第3章　关系数据库标准语言SQL

结构化查询语言(structured query language,SQL)是关系数据库的标准语言。SQL包括数据查询,数据库模式创建,数据库数据的插入与修改,数据库安全性、完整性定义与控制等一系列功能。

本章详细介绍SQL最基本的功能,并进一步讲述关系数据库的基本概念。

3.1　SQL概述

SQL是1974年由Boyce和Chamberlin提出的。1975－1979年IBM公司San Jose Research Laboratory研制的关系数据库管理系统原型系统System R中实现了这种语言。由于它功能丰富,语言简洁,使用方法灵活,备受用户及计算机工业界欢迎,被众多计算机公司和软件公司所采用。后来经各公司不断修改、扩充和完善,SQL最终发展成为关系数据库的标准语言。

第一个SQL标准是1986年10月由美国国家标准局(American National Standard Institute,ANSI)公布的,所以也称该标准为SQL86。1987年国际标准化组织(International Organization for Standardization,ISO)也通过了这一标准。

随后ANSI不断修改和完善SQL标准,并于1989年第二次公布SQL标准(SQL89)。

1992年公布了SQL92标准(SQL2)。

1999年公布了SQL99标准(SQL3)。

2003年公布了SQL2003标准(SQL：2003)。

2008年公布了SQL2008标准(SQL：2008)。

2011年公布了SQL2011标准(SQL：2011)。

2016年公布了SQL2016标准(SQL：2016)。

2016年发布的SQL2016标准已经扩展到了12部分,陆续引入了XML类型、Window函数、TRUNCATE操作、时序数据及JSON类型等。

自SQL成为国际标准语言以后,各个数据库厂家纷纷推出各自支持的SQL软件或与SQL的接口软件。这就使得SQL可以作为数据库通用的数据存取语言和标准接口,使不同数据库系统之间的互操作有了共同的基础。此举意义十分重大。因此,有人把确立SQL为关系数据库标准语言及其后的发展称为一场革命。

但是,目前没有一个数据库系统能够支持SQL标准的所有概念和特性。大部分数据库系统能支持SQL92标准中的大部分功能以及SQL99、SQL2003、SQL2008中的部分新概念。同时,许多软件厂商对SQL基本命令集还进行了不同程度的扩充和修改,又会支持标准以外的一些功能特性。

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语言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 已成为关系数据库领域中一个主流语言。

本书不是介绍完整的 SQL，而是介绍 SQL 的基本概念和基本功能。因此，在使用具体系统时，一定要阅读各产品的用户手册。

3.1.1 SQL 的特点

SQL 之所以能够为用户和业界所接受，成为国际标准，是因为它是一个综合的、通用的、功能极强的、简洁易学的语言。SQL 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）功能于一体，充分体现了关系数据库语言的特点和优点。其主要特点如下。

1. 综合统一

数据库的主要功能是通过数据库支持的数据语言来实现的。

格式化模型（层次模型、网状模型）的数据语言一般都分为模式数据定义语言（data definition language，DDL）、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）以及数据操纵语言（data manipulation language，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。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，修改模式并编译后再重装数据库，因此很麻烦。

而 SQL 则集数据定义语言（DDL）、数据操纵语言（DML）、数据控制语言（data control language，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、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的要求，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，还可根据需要随时地、逐步地修改模式，并不影响数据库的运行，从而使系统具有良好的可扩充性。

另外，在关系模型中，实体和实体间的联系均用关系表示，这种数据结构的单一性带来了数据操作符的统一性，查询、插入、删除、更新等每种操作都只需一种操作符，从而克服了格式化系统由于信息表示方式的多样性带来的操作复杂性。

2. 高度非过程化

格式化数据模型的数据操纵语言是面向过程的语言，用其完成某项请求，必须指定存取路径。而用 SQL 进行数据操作，用户只需提出“做什么”，而不必指明“怎么做”，因此用户无须了解存取路径，存取路径的选择以及 SQL 语句的操作过程由系统自动完成。这不但大大减轻了用户负担，而且有利于提高数据独立性。

3. 面向集合的操作方式

格式化数据模型采用的是面向记录的操作方式，任何一个操作其对象都是一条记录。例如，查询所有平均成绩在 80 分以上的学生姓名，用户必须说明完成该请求的具体处理过程，即如何用循环结构按照某条路径一条一条地把满足条件的学生记录读出来。而 SQL 采用集合操作方式，不仅查询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. 以同一种语法结构提供两种使用方式

