第3章

关系数据库语言——SQL

SQL(Structured Query Language,结构查询语言)是 1974 年由 Boyce 和 Chamberlin 提出的,是在关系数据库中使用最普遍的语言,包括数据定义、数据查询、数据操作和数据控制 4 种功能。本章主要介绍 SQL 中的基本操作。

SQL 介绍

3.1 SQL 简 介

SQL 是一个综合的、通用的、功能极强同时又简洁易学的语言。其主要特点如下。

(1) 综合统一。SQL 集数据定义语言(Data Definition Language, DDL)、数据操纵语言(Data Manipulation Language, 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,为数据库应用系统的开发提供了良好的环境。

(2) 高度非过程化。SQL 是高度非过程化语言,当进行数据操作时,只要提出"做什么",而无须指明"怎么做",用户不需要了解存取路径,存取路径的选择以及 SQL 语句的操作过程由系统自动完成。

(3) 面向集合的操作方式。SQL 采用集合操作方式,其操作对象和查找结果都是元组的集合。

(4) 以同一种语法结构提供两种使用方式。SQL 既可以作为独立的语言,采用联机交互的使用方式,用户在终端键盘上直接输入 SQL 命令对数据库进行操作,也可以采用嵌入式,嵌入高级语言(如 C#、Java)程序中,供程序员设计程序时使用。两种不同的使用方式中,SQL 的语法结构基本上是一致的,提供了极大的灵活性和方便性。

(5) 语言简洁,易学易用。SQL 功能极强,但十分简洁,完成核心功能只用了 9 个动词,如表 3.1 所示。

表 3.1 SQL 的动词

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

SQL 由以下几个部分组成。

(1) 数据定义语言。SQL DDL 提供定义关系模式和视图、删除关系和视图、修改关系模式的命令。

(2) 交互式数据操纵语言。SQL DML 提供查询、插入、删除和修改的命令。

(3) 完整性控制。SQL DDL 包括定义数据库中的数据必须满足的完整性约束条件的命令,对于破坏完整性约束条件的更新将被禁止。

(4) 安全性控制。SQL DDL 中包括说明对关系和视图的访问权限。

(5) 事务控制。SQL 提供定义事务开始和结束的命令。

(6) 嵌入式 SQL 和动态 SQL。用于嵌入某种通用的高级语言(C,C++,Java 等)中混合编程。其中,SQL 负责操纵数据库,高级语言负责控制程序流程。

3.2 数据定义

SQL 支持关系数据库的三级模式结构,如图 3.1 所示。其中,外模式对应于视图,模式对应于基本表,内模式对应于物理存储文件。

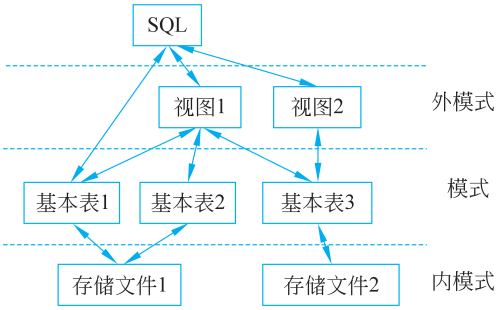

图 3.1 SQL 支持的数据库模式

SQL 的数据定义功能包括定义基本表、定义视图、定义索引。除此之外,SQL 还可以定义数据库、定义存储过程等。

3.2.1 基本表的定义

1. 基本表的创建

创建基本表的一般格式为

```
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>]);
```

说明:

(1) 定义中( )为语法结构,不可省略;< >括号内的内容为表定义的必选项;[ ]括号内的内容为表定义的可选项。

(2) 数据类型定义属性列的数据类型,不同数据库产品在数据类型的种类和关键词上

存在差异。

(3) 如果完整性约束条件涉及该表的多个属性列，则必须定义表级完整性约束条件，否则既可以定义列级完整性约束条件，也可以定义表级完整性约束条件。

2. 属性的数据类型

当用 SQL 语句定义表时，需要为表中的每一个属性定义数据类型及长度。PostgreSQL 的常用数据类型如表 3.2 所示。

表 3.2　PostgreSQL 的数据类型

数据类型	含　义
Smallint	短整型(2B)
Integer	整型(4B)
Bigint	大整型(8B)
Decimal	用户给定精度的浮点型
Real	单精度浮点型(4B)
Double precision	双精度浮点型(8B)
Char(n)	固定长度字符型，表示 n 个字符的固定长度字符串
Varchar(n)	可变长度的字符串，表示最多可以有 n 个字符的字符串
text	没有限制的可变长度的字符串
Bytea	1B 或 4B 的二进制串
Date	日期类型
Time	时间类型
Timestamp	时间戳类型
Money	货币类型
Boolean	布尔型

