3.48 查询结果

外连接就好像是为符号 * 一边的表(本例是 Out_stock 表)增加一个“万能”的行,该行全部由空值组成,它可以和另一边的表(本例是 Salvaging 表)中不满足连接条件的元组进行连接。由于这个“万能”行的各列全部是空值,因此在本例的连接结果中有一行来自 Out_stock 表的属性值全部是空值。

4. 复合条件连接查询

上述各个连接查询中,WHERE 子句中只有一个条件,即连接谓词。WHERE 子句中可以有多个连接条件,称为复合条件连接。

【例 3.49】　查询 20100015 号抢修项目所使用的物资编号、物资名称、规格和使用数量。

```
SELECT Out_stock.mat_num,mat_name,speci,Out_stock.amount
FROM Stock INNER JOIN Out_stock ON Stock.mat_num = Out_stock.mat_num
WHERE prj_num = '20100015'
```

查询结果如图 3.25 所示。

mat_num	mat_name	speci	amount
m001	护套绝缘电线	BVV-120	2
m002	架空绝缘导线	10KV-150	1

图 3.25　例 3.49 查询结果

连接操作除了可以是两张表连接、一张表与其自身连接外,还可以是两张以上的表进行连接,通常称为多表连接。

【例 3.50】　查询使用了护套绝缘电线的所有抢修项目编号及名称。

本查询涉及 3 张表,完成该查询的 SQL 语句如下。

```
SELECT Out_stock.prj_num,prj_name
FROM   (Stock INNER JOIN Out_stock ON Stock.mat_num = Out_stock.mat_num)
       INNER JOIN Salvaging ON Salvaging.prj_num = Out_stock.prj_num
WHERE mat_name = '护套绝缘电线'
```

5. 自身连接查询

连接操作不仅可以在两张表之间进行,还可以同一张表与自己进行连接,这种连接称为表的自身连接。

【例 3.51】　查询同时使用了编号为 m001 和 m002 物资的抢修工程的工程号。

在 Out_stock 表的每行记录中,只有某一抢修工程使用的一种物资信息,若要得到一个抢修工程同时使用的两种物资信息,则需要将 Out_stock 表与其自身连接。为方便连接运算,这里为 Out_stock 表取两个别名,分别为 A 和 B,通过 A 表的抢修工程项目号与 B 表的抢修工程项目号连接,这样自身连接之后得到的一张大表中的每行记录就可以表示同一抢修工程使用的两种物资信息,然后再对物资编号进行条件查询。

完成该查询的 SQL 语句为

```
SELECT A.prj_num
FROM Out_stock A   INNER JOIN Out_stock B ON A.  prj_num = B.prj_num
WHERE A.mat_num = 'm001' AND B.mat_num = 'm002'
```

查询结果如图 3.26 所示。

prj_num
20100015

图 3.26　例 3.51 查询结果

扫一扫 视频讲解

3.3.3　嵌套查询

扫一扫 视频讲解

在 SQL 中，一个 SELECT…FROM…WHERE 语句称为一个查询块。将一个查询块嵌套在另一个查询块的 WHERE 子句或 HAVING 子句的条件中的查询称为嵌套查询。

```
SELECT prj_name                          -- 外层查询或父查询
FROM Salvaging
WHERE prj_num IN
          (SELECT prj_num                -- 内层查询或子查询
           FROM Out_stock
           WHERE mat_num = 'm003');
```

在本例中，下层查询块 SELECT prj_num FROM Out_stock WHERE mat_num＝'m003'是嵌套在上层查询块 SELECT prj_name FROM Salvaging WHERE prj_num IN 的 WHERE 条件中的。上层的查询块称为外层查询或父查询，下层查询块称为内层查询或子查询。

SQL 允许多层嵌套查询，即一个子查询中还可以嵌套其他子查询。需要特别指出的是，子查询的 SELECT 语句不能使用 ORDER BY 子句，ORDER BY 子句只能对最终查询结果排序。

嵌套查询使用户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。以层层嵌套的方式构造程序是 SQL 中“结构化”的含义所在。

1. 带谓词 IN 的嵌套查询

在嵌套查询中，子查询的结果往往是一个集合，所以谓词 IN 是嵌套查询中最经常使用的谓词。

【例 3.52】　查询与规格为 BVV-120 的护套绝缘电线在同一个仓库存放的物资名称、规格和数量。

先分步完成此查询，然后再构造嵌套查询。

（1）确定规格为 BVV-120 的护套绝缘电线的物资所存放仓库的名称。

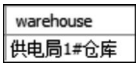
```
SELECT warehouse
FROM Stock
WHERE speci = 'BVV－120' AND mat_name = '护套绝缘电线';
```

查询结果如图 3.27 所示。

（2）查找所有存放在供电局 1＃仓库的物资。

```
SELECT mat_name, speci, amount
FROM Stock
WHERE warehouse  = '供电局 1＃仓库';
```

查询结果如图 3.28 所示。



warehouse
供电局1#仓库

图 3.27　例 3.52 查询结果(1)

mat_name	speci	amount
护套绝缘电线	BVV-120	220
架空绝缘导线	10KV-150	30

图 3.28　例 3.52 查询结果(2)

将步骤(1)查询嵌入步骤(2)查询的条件中,构造嵌套查询,SQL 语句如下。

```
SELECT mat_name, speci, amount
FROM Stock
WHERE warehouse IN
          (SELECT warehouse
           FROM Stock
           WHERE speci = 'BVV－120' AND mat_name = '护套绝缘电线');
```

本例中,子查询的查询条件不依赖于父查询,称为不相关子查询。不相关子查询是最简单的一类子查询,一种求解方法是由里向外处理,即先执行子查询,子查询的结果用于建立其父查询的查找条件。

本例中的查询也可以用自身连接来完成。

```
SELECT s1.mat_name, s1.speci, s1.amount
FROM Stock s1 INNER JOIN Stock s2 ON s1.warehouse = s2.warehouse
WHERE s2.speci = 'BVV－120' AND s2.mat_name = '护套绝缘电线';
```