SQL 既是自含式语言，又是嵌入式语言。作为自含式语言，它能够独立地用于联机交互的使用方式，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。作为嵌入式语言，SQL 语句能够嵌入高级语言（例如，C、COBOL、FORTRAN、PL/1、C++、Java 等）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，但由于设计巧妙，语言十分简洁，完成数据定义、数据操纵、数据控制的核心功能只用了 9 个动词：SELECT、CREATE、DROP、ALTER、INSERT、UPDATE、DELETE、GRANT、REVOKE，如表 3-1 所示。SQL 语法简单，接近英语口语，因此容易学习，容易使用。

表 3-1　SQL 语言的 9 个动词

SQL 功能	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

3.1.2　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。

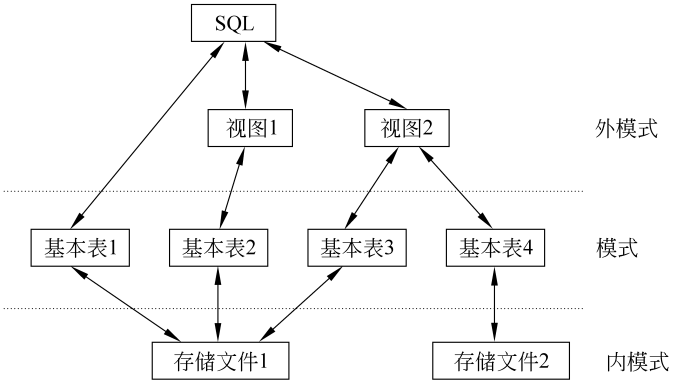

图 3-1　SQL 对关系数据库模式的支持

基本表是本身独立存在的表，在 SQL 中一个关系对应一个表。一些基本表对应一个存

储文件，一个表可以带若干索引，索引存放在存储文件中。

存储文件的逻辑结构组成了关系数据库的内模式。存储文件的物理文件结构是由数据库管理系统设计确定的。

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定义而不存放视图对应的数据，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。

用户可以用 SQL 对视图和基本表进行查询。在用户眼中，视图和基本表都是关系，而存储文件对用户是隐蔽的，即用户并不能直接看到存储文件，不能直接用 SQL 对存储文件进行操作。

从 3.2 节开始，将逐一介绍各 SQL 语句的功能和格式。为了突出基本概念和语句功能，略去了许多语法细节。而各种 DBMS 产品在实现标准 SQL 时也各有差别，一般都做了某种扩充。因此，读者具体使用某个 DBMS 产品时，一定要仔细参阅系统提供的有关手册。

3.2 数 据 定 义

关系数据库由模式、外模式和内模式组成，即关系数据库的基本对象是基本表、视图和索引。因此 SQL 的数据定义功能包括定义基本表、定义视图和定义索引，如表 3-2 所示。由于视图是基于基本表的虚表，索引是依附于基本表的，因此 SQL 通常不提供修改视图定义和修改索引定义的操作。用户如果想修改视图定义或索引定义，只能先将它们删除，然后再重建。不过有些关系数据库产品允许直接修改视图定义。

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
基本表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	
索引	CREATE INDEX	DROP INDEX	

本节只介绍如何定义基本表和索引，视图的概念及其定义方法将在 3.5 节讨论。

3.2.1 创建、修改与删除基本表

1. 创建基本表

建立数据库最重要的一步就是创建一些基本表。SQL 使用 CREATE TABLE 语句创建基本表，其一般格式为

```
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＞是所要创建的基本表的名字，它可以由一个或多个属性(列)组成。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。如果完整性约束条件涉及该表的多个属性列，则必须定义在表级上，否则既可以定义在列级，也可以定义在表级。

本章以学生-课程数据库为例讲解 SQL 语句。

为此，首先定义一个学生-课程(S-T)模式。学生-课程数据库中包括以下 3 个表。

- 学生表：Student(Sno，Sname，Ssex，Sage，Sdept)。
- 课程表：Course(Cno，Cname，Cpno，Ccredit)。
- 学生选课表：SC(Sno，Cno，Grade)。

例 3.1 创建一个学生表 Student，它由学号 Sno、姓名 Sname、性别 Ssex、年龄 Sage、所在系 Sdept 5 个属性组成，其中学号属性 Sno 是主码，学生姓名不能重名，即其值是唯一的。

```
CREATE TABLE Student
       (Sno    CHAR(9)    PRIMARY KEY,       /* Sno 是主码，列级完整性约束条件 */
        Sname  CHAR(20)   UNIQUE,            /* Sname 取唯一值，即学生姓名不能重名 */
        Ssex   CHAR(2),
        Sage   INT,
        Sdept  CHAR(20)
        );
```

系统执行上面的 CREATE TABLE 语句后，就在数据库中创建一个新的空的学生表 Student，并将有关学生表的定义及有关约束条件存放在数据字典中，如表 3-3 所示。

表 3-3 Student 表定义

Sno	Sname	Ssex	Sage	Sdept
↑ 字符型 长度为 9 主码	↑ 字符型 长度为 20 取值唯一	↑ 字符型 长度为 2	↑ 整型	↑ 字符型 长度为 20

例 3.2 创建一个课程表 Course。