3. 完整性约束条件

常用的约束子句如下。

(1) PRIMARY KEY 约束。PRIMARY KEY 子句用来定义表的主码，一个表只能包含一个 PRIMARY KEY 约束，如果表的主码由多个属性构成，需要在表级的完整性约束上定义；如果主码由单个属性构成，则既可在列级定义也可在表级定义。

(2) NOT NULL 或 NULL 约束。用关键词 NOT NULL 或 NULL 说明指定属性的属性值是否允许为空值。空值是关系数据库的一个重要概念，表示不确定或没有意义，与空串或 0 等具有不同的含义。

(3) UNIQUE 约束。用 UNIQUE 约束定义属性的属性取值必须是唯一的。

(4) FOREIGN KEY…REFERENCES 约束。用 FOREIGN KEY…REFERENCES 约束来定义参照完整性，因为涉及两个表中的属性，该约束必须定义在表级完整性上。

【例 3.1】　学生成绩管理数据库有三个基本表：学生表 Student，课程表 Course，学生选

课表 SC。

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）

关系的主码用下画线表示。各个表中的数据如图 3.2 所示。在 PostgreSQL 数据库中创建学生成绩管理数据库。

学生表 Student

学号 Sno	学生姓名 Sname	性别 Ssex	年龄 Sage	所在系 Sdept
202005001	李洋	男	18	计算机系
202008019	王欣	女	18	数学系
202006056	张楠	女	18	计算机系
202006758	谢小平	男	19	电子系

课程表 Course

课程号 Cno	课程名称 Cname	先修课 Cpno	学分 Ccredit
1	数据结构	2	3.5
2	C 语言程序设计		3
3	数据库系统原理	1	3

学生选课表 SC

学号 Sno	课程号 Cno	成绩 Grade
202005001	1	80
202008019	2	92
202006056	3	90
202005001	3	80
202006056	1	70

图 3.2　学生成绩管理数据库的数据示例

```
CREATE TABLE Student(
    Sno CHAR(12) PRIMARY KEY,
    Sname VARCHAR(8),
    Ssex CHAR(2) NOT NULL CHECK(Ssex IN ('男', '女')), /*性别只能取'男'或'女'*/
    Sage INT,
    Sdept VARCHAR(20));
CREATE TABLE Course(
    Cno CHAR(4) PRIMARY KEY,
    Cname CHAR(40),
    Cpno CHAR(4) ,
```

```
    Ccredit SMALLINT,
    FOREIGN KEY (Cpno) REFERENCES Course(Cno) );
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 */
    );
```

3.2.2　基本表的修改

基本表在创建之后可以用 SQL 的 ALTER TABLE 命令修改表结构,该命令的一般格式为

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

说明:

(1) ADD 子句用于增加新的属性和该属性上的完整性约束。

(2) DROP 子句用于删除指定的完整性约束条件。

(3) ALTER COLUMN 用于修改原有属性的数据类型。

【例 3.2】　向 Student 表增加“生源地”列,其数据类型为字符型。

```
ALTER TABLE Student ADD Saddress VARCHAR(20);
```

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 3.3】　将年龄的数据类型由整型改为短整型。

```
ALTER TABLE Student ALTER COLUMN Sage TYPE SMALLINT;
```

【例 3.4】　增加课程表中对成绩的约束,使 Grade 为 0～100。

```
ALTER TABLE SC ADD CHECK (Grade BETWEEN 0 AND 100);
```