可见,实现同一个查询可以有多种方法。当然,不同的方法,其执行效率可能会有所差别,甚至会相差很大。

【例 3.53】　查询“观澜站光缆抢修”工程项目所使用的物资编号和物资名称。

本查询涉及物资编号、物资名称和工程项目名称 3 个属性。物资编号和物资名称存放在 Stock 表中,工程项目名称存放在 Salvaging 表中,但 Stock 与 Salvaging 两张表之间没有直接联系,必须通过 Out_stock 表建立它们三者之间的联系,所以本查询实际上涉及 3 个关系。

```
SELECT mat_num, mat_name
FROM Stock                      -- 在 Stock 关系中取出 mat_num 和 mat_name
WHERE mat_num IN
        (SELECT mat_num
         FROM Out_stock         -- 在 Out_stock 关系中找出 20110003 号工程使用的物资编号
         WHERE prj_num IN
                (SELECT prj_num
                 FROM Salvaging      -- 在 Salvaging 关系中找出"观澜站光缆抢修"项目编号
                 WHERE prj_name = '观澜站光缆抢修'));
```

查询结果如图 3.29 所示。

本例同样可以用连接查询实现。

```
SELECT Stock.mat_num,mat_name
FROM Stock,Out_stock,Salvaging
WHERE Stock.mat_num = Out_stock.mat_num   AND
      Out_stock.prj_num = Salvaging.prj_num   AND
      prj_name = '观澜站光缆抢修';
```

mat_num	mat_name
m001	护套绝缘电线
m010	户外真空断路器

图 3.29　例 3.53 查询结果

还可以看到,当查询涉及多个关系时,用嵌套查询逐步求解,层次清楚,易于构造,具有结构化程序设计的优点。有些嵌套查询可以用连接运算代替,有些则不能。对于可以用连接运算代替嵌套查询的,到底采用哪种方法,用户可以根据自己的习惯确定。

2. 带比较运算符的子查询

带比较运算符的子查询是指父查询与子查询之间用比较运算符进行连接。当用户能确切知道内层查询返回的是单值时，可以使用>、<、=、>=、<=、!=或< >等比较运算符。

例如，在例 3.46 中，由于同一规格的护套绝缘电线只可能在一个仓库存放，也就是说内查询的结果是一个值，因此可以用=代替 IN，其 SQL 语句如下。

```
SELECT mat_name, speci, amount
FROM Stock
WHERE warehouse =
          ( SELECT warehouse
            FROM Stock
            WHERE speci = 'BVV - 120' AND mat_name = '护套绝缘电线');
```

【例 3.54】 查询出库存量超过该仓库物资平均库存量的物资编号、名称、规格及数量。

```
SELECT mat_num, mat_name,speci,amount
FROM Stock s1
WHERE amount >
          ( SELECT AVG(amount)
            FROM Stock s2
            WHERE s2.warehouse = s1.warehouse);
```

s1 是 Stock 表的别名，又称为元组变量，可以用来表示 Stock 表的一个元组。内层查询是求一个仓库所有物资的平均库存量，至于是哪一个仓库的平均库存量则要看参数 s1.warehouse 的值，而该值是与父查询相关的，因此这类查询称为相关子查询(Correlated Subquery)，整个查询语句称为相关嵌套查询(Correlated Nested Query)语句。

这个语句的一种可能的执行过程如下。

(1) 从外层查询中取出 Stock 表的一个元组 s1，将元组 s1 的 warehouse 值(供电局 1＃仓库)传给内层查询。

```
SELECT AVG(amount)
FROM Stock s2
WHERE s2.warehouse = '供电局 1＃仓库';
```

(2) 执行内层查询，得到值 125，用该值代替内层查询，得到外层查询。

```
SELECT mat_num,mat_name,speci,amount
FROM Stock s1
WHERE amount > 125 AND s1.warehouse = '供电局 1＃仓库';
```

(3) 执行这个查询，得到

```
(m001, 护套绝缘电线,BVV - 120, 220)
```

然后，外层查询取出下一个元组重复上述步骤，直到外层的 Stock 表中的元组全部处理完毕，查询结果如图 3.30 所示。

mat_num	mat_name	speci	amount
m001	护套绝缘电线	BVV-120	220
m004	护套绝缘电线	BVV-50	283
m008	护套绝缘电线	BVV-95	164
m009	交联聚乙烯绝缘电缆	YJV22-15KV	45

图 3.30 例 3.54 查询结果

求解相关子查询不能像求解不相关子查询那样，一次将子查询求解出来，然后求解父查询。相关子查询中内层查询由于与外层查询有关，因此必须反复求值。

【例 3.55】 查询其他仓库中比供电局 1＃仓库的某一物资库存量少的物资名称、规格和数量。

```
SELECT mat_name, speci, amount
FROM Stock
WHERE warehouse <> '供电局 1 # 仓库'
        AND amount < (SELECT MAX(amount)
                      FROM Stock
                      WHERE warehouse = '供电局 1 # 仓库');
```

查询结果如图 3.31 所示。

mat_name	speci	amount
护套绝缘电线	BVV-35	80
护套绝缘电线	BVV-70	130
护套绝缘电线	BVV-150	46
架空绝缘导线	10KV-120	85
护套绝缘电线	BVV-95	164
交联聚乙烯绝缘电缆	YJV22-15KV	45
户外真空断路器	ZW12-12	1