```
CREATE TABLE Course
       (Cno    CHAR(4)   PRIMARY KEY,     /* 列级完整性约束条件，Cno 是主码 */
        Cname  CHAR(40)  NOT NULL,         /* 列级完整性约束条件，Cname 不能取空值 */
        Cpno   CHAR(4),                    /* Cpno 的含义是某一门课程 Cno 的直接先修课 */
        Ccredit SMALLINT,                  /* Cno 的学分 */
        FOREIGN KEY (Cpno) REFERENCES Course(Cno)
               /* 表级完整性约束条件，Cpno 是外码，被参照表是 Course，被参照列是 Cno */
       );
```

本例说明参照表和被参照表可以是同一个表。

例 3.3 创建学生选课表 SC。

```
CREATE TABLE SC
      (Sno CHAR(9),
       Cno CHAR(4),
       Grade SMALLINT,
       PRIMARY KEY (Sno,Cno),
            /* 主码由两个属性构成,必须作为表级完整性进行定义 */
       FOREIGN KEY (Sno) REFERENCES Student(Sno),
            /* 表级完整性约束条件,Sno 是外码,被参照表是 Student */
       FOREIGN KEY (Cno) REFERENCES Course(Cno)
            /* 表级完整性约束条件,Cno 是外码,被参照表是 Course */
);
```

定义表的各个属性时需要指明其数据类型及长度。常用的主要有

SMALLINT	整型,长度为 2 字节。
INTEGER 或 INT	整型,长度为 4 字节。
BIGINT	整型,长度为 8 字节。
NUMERIC(p,s)	十进制数,共 p 位,其中小数点后有 s 位。s=0 时可以省略。 p 和 s 的取值范围取决于不同数据库的实现。
DECIMAL(p,s)	与 NUMERIC(p,s)类似,但是数据值精度不受 p 和 s 的限制。
FLOAT(p)	浮点型,p 为小数点前后的总位数。
REAL	长度为 4 字节。
DOUBLE PRECISION	双精度浮点型,长度为 8 字节。
CHARTER(n)或 CHAR(n)	长度为 n 的定长字符串。
VARCHAR(n)	最大长度为 n 的变长字符串。
DATE	日期型,格式为年-月-日,YYYY-MM-DD。
TIMESTAMP	日期加时间,格式为 YYYY-MM-DD HH:MM:SS。
BLOB/CLOB	大对象类型,BLOB 中保存二进制文件,CLOB 中保存文本类型。
BOOLEN	布尔型,取值可以为 TRUE、FALSE、UNKNOW(NULL)。

注意：不同的数据库系统支持的数据类型会有细小的差别。使用时一定阅读产品手册。

2. 修改基本表

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基本表,包括增加新列、增加新的完整性约束条件、修改原有的列定义或删除已有的完整性约束条件等。SQL 用 ALTER TABLE 语句修改基本表,其一般格式为

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

[MODIFY <列名><数据类型>];

其中，<表名>指定需要修改的基本表，ADD 子句用于增加新列和新的完整性约束条件，DROP 子句用于删除指定的完整性约束条件，MODIFY 子句用于修改原有的列定义。

例 3.4 向 Student 表增加“入学时间”列，其数据类型为日期型。

```
ALTER TABLE Student ADD Scome DATE;
```

不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 将年龄的数据类型改为 SMALLINT。

```
ALTER TABLE Student MODIFY Sage SMALLINT;
```

修改原有的列定义有可能会破坏已有数据。

例 3.6 删除关于学生姓名必须取唯一值的约束。

```
ALTER TABLE Student DROP UNIQUE(Sname);
```

经过上述修改后，Student 表如表 3-4 所示。

表 3-4 修改后的 Student 表定义

Sno	Sname	Ssex	Sage	Sdept	Scome
↑ 字符型 长度为 9 主码	↑ 字符型 长度为 20	↑ 字符型 长度为 2	↑ 整型	↑ 字符型 长度为 20	↑ 日期型

SQL 没有提供删除属性列的语句，用户只能间接实现这一功能，即先将原表中要保留的列及其内容复制到一个新表中，然后删除原表，并将新表的名称重新命名为原表名。

3. 删除基本表

当某个基本表不再需要时，可以使用 SQL 语句 DROP TABLE 进行删除。其一般格式为