3.2.3　基本表的删除

删除表命令的一般格式为

```
DROP TABLE <表名> {RESTRICT | CASCADE}
```

说明：

(1) RESTRICT：删除表是有限制的。如果删除的基本表被其他的表引用或存在依赖该表的对象，则此表不能被删除。

(2) CASCADE：删除该表没有限制。在删除基本表的同时，相关的依赖对象一起删除。

3.3 数据查询

数据库查询是数据库应用的核心内容，SQL 提供查询语句 SELECT，格式如下。

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

SQL 查询子句的顺序为 SELECT、FROM、WHERE、GROUP BY、HAVING 和 ORDER BY。其中，SELECT、FROM 是必需的。具体说明如下。

(1) SELECT 子句用来列出查询结果的属性，其输出可以是列名、表达式、聚集函数等；DISTINCT 选项将去掉查询结果中重复的元组，ALL 则保留查询结果中的重复元组，默认情况下为 ALL。

(2) FROM 子句指定查询的输入，可以是基本表，也可以是视图，或者是嵌套的子查询。

(3) WHERE 子句用于对查询输出进行限定或是设置输入对象之间的连接条件等。

(4) GROUP BY 子句用于对查询结果进行分组，可以利用它进行分组统计。

(5) HAVING 子句是对分组结果进行的限定条件，必须搭配 GROUP BY 子句才能出现。

(6) ORDER BY 子句是对查询结果进行排序，并不改变数据本身的物理存储。

SELECT 语句可完成简单的查询，也可完成复杂的连接查询及嵌套子查询。

3.3.1 单表查询

单表查询是仅涉及一个表的查询。

1. 选择表中的若干列

(1) 查询输出指定列。

【例 3.5】 查询全体学生的学号与姓名。

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

【例 3.6】 查询全体学生的姓名、学号、所在系。

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

(2) 查询输出所有属性列。

在 SELECT 关键字后面列出所有列名或者将<目标列表达式>指定为 *。

【例 3.7】 查询全体学生的详细记录。

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

或

```
SELECT  *
FROM Student;
```

(3) 查询输出经过计算的属性列。

SELECT 子句的<目标列表达式>可以是算术表达式、函数或别名。

【例 3.8】 查询全体学生的姓名及出生年份。

```
SELECT Sname, 2021-Sage     /* 假定当年的年份为 2021 年 */
FROM Student;
```

【例 3.9】 查询全体学生的姓名、出生年份和所有系,要求用小写字母表示所有系名。

```
SELECT Sname,2004-Sage "Year of Birth;", lower (Sdept)
FROM Student;
```

(4) 更改列标题。

用户可以通过指定别名来更改查询结果的列标题,这可以使含有计算表达式、常量、函数名的目标列表达式等的输出更简洁。

```
SELECT Sno 学号, Sname 姓名, 2021-Sage 出生年月
FROM Student;
```

(5) 去掉重复行。

SELECT 子句可使用 ALL 或 DISTINCT 选项来显示符合查询条件的所有行或对符合条件的行中去掉重复行,默认为 ALL,使用 DISTINCT 则对重复出现的行只保留一行。

【例 3.10】 查询全部的系。

```
SELECT Sdept 系 FROM Student;
```

等价于:

```
SELECT ALL Sdept FROM Student;
```

执行结果为

系
计算机系
数学系
计算机系

使用 DISTINCT 去掉结果表中的重复行：

```
SELECT DISTINCT Sdept FROM STUDENT;
```

执行结果为

系
计算机系
数学系

2. 带条件的查询

WHERE 子句用于设置查询条件，过滤掉不需要的数据行，只有满足条件的行才出现在查询结果中。常用的查询条件如表 3.3 所示。

表 3.3 WHERE 常用的查询条件

查 询 方 式	运 算 符
比较	=、>、>=、<、<=、<>
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(逻辑运算)	NOT、AND、OR