图 3.31　例 3.55 查询结果

3. 带 EXISTS 谓词的子查询

EXISTS 代表存在量词 ∃。带 EXISTS 谓词的子查询不返回任何数据，只产生逻辑真值 TRUE 或逻辑假值 FALSE。

【例 3.56】　查询所有使用了 m001 号物资的工程项目名称。

本查询涉及 Salvaging 和 Out_stock 表。我们可以在 Salvaging 表中依次取每个元组的 prj_num 值，用此值去检查 Out_stock 表。若 Out_stock 表中存在这样的元组，其 prj_num 值等于 Salvaging. prj_num 值，并且其 mat_num ='m001'，则取此 Salvaging. prj_name 送入结果关系。将此想法写成 SQL 语句：

```
SELECT prj_name
FROM Salvaging
WHERE EXISTS
       (SELECT *
        FROM Out_stock
        WHERE prj_num = Salvaging. prj_num AND mat_num = 'm001');
```

由 EXISTS 引出的子查询，其目标属性列表达式一般用 * 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。若内层子查询结果非空，则外层的 WHERE 子句条件为真(TRUE)，否则为假(FALSE)。

本例中子查询的查询条件依赖于外层父查询的某个属性值(Salvaging 表的 prj_num 值)，因此也是相关子查询(Correlated Subquery)。这个查询语句的处理过程是首先取外层查询 Salvaging 表中的第 1 个元组，根据它与内层子查询相关的属性值(prj_num 值)处理内层子查询，若 WHERE 子句返回值为真(TRUE)，则取此元组放入结果表；然后再取 Salvaging 表的下一个元组；重复这一过程，直至外层 Salvaging 表全部检查为止。

与 EXISTS 谓词相对应的是 NOT EXISTS 谓词。使用存在量词 NOT EXISTS 后，若内层子查询结果为空，则外层的 WHERE 子句返回真值，否则返回假值。

【例 3.57】　查询所有没有使用 m001 号物资的工程项目编号及名称。

```
SELECT prj_num,prj_name
FROM Salvaging
WHERE NOT EXISTS
      (SELECT *
       FROM Out_stock
       WHERE prj_num = Salvaging.prj_num AND mat_num = 'm001');
```

查询结果如图 3.32 所示。

prj_num	prj_name
20110006	朝阳围公变低压线抢修

图 3.32　例 3.57 查询结果

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、比较运算符的子

查询都能用带 EXISTS 谓词的子查询等价替换。

【例 3.58】 将例 3.46 改为带谓词 EXISTS 的查询，SQL 语句如下。

```
SELECT mat_name, speci, amount
FROM Stock s1
WHERE EXISTS
     (SELECT *
      FROM Stock s2
      WHERE S2.warehouse = S1.warehouse AND
            speci = 'BVV－120' AND mat_name = '护套绝缘电线');
```

由于带 EXISTS 谓词的相关子查询只关心内层查询是否有返回值，并不需要具体值，因此其效率并不一定低于不相关子查询，有时是高效的方法。

【例 3.59】 查询被所有抢修工程项目都使用了的物资名称及规格。

SQL 中没有全称量词(For All)，但是可以把带有全称量词的谓词转换为等价的带有存在量词的谓词，即

$$(\forall x)P = \neg(\exists x(\neg P))$$

这样，可将题目的意思转换为等价的用存在量词的形式：查询这样的物资，没有一个抢修工程没有使用过它。

```
SELECT mat_name, speci
FROM Stock
WHERE NOT EXISTS
(SELECT *
   FROM Salvaging
   WHERE NOT EXISTS
          ( SELECT *
             FROM Out_stock
             WHERE  mat_num = Stock.mat_num
               AND  prj_num = Salvaging.prj_num));
```

【例 3.60】 查询所用物资包含 20100016 号抢修工程所用物资的抢修工程项目号和项目名称。

本查询可用逻辑蕴含表达：查询抢修工程号为 x 的工程，对所有物资 y，只要 20100016 号工程项目使用了物资 y，则 x 也使用了 y。形式化表示如下。

用 p 表示谓词“20100016 号抢修工程使用了物资 y”。

用 q 表示谓词“抢修工程 x 使用了物资 y”。

则上述查询可表示为

$$(\forall y)p \rightarrow q$$

SQL 中没有蕴含(Implication)逻辑运算，但是可以利用谓词演算将一个逻辑蕴含的谓词等价转换为

$$p \rightarrow q = \neg p \vee q$$

该查询可转换为以下等价形式。

$$(\forall y)p \rightarrow q \equiv \neg(\exists y(\neg(p \rightarrow q))) \equiv \neg(\exists y(\neg(\neg p \vee q))) \equiv \neg(\exists y(p \wedge \neg q))$$

这样，可将题目的意思转换为等价的用存在量词的形式：不存在这样的物资 y，20100016 号抢修工程使用了物资 y，而抢修工程 x 没有使用物资 y。

```
SELECT prj_num,prj_name
FROM Salvaging
WHERE NOT EXISTS
```

```
        ( SELECT *
          FROM Out_stock A
          WHERE prj_num = '20100016'  AND
                NOT EXISTS
                ( SELECT *
                  FROM Out_stock B
                  WHERE B.mat_num = A.mat_num
                  AND B.prj_num = Salvaging.prj_num));
```

查询结果如图 3.33 所示。

prj_num	prj_name
20100016	沙河站2#公变出线电缆老化烧毁抢修
20110005	明珠立交电缆沟盖板破损抢修

图 3.33　例 3.60 查询结果

扫一扫

视频讲解

3.3.4　集合查询

SELECT 语句的查询结果是元组的集合，所以多个 SELECT 语句的结果可进行集合操作。UNION 运算符用来把两个或两个以上的 SELECT 语句的查询结果输出连接成一个单独的结果集。注意，参加集合操作的各结果表的列数必须相同，对应项的数据类型也必须相同。在执行集合查询时，查询结果的列标题为第 1 个查询语句的列标题，要对集合查询结果排序，也必须使用第 1 个查询语句中的列标题。

【例 3.61】　查询存放在供电局 1＃仓库且单价不大于 50 的物资，并按单价升序排列。

```
SELECT *
FROM Stock
WHERE warehouse = '供电局 1＃仓库'
UNION
SELECT *
FROM Stock
WHERE unit <= 50
ORDER BY unit;
```