```
DROP TABLE <表名>;
```

例 3.7 删除 Student 表。

```
DROP TABLE  Student;
```

基本表定义一旦删除，表中的数据和在此表上建立的索引都将自动被删除，而建立在此表上的视图虽仍然保留，但已无法引用。因此执行删除基本表的操作一定要格外小心。

3.2.2 创建与删除索引

创建索引是加快表的查询速度的有效手段。当需要在一本书中查询某些信息时，往往

首先通过目录找到所需信息的对应页码，然后再从该页码中找出所要的信息，这种做法比直接翻阅书的内容速度要快。如果把数据库表比作一本书，那么表的索引就是这本书的目录，可见通过索引可以大大加快表的查询。

SQL 支持用户根据应用环境的需要，在基本表上创建一个或多个索引，以提供多种存取路径，加快查询速度。一般说来，创建与删除索引由数据库管理员(DBA)或表的属主(即创建表的人)负责完成。系统在存取数据时会自动选择合适的索引作为存取路径，用户不必也不能选择索引。

1. 创建索引

在 SQL 中，创建索引使用 CREATE INDEX 语句，其一般格式为

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

其中，＜表名＞指定要创建索引的基本表的名字。索引可以建在该表的一列或多列上，各列名之间用逗号分隔。每个＜列名＞后面还可以用＜次序＞指定索引值的排列次序，包括升序(ASC)和降序(DESC)两种，默认值为 ASC。

UNIQUE 表示此索引的每个索引值只对应唯一的数据记录。

CLUSTER 表示要创建的索引是聚簇索引。聚簇索引是指索引项的顺序与表中记录的物理顺序一致的索引组织。例如，执行下面的 CREATE INDEX 语句：

```
CREATE CLUSTER INDEX Stusname ON Student(Sname);
```

将会在 Student 表的 Sname(姓名)列上创建一个聚簇索引，而且 Student 表中的记录将按照 Sname 值的升序存放。

用户可以在最常查询的列上创建聚簇索引以提高查询效率。显然在一个基本表上最多只能创建一个聚簇索引。创建聚簇索引后，更新索引列数据时，往往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创建聚簇索引。

例 3.8 为学生-课程数据库中的 Student、Course、SC 3 个表创建索引。

Student 表按学号升序创建唯一索引，Course 表按课程号升序创建唯一索引，SC 表按学号升序和课程号降序创建唯一索引。

```
CREATE UNIQUE INDEX Stusno ON Student(Sno);
CREATE UNIQUE INDEX Coucno ON Course(Cno);
CREATE UNIQUE INDEX SCno ON SC(Sno ASC,Cno DESC);
```

2. 删除索引

索引一经创建，就由系统使用和维护它，无须用户干预。创建索引是为了减少查询操作的时间，但如果数据增加、删除、修改频繁，系统会花费许多时间来维护索引。这时，可以删除一些不必要的索引。

在 SQL 中，删除索引使用 DROP INDEX 语句，其一般格式为

```
DROP INDEX <索引名>;
```

例 3.9 删除 Student 表的 Stusname 索引。

```
DROP INDEX Stusname;
```

删除索引时，系统会同时从数据字典中删除有关该索引的描述。

3.3 查　　询

建立数据库的目的是存储数据、查询和处理分析数据。可以说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。其一般格式为

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

SELECT 语句的含义是，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，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 子句，则将结果按 GROUP BY 后的＜列名 1＞的值进行分组，该属性列值相等的元组为一组，每组产生结果表中的一条记录。通常会在每组中作用集函数。

如果 GROUP 子句带 HAVING 短语，则只有满足指定条件的组才予输出。

如果有 ORDER 子句，则结果表还要按＜列名 2＞的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。下面以学生-课程数据库为例说明 SELECT 语句的各种用法。

学生-课程数据库中包括以下 3 表。

(1) Student(Sno，Sname，Ssex，Sage，Sdept)，其中 Sno 为主码，用下画线表示。

学生表 Student 由学号(Sno)、姓名(Sname)、性别(Ssex)、年龄(Sage)、所在系(Sdept)5 个属性组成。

(2) Course(Cno，Cname，Cpno，Ccredit)，其中 Cno 为主码。

课程表 Course 由课程号(Cno)、课程名(Cname)、先修课号(Cpno)、学分(Ccredit)4 个属性组成。

先修课是指在选修某课程之前，需要先选修的某门课程。因为该课程需要用到先选修的某些知识。

(3) SC(Sno，Cno，Grade)，其中(Sno，Cno)为主码。

学生选课表 SC 由学号(Sno)、课程号(Cno)、成绩(Grade)3 个属性组成。

限于篇幅，这里列出了这 3 个表的极其少量的样本数据，如表 3-5～表 3-7 所示。

表 3-5　Student 表的部分数据

Sno	Sname	Ssex	Sage	Sdept
2020001	李勇	男	20	CS
2020002	刘晨	女	19	IS
2020003	王名	女	18	MA
2020004	张立	男	18	IS
⋮	⋮	⋮	⋮	⋮

在表 3-5 中，CS 代表计算机系，IS 代表信息系，MA 代表数学系。

表 3-6　Course 表的部分数据

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL 语言	6	4
8	DB_Design	1	4
9	DB_Programing	1	2
10	DB_DBMS Design	1	4
⋮	⋮	⋮	⋮

表 3-7　SC 表的部分数据

Sno	Cno	Grade
2020001	1	92
2020001	2	85
2020001	3	88
2020002	2	90
2020002	3	80
2020004	3	
⋮	⋮	⋮

3.3.1　单表查询

单表查询是指仅涉及一个数据库表的查询，例如选择一个表中的某些列值、选择一个表中的某些特定行等。单表查询是一种最简单的查询操作。

1. 选择表中的若干列

选择表中的全部列或部分列，这类运算又称为投影。其变化方式主要表现在 SELECT

子句的＜目标表达式＞上。

1）查询指定列

在很多情况下，用户只对表中的一部分属性列感兴趣，这时可以通过在 SELECT 子句的＜目标列表达式＞中指定要查询的属性，有选择地列出感兴趣的列。

例 3.10 查询全体学生的学号与姓名。