(1) 比较运算符。

用于进行比较的运算符一般包括=(等于)、>(大于)、>=(大于或等于)、<(小于)、<=(小于或等于)、<>(不等于)。

【例 3.11】 查询计算机系全体学生的名单。

```
SELECT Sname
FROM Student
WHERE Sdept='计算机系';
```

【例 3.12】 查询所有年龄在 20 岁以下的学生姓名及其年龄。

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

(2) 确定范围。

【例 3.13】 查询年龄为 20～23 岁(包括 20 岁和 23 岁)的学生的姓名、系别和年龄。

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

【例 3.14】 查询年龄不为 20～23 岁的学生姓名、系别和年龄。

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

(3) 确定集合。

【例 3.15】 查询信息系(IS)、数学系(MA)和计算机系(CS)学生的姓名和性别。

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

【例 3.16】 查询既不是信息系(IS)、数学系(MA),也不是计算机系(CS)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,常用于模糊查找,它判断列值是否与指定的字符串格式相匹配。其语法格式如下。

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

【例 3.17】 查询学号为 200215121 的学生的详细情况。

```
SELECT *
FROM  Student
WHERE  Sno LIKE '200215121';
```

等价于:

```
SELECT  *
FROM  Student
WHERE Sno = ' 200215121 ';
```

匹配串也可以是含通配符%和_的字符串。%代表任意长度的字符,如 a%b 表示以 a 开头,以 b 结尾的任意长度的字符串;_代表单个字符,如 a_b 表示以 a 开头,以 b 结尾的长度

为 3 的任意字符串。

【例 3.18】 查询所有姓刘学生的姓名、学号和性别。

```
SELECT Sname, Sno, Ssex
FROM Student
WHERE Sname LIKE '刘%';
```

【例 3.19】 查询姓“欧阳”且全名为三个汉字的学生的姓名。

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

使用换码字符将通配符转义为普通字符。

【例 3.20】 查询 DB_Design 课程的课程号和学分,其中的_为普通字符。

```
SELECT Cno,Ccredit
FROM Course
WHERE Cname LIKE 'DB_Design' ESCAPE '_';
```

【例 3.21】 查询以"DB_"开头,且倒数第 3 个字符为 i 的课程信息。

```
SELECT   *
FROM  Course
WHERE  Cname LIKE  'DB_%i_ _' ESCAPE ' _';
```

(5) 多重条件查询。

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,用户可以用括号改变优先级。

【例 3.22】 查询计算机系年龄在 20 岁以下的所有女生的信息。

```
SELECT *
FROM Student
WHERE Sdept='计算机系' and Sage<20 and Ssex='女';
```

3. ORDER BY 子句

用户可以用 ORDER BY 子句指定按照一个或多个属性列的升序(ASC)或降序(DESC)排列输出查询结果,默认值为升序。当排序的属性包含空值时,空值在升序中最先显示,在降序中最后显示。

【例 3.23】 查询选修了 1 号课程的选修信息,按照成绩的降序排列。

```
SELECT *
FROM SC
WHERE Cno='1'
ORDER BY Grade DESC;
```


聚集查询

4. 聚集函数

SQL 提供的常用聚集函数如表 3.4 所示。

表 3.4　常用聚集函数

聚 集 函 数	含　义
COUNT([DISTINCT|ALL]*)	统计行数
COUNT([DISTINCT|ALL]<列名>)	统计给定属性列的值的个数
SUM([DISTINCT|ALL]<列名>)	计算给定属性列的总和
AVG([DISTINCT|ALL]<列名>)	计算给定属性列的平均值
MAX([DISTINCT|ALL]<列名>)	求给定属性列的最大值
MIN([DISTINCT|ALL]<列名>)	求给定属性列的最小值

说明：如果指定 DISTINCT 短语，则表示在计算时取消重复指定属性列中的重复值。默认情况为 ALL 值，即不取消对重复值的计算。

【例 3.24】　统计数学系的学生总人数。

```
SELECT COUNT(*) 数学系
FROM Student
WHERE Sdept='数学系';
```

【例 3.25】　计算选修了 5 号课程学生的平均成绩。

```
SELECT, AVG(Grade) '5'
FROM SC
WHERE Sno='5';
```

5. GROUP BY 子句

GROUP BY 分组子句是对查询结果按给定的属性或属性集进行分组。对查询结果分组的目的是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；进行分组后，聚集函数将作用于每一个组，即对每个组分别统计。