本查询实际上是求存放在供电局 1＃仓库的物资与单价不大于 50 的物资的并集。在使用 UNION 将多个查询结果合并起来时，系统会自动去掉重复元组。

【例 3.62】　查询使用了编号为 m001 或 m002 的物资的抢修工程编号。

本查询实际上是求使用了编号为 m001 的物资的工程集合与使用了编号为 m002 的物资的工程集合的并集。

```
SELECT prj_num
FROM Out_stock
WHERE mat_num = 'm001'
UNION
SELECT prj_num
FROM Out_stock
WHERE mat_num = 'm002';
```

3.3.5　通过中间表查询

查询过程中，有时需要根据查询结果生成中间表，再通过中间表继续查询，得到需要的结果。

【例 3.63】 查询使用抢修物资数量前三的抢修工程编号。

```
SELECT prj_num
FROM  (SELECT prj_num,SUM(amount) AS sum_amount
       FROM Out_stock
       GROUP BY prj_num)  AS S
ORDER BY sum_amount  DESC
LIMIT 3;
```

该查询中，FROM 查询的是一张命名为 S 的中间表。查询结果如图 3.34 所示。

prj_num
20110004
20110005
20100016

图 3.34 例 3.63 查询结果

SQL Server 中该查询应该写为

```
SELECT top 3 prj_num
FROM  (SELECT prj_num,SUM(amount) AS sum_amount
       FROM Out_stock
        GROUP BY prj_num)  AS S
ORDER BY sum_amount  DESC;
```

【例 3.64】 查询每个抢修工程的编号、名称及使用的抢修物资总数量。

```
SELECT Salvaging.prj_num,prj_name,sum_amount
FROM Salvaging INNER JOIN
       (SELECT prj_num,SUM(amount) AS sum_amount
        FROM Out_stock
        GROUP BY prj_num) as S
ON Salvaging.prj_num = S.prj_num
```

扫一扫

视频讲解

3.4 数据更新

数据更新操作主要包括插入数据（INSERT）、修改数据（UPDATE）和删除数据（DELETE）。

3.4.1 插入数据

SQL 的 INSERT 语句可用于向数据表中插入数据，既可以插入单行，也可以插入多行，甚至可以插入子查询结果。

1. 插入单行元组

插入单行元组的 INSERT 语句的一般语法格式为

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

其功能是将新元组插入指定的表中。新元组的属性列 1 的值为常量 1，属性列 2 的值为常量 2，以此类推。INTO 子句中没有出现的属性列，新记录在这些列上将取空值。但用户必须注意的是，在定义表时说明 NOT NULL 的属性列不能取空值，否则会出错。

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

【例 3.65】　将新的配电物资(物资编号：m020；物资名称：架空绝缘导线；规格：10KV-100；仓库名称：供电局 1＃仓库；单价：12.8；库存数量：50)插入配电物资库存记录表(Stock)中。

```
INSERT
INTO Stock(mat_num,mat_name,speci,warehouse,unit,amount)
VALUES ('m020','架空绝缘导线','10KV－100','供电局 1＃仓库', 12.8,50);
```

本例中指出了新增加元组在哪些属性列上要赋值，属性列的顺序可以与 CREATE TABLE 中的顺序不一样。VALUES 子句对新元组的各属性列赋值，字符串要用单引号(英文符号)括起来。

【例 3.66】　将新的抢修工程(20110011，观澜站电缆接地抢修，2011-2-3 0:00:00，2011-2-5 12:00:00，1)插入抢修工程计划表(Salvaging)中。

```
INSERT
INTO Salvaging
VALUES ('20110011','观澜站电缆接地抢修','2011－2－3 0:00:00','2011－2－5
12:00:00',1);
```

本例中 INTO 子句只指出了表名，没有指出属性名，这就表示新元组要在表的所有属性列上都指定值，而且属性列的顺序与 CREATE TABLE 中的顺序相同。VALUES 子句对新元组的各属性列赋值，一定要注意值与属性列的顺序要一一对应。

2. 插入多行元组

插入多行元组的 INSERT 语句的一般语法格式为

```
INSERT
INTO <表名>[<属性列 1>[,<属性列 2>...]]
VALUES(<常量 1_1>[,<常量 1_2>]...),
      (<常量 2_1>[,<常量 2_2>]...)
      …
      (<常量 n_1>[,<常量 n_2>]...);
```

该语句基本用法与插入单行元组类似，但是允许将多条数据记录用逗号隔开，放在关键字 VALUES 的后面，插入数据表中。

【例 3.67】　将多行数据记录插入领料出库表(Out_stock)中。

```
INSERT
INTO Out_stock
VALUES ('20110006','m001',2,'2011－3－9','工程 4 部'),
       ('20110006','m002',3,'2011－3－9','工程 4 部');
```

3. 插入子查询结果

子查询不仅可以嵌套在 SELECT 语句中，用于构造父查询的条件，也可以嵌套在 INSERT 语句中，用于生成要批量插入的数据。

插入子查询结果的 INSERT 语句的语法格式为

```
INSERT
INTO <表名>[(<属性列 1>[,<属性列 2>...])
子查询;
```

【例 3.68】　对每个抢修工程项目，求其所用物资的总费用，并把结果存入数据库。

首先，在数据库中建立一张新表，其中一列存放抢修工程项目号，另一列存放相应的物

资总费用。

```
CREATE TABLE Prj_cost
(     prj_num char(8)   PRIMARY KEY,
      cost decimal(18, 2)
);
```

然后，对 Out_stock 和 Stock 表自然连接后按工程项目号 prj_num 分组，利用聚集函数求出其总费用，并将其插入新表中。

```
INSERT
INTO Prj_cost
    SELECT prj_num, SUM(Out_stock.amount * unit)
    FROM Out_stock,Stock
    WHERE Out_stock.mat_num = Stock.mat_num
    GROUP BY prj_num
```