```
SELECT Sno, Sname
FROM Student;
```

＜目标列表达式＞中各个列的先后顺序可以与表中的顺序不一致。也就是说，用户在查询时可以根据应用的需要改变列的显示顺序。

例 3.11 查询全体学生的姓名、学号、所在系。

```
SELECT Sname, Sno, Sdept
FROM Student;
```

这时结果表中的列的顺序与基本表中不同，是按查询要求，先列出姓名属性，再列学号属性和所在系属性。

2）查询全部列

将表中的所有属性列都选出来，可以有两种方法。一种方法就是在 SELECT 关键字后面列出所有列名。如果列的显示顺序与其在基本表中的顺序相同，也可以简单地将＜目标列表达式＞指定为 *。

例 3.12 查询全体学生的详细记录。

```
SELECT *
FROM Student;
```

该 SELECT 语句实际上是无条件地把 Student 表的全部信息都查询出来，所以也称全表查询，这是最简单的一种查询。

3）查询经过计算的值

SELECT 子句的＜目标列表达式＞不仅可以是表中的属性列，也可以是有关表达式，即可以将查询出来的属性列经过一定的计算后列出结果。

例 3.13 查询全体学生的姓名及其出生年份。假设提交查询的年份是 2020 年。

```
SELECT Sname, 2020-Sage
FROM Student;
```

本例中，＜目标列表达式＞中第二项不是通常的列名，而是一个计算表达式，是用当前的年份（假设为 2020 年）减学生的年龄，这样，所得的即是学生的出生年份。输出的结果为

```
Sname   2020-Sage
-----   ----------
李勇      2000
刘晨      2001
王名      2002
```

```
张立      2002
```

＜目标列表达式＞不仅可以是算术表达式，还可以是字符串常量、函数等。

例 3.14 查询全体学生的姓名、出生年份和所在系，要求用小写字母表示所在系名。

```
SELECT Sname,'Year of Birth:', 2020-Sage, ISLOWER(Sdept)
FROM Student;
```

输出的结果为

```
Sname  'Year of Birth:'     2020-Sage  ISLOWER(Sdept)
-----  ---------------  --------  ---------------
李勇      Year of Birth:        2000           cs
刘晨      Year of Birth:        2001           is
王名      Year of Birth:        2002           ma
张立      Year of Birth:        2002           is
```

用户可以通过指定别名来改变查询结果的列标题，这对于含算术表达式、常量、函数名的目标列表达式尤为有用。例如，对于例 3.14，可以如下定义列别名

```
SELECT Sname NAME, 'Year of Birth:'BIRTH, 2020-Sage BIRTHDAY,ISLOWER(Sdept)
DEPARTMENT FROM Student;
```

输出的结果为

```
NAME        BIRTH              BIRTHDAY      DEPARTMENT
-----    -------------    --------    -----------
李勇        Year of Birth:       2000              cs
刘晨        Year of Birth:       2001              is
王名        Year of Birth:       2002              ma
张立        Year of Birth:       2002              is
```

2. 选择表中的若干元组

通过＜目标列表达式＞的各种变换，可以根据实际需要，从一个指定的表中选择所有元组的全部或部分列。如果只想选择部分元组的全部或部分列，则还需要指定 DISTINCT 短语或指定 WHERE 子句。

1）消除取值重复的行

两个本来并不完全相同的元组，投影到指定的某些列上后，可能变成完全相同的行了。

例 3.15 查询所有选修过课程的学生的学号。