【例 3.26】　查询每门课程的选课人数。

要查询每门课程的选课人数，则需要对表 SC 按照课程号进行分组，课程号相同的为同一个组，对每个组分别计算行数。

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

查询结果为

Cno	COUNT(*)
1	2
2	1
3	2

【例 3.27】 计算每个学生的平均成绩。

要计算每个学生的平均成绩,则需要对 SC 中按学号进行分组,学号相同的为同一个组,对每个组计算平均成绩。

```
SELECT Sno, AVG(Grade)
FROM SC
GROUP BY Sno;
```

执行结果为

Sno	AVG(Grade)
202005001	75
202008019	92
202006056	85

如果对分组后统计的结果进行筛选,要求只输出满足条件的组,则用 HAVING 子句来给出限定条件。

【例 3.28】 输出学生人数小于 100 人的系及总人数。

```
SELECT Sdept, COUNT(*)
FROM Student
GROUP BY Sdept HAVING COUNT(*)<100;
```

说明:

(1) HAVING 子句和 WHERE 子句都是筛选条件,但 WHERE 作用于基表或视图,而 HAVING 子句作用于组。

(2) 聚集函数可以出现在 SELECT 子句、HAVING 子句和 ORDER BY 子句之后,不能出现在 WHERE 子句后。

(3) HAVING 子句必须紧随 GROUP BY 子句,不能单独出现。

(4) 带 GROUP BY 子句的 SELECT 输出只能是分组属性和聚集函数,不能输出与分组无关的属性列。


空值查询

6. 空值查询

(1) 空值 NULL 的含义。

在 SQL 中允许某些元组在某个属性列上取空值 NULL,用来表示“不知道”或“不存在”或“无意义”的值。如某个学生的年龄取值为 NULL,表示不知道该学生的年龄,该学生年龄的值是存在的,但不知道该值是什么;又如某个学生缺考某门课程,其成绩取值为空值,表示该学生本课程的成绩不存在,不是其他任何数值。

外连接运算可导致某些元组中产生空值,某些元组的插入也可能产生空值,但存在约束为 NOT NULL(非空值)或 UNIQUE(唯一值)的属性值不允许为空值。

(2) 空值的运算规则。

空值 NULL 作为一种特殊的属性值也可以参加运算,但它不是常量,不可以直接将

NULL 作为一个操作数，其运算规则如下。

① 空值 NULL 与任何值（包括另一个 NULL）进行算术运算，其结果仍然是空值 NULL。

② 当使用比较运算符（如＝或＜）时，比较空值 NULL 与任何值（包括另一个 NULL）时，其结果都为 UNKNOWN。值 UNKNOWN 是另外一个与 TRUE 和 FLASE 相同的布尔值，传统的二值逻辑运算扩展成了三值逻辑。三值逻辑运算的结果如表 3.5 所示。

表 3.5　三值逻辑真值表

X	*Y*	*X* AND *Y*	*X* OR *Y*	NOT *X*
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

（3）与空值有关的查询。

空值是一个很特殊的值，含有不确定性，需要做特殊的处理。判断一个属性的值是否为空值时，必须用 IS NULL 或 IS NOT NULL 来表示，不能直接有关系运算符。

【例 3.29】　查询缺考学生的学号和课程号。

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

查询涉及允许为空值 NULL 的属性时，在 WHERE 条件表达式中需要特别注意有关 NULL 的特殊的运算规则。

【例 3.30】　查询成绩不合格的学生学号。

```
SELECT Sno
FROM SC
WHERE Grade<60 or Grade is NULL;
```

查询返回 WHERE 条件为真的结果，而 NULL 与 60 比较运算的结果为 UNKNOWN，Grade＜60 的条件只能查找出参加了考试而不及格的学生，不能查询出未参加考试的学生学号，因此要找出所有不合格的学生需要合并上 Grade is NULL 的条件。

当 NULL 出现在集合中时，要注意其逻辑运算规则。

【例 3.31】　查询年龄不为空，也不为 18 岁和 19 岁的学生信息。

```
SELECT *
FROM Student
WHERE Sage not in (18,19, NULL);
```

无论数据库中的值如何,该查询都将返回空值。WHERE 条件等价于"Sage<>18 and Sage<>19 and Sage<>NULL",Sage<>NULL 运算的结果为 UNKNOWN,"Sage<>18 and Sage<>19 and Sage<>NULL"的结果只能是 FALSE 或 UNKNOWN,WHERE 条件只在为真时返回查询结果。