3.4.2 修改数据

修改操作(UPDATE)语句的一般语法格式为

```
UPDATE <表名>
SET <列名 1> = <表达式 1> [,<列名 2> = <表达式 2>...]
[WHERE <条件>];
```

其功能是修改指定表中满足 WHERE 子句条件的元组。其中，SET 子句给出<表达式 i >的值取代<列名 i >相应的属性列的值。如果省略 WHERE 子句，则表示要修改表中所有元组。

1. 修改单个元组的值

【例 3.69】 将编号为 m020 的物资的单价修改为 44.5 元。

```
UPDATE Stock
SET unit = 44.5
WHERE mat_num = 'm020';
```

2. 修改多个元组的值

【例 3.70】 将所有物资的单价加 1。

```
UPDATE Stock
SET unit = unit + 1;
```

3. 带子查询的修改

子查询也可以嵌套在 UPDATE 语句中，用于构造修改的条件。

【例 3.71】 将供电局 1＃仓库的所有物资的领取数量置零。

由于物资所在仓库的信息在 Stock 表中，而物资的领取数量在 Out_stock 表中，因此可以将 SELECT 子查询作为 WHERE 子句的条件表达式。

```
UPDATE Out_stock
SET amount = 0
WHERE mat_num in
       ( SELECT mat_num
         FROM Stock
         WHERE warehouse = '供电局 1＃仓库');
```

该语句还可以写为

```
UPDATE Out_stock, Stock
SET Out_stock.amount = 0
WHERE Stock. mat_num = Out_stock. mat_num
AND warehouse = '供电局1#仓库';
```

3.4.3　删除数据

删除数据操作(DELETE)语句的一般语法格式为

```
DELETE
FROM <表名>
[WHERE <条件>];
```

DELETE 语句的功能是从指定表中删除满足 WHERE 子句条件的所有元组。如果省略 WHERE 子句，表示删除表中全部元组，但表的定义仍在数据字典中。也就是说，DELETE 语句删除的是表中的数据，而不是关于表的定义。

1. 删除单个元组的值

【例 3.72】　删除项目号为 20110001 的抢修工程领取的编号为 m001 的物资出库记录。

```
DELETE
FROM Out_stock
WHERE prj_num = '20110001' AND mat_num = 'm001';
```

2. 删除多个元组的值

【例 3.73】　删除所有抢修工程的领料出库记录。

```
DELETE
FROM Out_stock;
```

这条 DELETE 语句将删除 Out_stock 表的所有元组，使其成为空表。

3. 带子查询的删除

子查询同样也可以嵌套在 DELETE 语句中，用于构造执行删除操作的条件。

【例 3.74】　删除“观澜站光缆抢修”工程项目的所有领料出库记录。

```
DELETE
FROM Out_stock
WHERE prj_num in
        (SELECT prj_num
         FROM Salvaging
         WHERE prj_name = '观澜站光缆抢修');
```

值得注意的是，由于增、删、改操作每次只能对一张表进行操作，如果不注意关系之间的参照完整性和操作顺序，就会导致操作失败甚至发生数据库不一致的问题。在后续章节中将详细介绍参照完整性的检查和控制。

3.5　视图

扫一扫

视频讲解

视图是关系数据库系统提供给用户以多种角度观察数据库中数据的重要机制。

视图是从一张或几张基本表(或视图)导出的表，它与基本表不同，是一张虚表。数据库中只存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中，所以

基本表中的数据发生变化，从视图中查询出的数据也就随之改变了。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中自己感兴趣的数据及其变化。

视图一经定义，就可以和基本表一样被查询、删除，也可以在一个视图之上再定义新的视图，但对视图的更新（增加、删除、修改）操作则有一定的限制。

3.5.1 视图的建立与删除

1. 建立视图

SQL 用 CREATE VIEW 命令建立视图，一般语法格式为

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

其中，<子查询>可以是任意的 SELECT 语句，但通常不允许含有 ORDER BY 子句；WITH CHECK OPTION 表示用视图进行 UPDATE、INSERT 或 DELETE 操作时要保证更新、插入或删除的元组满足视图定义中的谓词条件（即子查询中的条件表达式）。

组成视图的属性列名要么全部省略，要么全部指定。如果视图定义中省略了属性列名，则隐含该视图由子查询中 SELECT 子句的目标列组成。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）某个目标列不是单纯的属性名，而是聚集函数或列表达式。

（2）多表连接导出的视图中有几个同名列作为该视图的属性名。

（3）需要在视图中为某个列启用新的更合适的名字。

【例 3.75】 建立供电局 1＃仓库所存放物资的视图。

```
CREATE VIEW s1_stock
AS
SELECT mat_num,mat_name,speci,amount,unit
FROM Stock
WHERE warehouse = '供电局 1＃仓库';
```

本例中省略了 s1_stock 视图的列名，则 s1_stock 视图中就隐含了子查询中 SELECT 子句的 5 个目标列。

注意：RDBMS 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。只是在对视图进行查询时，才按视图的定义从基本表中将数据查出。

```
SELECT * FROM s1_stock
```

执行上述对视图的查询后，可得到如图 3.35 的查询结果。

mat_num	mat_name	speci	amount	unit
m001	护套绝缘电线	BVV-120	220	89.80
m002	架空绝缘导线	10KV-150	30	17.00

图 3.35 例 3.75 查询结果

【例 3.76】 建立供电局 1＃仓库所存放物资的视图，并要求进行修改和插入操作时仍需保证该视图只有供电局 1＃仓库所存放的物资。

```
CREATE VIEW s2_stock
AS
SELECT mat_num,mat_name,speci,amount,unit
```

```
FROM Stock
WHERE warehouse = '供电局 1 # 仓库'
WITH CHECK OPTION;
```

由于在定义 s2_stock 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 会自动检查或加上 warehouse ='供电局 1 # 仓库'的条件。