```
SELECT Sno
FROM SC;
```

执行上面的 SELECT 语句后，输出的结果为

```
Sno
-------
2020001
```

```
2020001
2020001
2020002
2020002
2020004
```

该查询结果里包含了许多重复的行。如果想去掉结果表中重复的行，必须指定DISTINCT 短语：

```
SELECT DISTINCT Sno
FROM SC;
```

输出的结果为

```
Sno
-------
2020001
2020002
2020004
```

如果没有指定 DISTINCT 短语，则默认为 ALL，即要求结果表中保留取值重复的行。也就是说

```
SELECT Sno
FROM SC;
```

与

```
SELECT ALL Sno
FROM SC;
```

完全等价。

2）查询满足条件的元组

查询满足指定条件的元组可以通过 WHERE 子句实现。WHERE 子句常用的查询条件如表 3-8 所示。

表 3-8　常用的查询条件

查询条件	谓　词
比较大小	=，>，<，>=，<=，！=，<>，！>，！<，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	AND，OR

（1）比较大小。

用于进行比较的运算符一般包括

```
=                等于
```

>	大于
<	小于
>=	大于或等于
<=	小于或等于
!=或<>	不等于

有些产品中还包括

!>	不大于
!<	不小于

逻辑运算符 NOT 可与比较运算符同用,对条件求非。

例 3.16 查计算机系全体学生的名单。

```
SELECT Sname
FROM  Student
WHERE Sdept ='CS';
```

例 3.17 查所有年龄在 20 岁以下的学生姓名及其年龄。

```
SELECT Sname, Sage
FROM Student
WHERE Sage <20;
```

或

```
SELECT Sname, Sage
FROM Student
WHERE NOT Sage >=20;
```

例 3.18 查考试成绩有不及格的学生的学号。

```
SELECT DISTINCT Sno
FROM SC
WHERE Grade <60;
```

这里使用了 DISTINCT 短语,当一个学生有多门课程不及格,他的学号也只列一次。

(2) 确定范围。

谓词 BETWEEN … AND …和 NOT BETWEEN … AND …可以用来查询属性值在(或不在)指定范围内的元组,其中 BETWEEN 后是范围的下限(即低值),AND 后是范围的上限(即高值)。

例 3.19 查询年龄在 20～23 岁(包括 20 岁和 23 岁)的学生的姓名、所在系和年龄。

```
SELECT Sname, Sdept, Sage
FROM Student
WHERE Sage BETWEEN 20 AND 23;
```

与 BETWEEN…AND…相对的谓词是 NOT BETWEEN…AND…

例 3.20 查询年龄不在 20～23 岁的学生姓名、所在系和年龄。

```
SELECT Sname, Sdept, Sage
FROM Student
WHERE Sage NOT BETWEEN 20 AND 23;
```

(3) 确定集合。

谓词 IN 可以用来查找属性值属于指定集合的元组。

例 3.21 查信息系(IS)、数学系(MA)和计算机科学系(CS)的学生的姓名和性别。

```
SELECT Sname, Ssex
FROM Student
WHERE Sdept IN ('IS', 'MA', 'CS');
```

与 IN 相对的谓词是 NOT IN,用于查询属性值不属于指定集合的元组。

例 3.22 查既不是信息系、数学系,也不是计算机科学系的学生的姓名和性别。

```
SELECT Sname, Ssex
FROM Student
WHERE Sdept NOT IN ('IS','MA','CS');
```

(4) 字符匹配。

谓词 LIKE 可以用来进行字符串的匹配。其一般语法格式如下:

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

其含义是查询指定的属性列值与<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:

① %(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。acb,addgb,ab 等都满足该匹配串。

② _(下画线):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。acb,afb 等满足该匹配串。

例 3.23 查询学号为 2020001 的学生的详细情况。

```
SELECT *
FROM Student
WHERE Sno LIKE '2020001';
```

该语句实际上与下面的语句完全等价:

```
SELECT *
FROM Student
WHERE Sno ='2020001';
```

也就是说,如果 LIKE 后面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 3.24 查所有姓刘的学生的姓名、学号和性别。

```
SELECT Sname, Sno, Ssex
FROM Student
```

```
WHERE Sname LIKE '刘%';
```

例 3.25 查姓“欧阳”且全名为 3 个汉字的学生的姓名。

```
SELECT Sname
FROM Student
WHERE Sname LIKE '欧阳__';
```

注意,由于一个汉字占两个字符的位置,所以匹配串欧阳后面需要跟两个_。

例 3.26 查名字中第二个字为“阳”字的学生的姓名和学号。

```
SELECT Sname,Sno
FROM Student
WHERE Sname LIKE '__阳%';
```

例 3.27 查所有不姓刘的学生姓名。