3.3.2 多表查询

若一个查询同时涉及两个及以上的表或视图,则称为多表查询或连接查询。连接查询包括等值连接查询与非等值连接查询、自然连接查询、自身连接查询、外连接查询和复合条件连接查询等。

1. 等值连接与非等值连接查询

连接查询的 WHERE 子句中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,比较运算符包括=、>、<、<>等。

连接谓词也可以使用下面的形式:

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

若比较运算符为=称为等值连接,其他比较运算符则称为非等值连接。

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,但名字不一定相同。若查询涉及多个表时,表中可能会有两个或两个以上的属性具有相同的名字,需要明确指定这些相同名字的属性如何被使用,SQL 通过在属性前加上关系名和一个点运算来解决,如 $R.A$ 表示关系 R 的属性 A。

【例 3.32】 查询每个学生及其选修课程的情况。

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

该查询的输出涉及学生表 Student 和选修表 SC 中的信息,因此要将这两个表通过共同的属性(学号 Sno)进行连接;属性列"学号"既出现在学生表 Student 中也出现在选修表 SC 中,因此要明确指明使用学生表中的学号还是选修表中的学号。

【例 3.33】 查询每个学生的学号、姓名、选修的课程名称及成绩。

```
SELECT Student.Sno, Sname, Cname, Grade
FROM Student, SC, Course
WHERE Student.Sno=SC.Sno AND SC.Cno=Course.Cno;
```

该查询的输出涉及学生表 Student、选修表 SC 和课程表 Course 中的信息，是多表连接的查询，要考虑多个表之间的连接关系。例如，学生表 Student 与选修表 SC 通过共同的属性“学号 Sno”进行连接，选修表 SC 与课程表 Course 通过共同的属性“课程号 Cno”进行连接，多表连接时所有的连接条件必须同时成立，用逻辑与 AND 表示。

2. 自然连接

自然连接是一种特殊的等值连接，是在等值连接中将重复的属性列去掉。

【例 3.34】　查询学生选修课程的情况，要求输出学生的姓名、学号、课程号和成绩。

```
SELECT Sname, Student.Sno, Cno, Grade
FROM Student,SC
WHERE Student.Sno=SC.Sno;
```

3. 自连接

SELECT 查询语句不但支持不同表之间的连接，而且支持任意表自身的连接。一个表与其自身进行连接称为表的自连接。当进行自连接查询时，需要在 FROM 子句中将关系 R 列出多次，对每一个 R 的出现定义一个别名来进行区分。在 SELECT 和 WHERE 子句中，通过别名加点符号来消除关系 R 的属性歧义。别名可以作为关系 R 的另外一个名字出现在需要的地方。

【例 3.35】　查询先修课相同的课程号。

```
SELECT C1.Cno, C2.Cno
FROM Course C1, Course C2
WHERE C1.Cpno=C2.Cpno and C1.Cno<>C2.Cno;
```

在 FROM 子句中为表 Course 声明了两个别名 C1 和 C2，在 SELECT 子句中输出表 Course 两行元组的 Cno 字段，在 WHERE 子句中由别名 C1 和 C2 引用来表示表 Course 二行元组的 Cpno 字段值相同；为了避免二行元组是相同的，在 WHERE 子句中加上了第二个条件。

4. 外连接

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。但假如想查询学生表中每个学生的基本情况及选课情况，若某个学生没有选课，则只输出该学生的基本情况，选课信息为空，这样的查询要求就需要使用外连接。外连接是通过在悬浮元组里填充空值来使之成为查询结果。

外连接的基本格式为

```
SELECT [ALL|DISTINCT] <目标列表达式> [, <目标列表达式>] …
FROM <表 1> LEFT|RIGHT|FULL [OUTER] JOIN <表 1> ON <连接条件>
[ WHERE <条件表达式> ]
```

外连接包括左外连接(LEFT)、右外连接(RIGHT)和全外连接(FULL)三种。左外连接列出左边关系中所有的元组，右边关系悬浮元组里填充空值；右外连接列出右边关系中所有的元组，左边关系悬浮元组里填充空值；全外连接则输出两个表的所有元组，左、右两边关系中悬浮元组里填充空值。