若一个视图是从单张基本表中导出的，并且只是去掉了基本表的某些行和某些列，保留了主键，这类视图称为行列子集视图。例 3.75 和例 3.76 建立的两个视图就是行列子集视图。

【例 3.77】 建立包含抢修工程项目名称(prj_name)、出库物资名称(mat_name)、规格(speci)及领取数量(amount)的视图。

本视图由 3 张基本表的连接操作导出，SQL 语句如下。

```
CREATE VIEW s1_outstock
AS
SELECT prj_name,mat_name,speci,out_stock.amount
FROM Stock,Salvaging,Out_stock
WHERE Stock.mat_num = Out_stock.mat_num AND
    Salvaging.prj_num = Out_stock.prj_num;
```

视图不仅可以建立在一张或多张基本表上，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。

【例 3.78】 建立供电局 1 # 仓库所存放物资库存数量不少于 50 的视图。

```
CREATE VIEW s3_stock
AS
 SELECT mat_num,mat_name,speci,amount
 FROM s1_stock
 WHERE amount > = 50;
```

本例中的 s3_stock 视图就是建立在 s1_stock 视图之上的。

在建立视图时也可根据应用的需要，设置一些由基本数据经过各种计算派生出的属性列，由于这些派生属性在基本表中并不实际存在，因此也称为虚拟列。带虚拟列的视图也称为带表达式的视图。

【例 3.79】 建立一个体现抢修工程项目实际抢修天数的视图。

```
CREATE VIEW s1_salvaging(prj_name,start_date,end_date,days)
AS
SELECT prj_name, start_date, end_date, datediff(end_date,start_date)
FROM Salvaging;
```

注意：本例中由于 SELECT 子句的目标列中含有表达式，因此必须在 CREATE VIEW 的视图名后面明确说明视图的各个属性列名。

在创建视图时还可以用带有聚集函数和 GROUP BY 子句的查询定义视图，称为分组视图。

【例 3.80】 将仓库名称与其仓库内所存放物资的种类数定义为一个视图。

```
CREATE VIEW s4_stock(warehouse,counts)
AS
SELECT warehouse,COUNT(mat_num)
FROM Stock
GROUP BY warehouse;
```

由于 AS 子句中 SELECT 语句的物资种类目标列是通过作用聚集函数得到的，所以在 CREATE VIEW 中必须明确定义组成 s4_stock 视图的各个属性列名，s4_stock 是一个分组视图。

【例 3.81】 将所有已按期完成的抢修工程定义为一个视图。

```
CREATE VIEW s2_salvaging(prj_num,prj_name,start_date,end_date,prj_status)
AS
SELECT *
FROM Salvaging
WHERE prj_status = 1;
```

本例中 s2_salvaging 视图是由子查询 SELECT * 建立的，则说明 s2_salvaging 视图与基本表 Salvaging 的属性列一一对应。如果以后修改了基本表 Salvaging 的结构，则 Salvaging 表与 s2_salvaging 视图的映射关系就被破坏了，该视图就无法正确使用。为避免出现这种问题，最好在修改基本表之后删除由该基本表导出的视图，然后重建这个视图。

2. 删除视图

删除视图语句的一般语法格式为

```
DROP VIEW <视图名>;
```

视图删除后其定义将从数据字典中删除。但是由该视图导出的其他视图的定义仍在数据字典中，不过均已无法使用，需要使用 DROP VIEW 语句将其一一删除。

【例 3.82】 删除 s1_stock 视图。

```
DROP VIEW s1_stock;
```

由于从 s1_stock 视图还导出了 s3_stock 视图，虽然 s3_stock 视图的定义仍在数据字典中，但已无法使用，所以需要使用 DROP VIEW s3_stock 语句将其删除。

3.5.2 查询视图

建立视图后，用户就可以像查询基本表一样使用视图了。

【例 3.83】 在供电局 1＃仓库的物资视图 s1_stock 中找出单价小于 20 元的物资名称、规格和单价。

```
SELECT mat_name,speci,unit
FROM s1_stock
WHERE unit < 20;
```

DBMS 执行对视图的查询时，首先进行有效性检查，即检查所涉及的表、视图等是否存在。如果存在，则从数据字典中取出视图的定义，把定义中的子查询和用户的查询语句结合起来，转换为等价的对基本表的查询，然后再执行这个修正后的查询。这个转换过程称为视图消解（View Resolution）。

本例转换后的查询语句为

```
SELECT mat_name,speci,unit
FROM Stock
WHERE warehouse = '供电局 1＃仓库' AND unit < 20;
```

由此可见，对视图的查询实质上就是对基本表的查询，因此基本表的变化可以反映到视图上，视图就像是基本表的窗口一样，通过视图可以看到基本表动态的变化情况。

【例 3.84】　查询使用了供电局 1# 仓库物资的抢修工程项目号。

本查询涉及 s1_stock 视图和基本表 Out_stock，通过将二者连接完成用户请求。SQL 语句如下。

```
SELECT DISTINCT prj_num
FROM s1_stock INNER JOIN Out_stock ON s1_stock.mat_num = Out_stock.mat_num;
```

通常情况下，对视图的查询是直截了当的，但有时这种转换不能直接进行，因而查询会产生问题。

【例 3.85】　查询所存物资种类大于 2 的仓库名称。

```
SELECT warehouse
FROM s4_stock
WHERE counts > 2;
```

将此查询与 s4_stock 视图的定义结合后，转换得到查询语句：

```
SELECT warehouse
FROM Stock
WHERE COUNT(mat_num)> 2
GROUP BY warehouse;
```

而这条查询语句是不正确的，因为在 WHERE 子句中不允许使用聚集函数作为条件表达式。正确的查询语句应转换为

```
SELECT warehouse
FROM Stock
GROUP BY warehouse
HAVING COUNT(mat_num)> 2;
```

目前多数关系数据库系统对于行列子集视图的查询均能正确转换，但对于非行列子集视图的查询就不一定能正确转换了。所以，对视图进行查询时应尽量避免这类查询，最好直接对基本表进行查询。