```
SELECT Sname
FROM Student
WHERE Sname NOT LIKE '刘%';
```

如果用户要查询的匹配字符串本身就含有%或_,例如要查名字为 DB_Design 的课程的学分,应如何实现呢? 这时就要使用 ESCAPE '<换码字符>'短语对通配符进行转义了。

例 3.28 查 DB_Design 课程的课程号和学分。

```
SELECT Cno, Ccredit
FROM Course
WHERE Cname LIKE 'DB\_Design' 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,而是取其本身含义,被转义为普通的_字符。

例 3.29 查以“DB_”开头,且倒数第 3 个字符为 i 的课程的详细情况。

```
SELECT *
FROM Course
WHERE Cname LIKE 'DB\_%i__' ESCAPE '\';
```

注意这里的匹配字符串'DB_%i__'。第一个_前面有换码字符\,所以它被转义为普通的_字符。而%、第 2 个_和第 3 个_前面均没有换码字符\,所以它们仍作为通配符。其输出的结果为

```
Cno          Cname                  Ccredit
------    -------------        ----------
8          DB_Design               4
9          DB_Programing           2
10         DB_DBMS Design          4
```

(5) 空值。

谓词 IS NULL 和 IS NOT NULL 可用来查询空值和非空值。

例 3.30 某些学生选修某门课程后没有参加考试,所以有选课记录,但没有考试成绩,

下面来查一下缺少成绩的学生的学号和相应的课程号。

```
SELECT Sno, Cno
FROM SC
WHERE Grade IS NULL;
```

注意,这里的 IS 不能用等号(=) 代替。

例 3.31 查所有有成绩的记录的学生学号和课程号。

```
SELECT Sno, Cno
FROM SC
WHERE Grade IS NOT NULL;
```

(6) 多重条件。

逻辑运算符 AND 和 OR 可用来连接多个查询条件。如果这两个运算符同时出现在同一个 WHERE 条件子句中,则 AND 的优先级高于 OR,但用户可以用圆括号改变优先级。

例 3.32 查计算机科学系(CS)年龄在 20 岁以下的学生姓名。

```
SELECT Sname
FROM Student
WHERE Sdept='CS' AND Sage<20;
```

例 3.21 中查信息系(IS)、数学系(MA)和计算机科学系(CS)的学生的姓名和性别。其中的 IN 谓词实际上是多个 OR 运算符的缩写,因此,例 3.21 中的查询也可以用 OR 运算符写成如下等价形式:

```
SELECT Sname, Ssex
FROM Student
WHERE Sdept='IS' OR Sdept='MA' OR Sdept='CS';
```

3. 对查询结果排序

如果没有指定查询结果的显示顺序,DBMS 将按其最方便的顺序(通常是元组在表中的先后顺序)输出查询结果。用户也可以用 ORDER BY 子句指定按照一个或多个属性列的升序(ASC)或降序(DESC)重新排列查询结果,其中 ASC 为默认值。

例 3.33 查询选修了 3 号课程的学生的学号及其成绩,查询结果按分数的降序排列。

```
SELECT Sno, Grade
FROM SC
WHERE Cno='3'
ORDER BY Grade DESC;
```

输出结果为

```
  Sno        Grade
-------    -------
2020004                    /* 注意,有的数据库系统显示空值为 NULL,有的不显示任何值 */
2020001       88
```

```
2020002        80
```

前面已经提到,可能有些学生(例如表 3-7 中的 2020004)选修了 3 号课程后没有参加考试,即成绩列为空值。用 ORDER BY 子句对查询结果按成绩排序时,若按升序排,成绩为空值的元组将最后显示;若按降序排,成绩为空值的元组将最先显示。

例 3.34 查询全体学生情况,查询结果按所在系升序排列,对同一系中的学生按年龄降序排列。

```
SELECT *
FROM Student
ORDER BY Sdept, Sage DESC;
```

4. 使用集函数

为了进一步方便用户,增强检索功能,SQL 提供了许多集函数,主要包括

```
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>)       计算一列值的总和(此列必须是数值型)
AVG([DISTINCT|ALL] <列名>)       计算一列值的平均值(此列必须是数值型)
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
```

如果指定 DISTINCT 短语,则表示在计算时要取消指定列中的重复值。如果不指定 DISTINCT 短语或指定 ALL 短语(ALL 为默认值),则表示不取消重复值。