【例 3.36】 查询所有学生的选课情况,包括未选课的学生。

```
SELECT Student.Sno, Sname, Ssex, Sdept, Cno, Grade
FROM Student LEFT JOIN SC ON (Student.Sno=SC.Sno);
```

针对图 3.2 的数据库表其查询结果为

学号 Sno	学生姓名 Sname	性别 Ssex	年龄 Sage	所在系 Sdept	课程号 Cno	成绩 Grade
202005001	李洋	男	18	计算机系	1	80
202005001	李洋	男	18	计算机系	3	80
202008019	王欣	女	18	数学系	2	92
202006056	张楠	女	18	计算机系	1	70
202006056	张楠	女	18	计算机系	3	90
202006758	谢小平	男	19	电子系	NULL	NULL

5. 复合条件连接

一条 SQL 语句可以同时完成选择和连接查询,这时 WHERE 子句由连接谓词和选择谓词组成复合条件。

【例 3.37】 查询选修了 2 号课程且成绩在 90 分以上的学生的学号、姓名和成绩。

```
SELECT Student.Sno, Sname, Grade
FROM Student, SC
Where Student.Sno=SC.Sno and Cno='2' and Grade>90;
```

选择谓词 Cno='2'和 Grade>90 从选修表 SC 中找出满足条件的元组形成中间结果,再通过连接谓词 Student.Sno=SC.Sno 将学生表 Student 和中间结果连接起来输出满足条件的结果。

3.3.3 嵌套子查询

在 SQL 中,一个查询可以通过不同的方式被用来计算另一个查询。当某个查询是另一个查询的一部分时,称之为子查询。集合查询的并、交、差就是通过子查询来完成的。

子查询可能返回单个常量,这个常量能在 WHERE 子句中与另一个常量进行比较;子查询也可能返回关系,该关系可以在 WHERE 中使用,也可以出现在 FROM 子句中。子查询以层层嵌套的方式来构造程序正是 SQL 中结构化的含义所在。

需要特别指出的是,子查询语句中不允许使用 ORDER BY 子句,ORDER BY 子句只能对最终查询结果排序。


WHERE 子句嵌套子查询

1. WHERE 子句嵌套子查询

(1) IN 子查询。

带有 IN 谓词的子查询是指父查询与子查询之间用 IN 进行连接,用于判断父查询的某个属性的值是否在子查询的结果中。IN 表示某元素属于某个集合,NOT IN 则表示某元素

不属于某个集合。谓词 IN 是嵌套查询中最经常使用的谓词。

【例 3.38】　查询选修了 3 号课程的学生学号和姓名。

```
SELECT Sno, Sname
FROM Student
WHERE Sno IN (SELECT Sno
              FROM SC
              WHERE Cno='3');
```

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。

此查询也可以用连接查询完成：

```
SELECT Student.Sno, Sname
FROM Student, SC
WHERE Student.Sno=SC.Sno and Cno='3';
```

可见实现同一个查询可以用很多种方法，不同的方法其执行效率可能会存在差别。

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特点。

【例 3.39】　查询选修了课程名称为“操作系统”的学生学号和姓名。

```
SELECT Sno, Sname
FROM Student
WHERE Sno IN (SELECT Sno
              FROM SC
              WHERE Cno IN (SELECT Cno
                            FROM Course
                            WHERE Cname='操作系统'));
```

本查询也可以用连接查询替代：

```
SELECT Student.Sno, Sname
FROM Student, SC,Course
WHERE Student.Sno=SC.Sno and SC.Cno=Course.Cno and Cname='操作系统';
```

但有些嵌套子查询不能由连接查询替代，如例 3.40。

【例 3.40】　查询没有选修 3 号课程的学生学号和姓名。

```
SELECT Sno, Sname
FROM Student
WHERE Sno NOT IN (SELECT Sno
                  FROM SC
                  WHERE Cno='3');
```

(2) 带比较运算符的子查询。

带有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

确切知道子查询结果返回的是单个值时,可以用=、>、<、<>等比较运算符。

【例 3.41】 查询与“刘霞”在同一个系的学生的学号和姓名。

```
SELECT Sno,Sname
FROM Student
WHERE Sdept=(SELECT Sdept
FROM Student
WHERE Sname='刘霞');
```