3.5.3　更新视图

更新视图是指通过视图插入(INSERT)、删除(DELETE)和修改(UPDATE)数据。

由于视图实际上是不存储数据的虚表，因此对视图的更新最终要转换为对基本表的更新。与查询视图一样，对视图的更新也是通过视图消解转换为对基本表的更新。

为防止用户在通过视图对数据进行增、删、改时有意或无意地对不属于视图范围内的基本表数据进行操作，可在建立视图时加上 WITH CHECK OPTION 子句。这样在视图上作更新操作时，RDBMS 会自动检查视图定义中的条件，若不满足，则拒绝执行该操作。

【例 3.86】　将供电局 1# 仓库的物资视图 s1_stock 中编号为 m001 的物资的库存量修改为 100。

```
UPDATE s1_stock
SET amount = 100
WHERE mat_num = 'm001';
```

DBMS 自动转换为对基本表的更新语句如下。

```
UPDATE Stock
SET amount = 100
WHERE warehouse = '供电局 1# 仓库' AND mat_num = 'm001';
```

【例 3.87】 向供电局 1＃仓库的物资视图 s1_stock 中插入一个新的物资记录，其中物资编号为 m022，物资名称为“护套绝缘电线”，规格为 BVV-150，数量为 100，单价为 14.5。

```
INSERT
INTO s1_stock
VALUES('m022', '护套绝缘电线', 'BVV－150', 100, 14.5);
```

如图 3.36 所示，执行该语句后，发现 MySQL 中该条记录被成功插入基本表 Stock 中，只是 warehouse 属性列为 NULL，用 SELECT ＊ FROM s1_stock 语句是看不到刚插入的元组的。

```
INSERT
INTO s2_stock
VALUES('m023', '护套绝缘电线', 'BVV－150', 100,14.5);
```

mat_num	mat_name	speci	warehouse	amount	unit
m001	护套绝缘电线	BVV-120	供电局1#仓库	220	89.80
m002	架空绝缘导线	10KV-150	供电局1#仓库	30	17.00
m003	护套绝缘电线	BVV-35	供电局2#仓库	80	22.80
m004	护套绝缘电线	BVV-50	供电局2#仓库	283	32.00
m005	护套绝缘电线	BVV-70	供电局2#仓库	130	40.00
m006	护套绝缘电线	BVV-150	供电局3#仓库	46	85.00
m007	架空绝缘导线	10KV-120	供电局3#仓库	85	14.08
m008	护套绝缘电线	BVV-95	供电局3#仓库	164	88.00
m009	交联聚乙烯绝缘电缆	YJV22—15KV	供电局4#仓库	45	719.80
m010	户外真空断路器	ZW12-12	供电局4#仓库	1	13600.00
m022	护套绝缘电线	BVV-150	NULL	100	14.50

图 3.36 插入记录后 Stock 表中的数据

如果将这条记录插入供电局 1＃仓库的物资视图 s2_stock 中，将无法执行。这主要是由于在定义 s2_stock 视图时应用了 WITH CHECK OPTION 子句，其作用是限制 warehouse 的值必须是“供电局 1＃仓库”才允许由 s2_stock 视图插入，否则 DBMS 拒绝执行该插入操作。

【例 3.88】 删除供电局 1＃仓库的物资视图 s1_stock 中编号为 m001 的物资的记录。

```
DELETE
FROM s1_stock
WHERE mat_num = 'm001';
```

DBMS 自动转换为对基本表的删除语句如下。

```
DELETE
FROM Stock
WHERE warehouse = '供电局 1＃仓库' AND mat_num = 'm001';
```

在关系数据库中，并不是所有视图都可用于更新操作，因为有些视图的更新操作不能唯一有意义地转换为对应基本表的更新操作。目前，各个关系数据库系统一般都只允许对行列子集视图进行更新，而且各个系统对视图的更新还有更进一步的规定，由于各系统实现方法上的差异，这些规定也不尽相同。

例如，MySQL 中规定以下视图无法更新。

（1）若视图的字段来自聚集函数，则此视图不允许更新。

（2）若视图定义中含有 GROUP BY 子句，则此视图不允许更新。

（3）若视图定义中含有 DISTINCT 关键字，则此视图不允许更新。

(4) 一个不允许更新的视图上定义的视图也不允许更新。

应该指出的是,不可更新的视图与不允许更新的视图是两个不同的概念。前者指理论上已证明是不可更新的视图;后者指实际系统中不支持其更新,但它本身有可能是可更新的视图。

3.5.4 视图的作用

视图最终是定义在基本表上的,对视图的一切操作最终也要转换为对基本表的操作,而且对于非行列子集视图进行查询或更新时还有可能出现问题。既然如此,为什么还要定义视图呢?这是因为合理使用视图能够带来许多好处。

1. 视图能够简化用户的操作

视图机制可以让用户关注自己感兴趣的数据,如果这些数据不是直接来自基本表,则可以通过定义视图使数据库看起来结构简单、清晰,并且可以简化用户的数据查询操作。例如,那些经常使用的查询被定义为视图,从而使用户不必在每次对该数据执行操作时都指定所有查询条件。再者,如果用户视图是由多张基本表导出的,则视图机制也把表与表之间的连接操作对用户隐蔽了。也就是说,用户所做的是对一张虚表的简单查询,而这张虚表是怎样得来的,用户无须了解。