例 3.35 查询学生总人数。

```
SELECT COUNT(*)
FROM Student;
```

例 3.36 查询选修了课程的学生人数。

```
SELECT COUNT(DISTINCT Sno)
FROM SC;
```

学生每选修一门课,在 SC 中都有一条相应的记录,而一个学生一般都要选修多门课程,为避免重复计算学生人数,必须在 COUNT 函数中用 DISTINCT 短语。

例 3.37 计算选修了 1 号课程的学生的平均成绩。

```
SELECT AVG(Grade)
FROM SC
WHERE Cno='1';
```

例 3.38 查询学习 1 号课程的学生最高分数。

```
SELECT MAX(Grade)
FROM SC
WHERE Cno='1';
```

5. 对查询结果分组

GROUP BY 子句可以将查询结果表的各行按一列或多列取值相等的原则进行分组。

对查询结果分组的目的是细化集函数的作用对象。如果未对查询结果分组，集函数将作用于整个查询结果，即整个查询结果只有一个函数值，如上面的例 3.35～3.38。而集函数可以作用于每个组，即每组都有一个函数值。

例 3.39 查询各个课程号与相应的选课人数。

```
SELECT Cno, COUNT(Sno)
FROM SC
GROUP BY Cno;
```

该 SELECT 语句对 SC 表按 Cno 的取值进行分组，所有具有相同 Cno 值的元组为一组，然后对每组作用集函数 COUNT，以求得该组的学生人数。按照表 3-7，一种可能的查询结果为

```
  Cno          COUNT(Sno)
------    -----------
  1              42
  2              34
  3              44
  4              33
  5              48
  6              33
  7              30
  8              42
  9              40
  10             30
```

如果分组后还要求按一定的条件对这些组进行筛选，最终只输出满足指定条件的组，则可以使用 HAVING 短语指定筛选条件。

例 3.40 查询信息系(IS)选修了 3 门以上课程的学生的学号。为简单起见，这里假设 SC 表中有一列 Dept，它记录了学生所在系。

```
SELECT Sno
FROM  SC
WHERE Dept='IS';
GROUP BY Sno
HAVING COUNT(*)>3;
```

查选修课程超过 3 门的信息系学生的学号，首先需要通过 WHERE 子句从基本表中求出信息系的学生。然后求其中每个学生选修了几门课，为此需要用 GROUP BY 子句按 Sno 进行分组，再用集函数 COUNT 对每组计数。如果某一组的元组数目大于 3，则表示此学生选修的课程超过 3 门，应将他的学号选出。HAVING 短语指定选择组的条件，只有满足条件(即元组个数＞3)的组才会被选出。

WHERE 子句与 HAVING 短语的根本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。HAVING 短语作用于组,从中选择满足条件的组。

3.3.2 连接查询

一个数据库中的多个表之间一般都存在某种内在联系,它们共同提供有用的信息。前面的查询都是针对一个表进行的。若一个查询同时涉及两个以上的表,则称为连接查询。连接查询实际上是关系数据库中最主要的查询,主要包括等值连接查询、非等值连接查询、自身连接查询、外连接查询和复合条件连接查询等。

1. 等值与非等值连接查询

当用户的一个查询请求涉及数据库的多个表时,必须按照一定的条件把这些表连接在一起,以便能够共同提供用户需要的信息。用来连接两个表的条件称为连接条件或连接谓词,其一般格式为

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

其中比较运算符主要有=、>、<、>=、<=、!=

此外,连接谓词还可以使用下面形式:

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

当连接运算符为=时,称为等值连接。使用其他运算符称为非等值连接。

例 3.41 查询每个学生及其选修课程的情况。

学生情况存放在 Student 表中,学生选课情况存放在 SC 表中,所以本查询实际上同时涉及 Student 与 SC 两个表中的数据。这两个表之间的联系是通过两个表都具有的属性 Sno 实现的。要查询学生及其选修课程的情况,就必须将这两个表中学号相同的元组连接起来。这是一个等值连接。完成本查询的 SQL 语句为

```
SELECT Student.*, SC.*
FROM Student, SC
WHERE Student.Sno=SC.Sno;
```

本例的连接条件是 Student.Sno=SC.Sno,该连接是等值连接。

连接谓词中的列名称为连接字段。连接条件中的各连接字段类型必须是可比的,但不必是相同的。例如,可以都是字符型,或都是日期型;也可以一个是整型,另一个是实型,整型和实型都是数值型,因此是可比的。但若一个是字符型,另一个是整型就不允许了,因为它们是不可比的类型。

从概念上讲,DBMS 执行连接操作的过程是,首先在表 1 中找到第一个元组,然后从头开始顺序扫描或按索引扫描表 2,查询满足连接条件的元组,每找到一个元组,就将表 1 中的第一个元组与该元组拼接起来,形成结果表中的一个元组。表 2 全部扫描完毕后,再到表 1 中找第二个元组,然后从头开始顺序扫描或按索引扫描表 2,查询满足连接条件的元组,每找到一个元组,就将表 1 中的第二个元组与该元组拼接起来,形成结果表中的一个元组。