2. 视图使用户能以多种角度看待同一数据

视图机制能使不同的用户以多种角度看待同一数据,当许多要求不同的用户共享同一个数据库时,这种灵活性是非常重要的。

3. 视图为重构数据库提供了一定程度的逻辑独立性

数据的独立性分为两种,即物理独立性与逻辑独立性。数据的物理独立性是指用户和用户程序不依赖于数据库的物理结构;数据的逻辑独立性是指当数据库重新构造时,如增加新的关系或对原有关系增加新的字段等,用户和用户程序不会受影响。层次数据库和网状数据库一般能较好地支持数据的物理独立性,而对于逻辑独立性则不能完全支持。在数据库中,数据库的重构往往是不可避免的。重构数据库最常见的是将一张基本表"垂直"地分成多张基本表。例如,将配电物资库存记录表 Stock(mat_num, mat_name, speci, warehouse, amount, unit)分为 s1(mat_num, mat_name, speci, warehouse)和 s2(mat_num, amount, unit)两张表。这时 Stock 表为 s1 表和 s2 表自然连接的结果。如果建立一个 stock 视图:

```
CREATE VIEW stock(mat_num,mat_name,speci,warehouse,amount,unit)
AS
SELECT s1.mat_num, s1.mat_name, s1.speci, s1.warehouse, s2.amount, s2.unit
FROM s1, s2
WHERE s1.mat_num = s2. mat_num;
```

这样尽管数据库的逻辑结构改变了,但不必修改应用程序,因为新建立的视图定义了用户原来的关系,使用户的外模式保持不变,用户的应用程序通过视图仍然能够查找数据。

当然,视图只能在一定程度上提供数据的逻辑独立性。例如,由于对视图的更新是有条件的,因此应用程序中修改数据的语句可能仍会因基本表结构的改变而改变。

4. 视图能够对机密数据提供安全保护

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，这样视图机制就自动提供了对机密数据的安全保护功能。例如，Stock 表涉及 5 个仓库的物资记录，可以在其上定义 5 个视图，每个视图只包含一个仓库的物资记录，并且只允许每个仓库的管理员查询自己仓库的物资视图。

小结

数据库标准语言 SQL 分为数据定义、数据查询、数据更新和数据控制四大部分，本章系统、详尽地讲解了前 3 部分的主要内容，数据控制部分将在数据库安全性中介绍。视图是关系数据库系统中的重要概念，合理地使用视图有许多好处。

SQL 是关系数据库的工业标准，目前，大部分数据库管理系统都支持 SQL-92 标准，但至今尚没有一个数据库系统能完全支持 SQL-99 标准。

本章的所有示例全部在 MySQL 上运行通过。

扫一扫

自测题

习题 3

一、选择题

1. 关系代数中的 π 运算符对应 SELECT 语句中的(　　)子句。

A. SELECT　　B. FROM　　C. WHERE　　D. GROUP BY

2. 关系代数中的 σ 运算符对应 SELECT 语句中的(　　)子句。

A. SELECT　　B. FROM　　C. WHERE　　D. GROUP BY

3. SELECT 语句执行的结果是(　　)。

A. 数据项　　B. 元组　　C. 表　　D. 视图

4. 视图创建完毕后，数据字典中存放的是(　　)。

A. 查询语言　　B. 查询结果

C. 视图定义　　D. 所引用的基本表的定义

5. SQL 创建视图应使用(　　)语句。

A. CREATE　SCHEMEA　　B. CREATE　TABLE

C. CREATE　VIEW　　D. CREATE　DATABASE

6. 当两个子查询的结果(　　)时，可以执行集合操作。

A. 结构完全不一致　　B. 结构完全一致

C. 结构部分一致　　D. 主键一致

7. SELECT 语句中与 HAVING 子句同时使用的是(　　)子句。

A. ORDER BY　　B. WHERE

C. GROUP BY　　D. 无须配合

8. WHERE 子句的条件表达式中，可以匹配 0 个到多个字符的通配符是(　　)。

A. *　　B. %　　C. _　　D. ?

9. 与 WHERE G BETWEEN 60 AND 100 语句等价的语句是(　　)。

A. WHERE　G>60 AND G<100　　B. WHERE　G>=60 AND G<100

C. WHERE　G>60 AND G<=100　　D. WHERE　G>=60 AND G<=100

10. SQL 中，“DELETE　FROM　表名”语句表示(　　)。

A. 从基本表中删除所有元组　　B. 从基本表中删除所有属性

C. 从数据库中删除这个基本表　　D. 从基本表中删除重复元组

二、综合题

1. 用 SQL 语句创建第 2 章习题中的 4 张表(见表 2.8～表 2.11)：客户表(Customers)、代理人表(Agents)、产品表(Products)和订单表(Orders)。

2. 用 SQL 语句实现第 2 章综合题 3 中的 8 个查询。

3. 用 SQL 语句实现第 2 章综合题 4 中的 7 个查询。

4. 针对综合题 1 中的 4 张表，用 SQL 语句完成以下各项操作。

(1) 查询订货数量为 500～800 的订单情况。

(2) 查询产品名称中含“水”字的产品名称与单价。

(3) 查询每个月的订单数、总订货数量以及总金额，要求赋予别名，并按月份降序排列。

(4) 查询姓王且名字为两个字的客户在 1 月份的订单情况，并按订货数量降序排列。

(5) 查询上海客户中总订货数量超过 2000 的订货月份。

(6) 查询每个产品的产品编号、产品名称、总订货数量以及总金额。

(7) 查询没有通过北京代理商订购笔袋的客户编号与客户名称。

(8) 查询这样的订单号：订货数量大于 3 月份所有订单的订货总量。

(9) 向产品表中增加一个产品，名称为“粉笔”，编号为 P20，单价为 1.50 元，销售数量为 25000 支。

(10) 将所有单价大于 1.00 元的产品单价提高 10%。

(11) 将所有由上海代理商代理的笔袋的订货数量修改为 2000。

(12) 将由 A06 供给 C006 的产品 P01 改为由 A05 供应，请进行必要的修改。

(13) 从客户关系中删除 C006 记录，并从供应情况关系中删除相应的记录。

(14) 删除 3 月份订购尺子的所有订单情况。

(15) 为上海的客户创建一个代理情况视图，包括代理人姓名、产品名称及产品单价。

(16) 创建一个视图，要求包含单价大于 1.00 元的所有产品的产品名称、总订货数量以及总金额。