【例 3.42】 查询“计算机系”超出全校学生平均年龄的学生学号和姓名。

```
SELECT Sno,Sname
FROM Student
WHERE Sdept='计算机系' and Sage > (SELECT AVG(Sage)
FROM Student);
```

(3) 带 ANY 或 ALL 谓词的子查询。

在带比较运算符的子查询中,当子查询返回多值时,要与 ANY 或 ALL 谓词修饰符配合使用。

ANY:表示任意一个值,在进行运算时,只要子查询中有一行能使结果为无休止,则结果就为真。

ALL:表示所有值,在进行比较运算时,子查询中的所有行都使结果为真时,结果才为真。其具体语义如下。

>ANY 大于子查询结果中的某个值
>ALL 大于子查询结果中的所有值
<ANY 小于子查询结果中的某个值
<ALL 小于子查询结果中的所有值
>=ANY 大于或等于子查询结果中的某个值
>=ALL 大于或等于子查询结果中的所有值
<=ANY 小于或等于子查询结果中的某个值
<=ALL 小于或等于子查询结果中的所有值
=ANY 等于子查询结果中的某个值
<>ANY 不等于子查询结果中的某个值
<>ALL 不等于子查询结果中的任何一个值

【例 3.43】 查询数学系比计算机系任意一个学生年龄都大的学生姓名和年龄。

```
SELECT Sname, Sage
FROM  Student
WHERE Sdept='数学系' and Sage>ALL (SELECT Sage
                                   FROM Student
                                   WHERE Sdept='计算机系');
```

事实上,用聚集函数实现子查询通常比直接用 ANY 或 ALL 查询效率要高。ANY、ALL 与聚集函数的对应有关系如表 3.6 所示。

表 3.6　ANY、ALL 谓词与聚集函数的等价转换关系

	=	<>	<	<=	>	>=
ANY	IN	--	<MAX	<=MAX	>MIN	>=MIN
ALL	--	NOT IN	<MIN	<=MIN	>MAX	>=MAX

(4) 带 EXIST 谓词的子查询。

EXIST 代表存在量词∃。带有 EXIST 谓词的子查询不返回任何实际数据，只产生逻辑真值“TRUE”或逻辑假值“FALSE”。因此，带 EXIST 谓词的子查询中，其目标列表达式通常用“*”，因为带 EXIST 的子查询只返回“TRUE”或“FALSE”，给出列名无实际意义。

使用 EXIST，若子查询结果为非空，则外层 WHERE 子句返回真值，否则返回假值。

【例 3.44】　查询选修了 3 号课程的学号和姓名。

```
SELECT Sno, Sname
FROM Student
WHERE EXISTS (SELECT *
              FROM SC, Student
              WHERE SC.Sno=Student.Sno and Cno='3' );
```

该子查询与前面 IN 子查询不同，其子查询的条件依赖于父查询的某个属性值，这类查询称为相关子查询。相关子查询的处理过程是先取父查询表中的第 1 行元组到子查询，若子查询结果为非空，则 WHERE 返回为真，则取父查询的当前元组放入结果表；再取父查询表的下一行元组，重复此过程直到父查询表全部访问完。

与 EXIST 谓词相对应的是 NOT EXIST。使用 NOT EXIST，若子查询结果为空，则外层 WHERE 子句返回真值，否则返回假值。SQL 没有提供表示全称量词∀的谓词。

【例 3.45】　查询选修了全部课程的学生姓名。

在 Student 表中查找学生，要求这个学生选修了全部课程。换句话说，即在 *S* 表中查找这样的学生：在 *C* 表中不存在一门课程这个学生没有选。按照此语义，可写出 SQL 查询语句为

```
SELECT Sname
FROM Student
WHERE NOT EXISTS (SELECT *
                  FROM C
                  WHERE NOT EXISTS (SELECT *
                                    FROM SC
                                    WHERE SC.Sno=Student.Sno and SC.Cno=C.Cno));
```

【例 3.46】　查询所学课程包括学号 3 的学生所学课程的学生学号。

在 SC 表中查找一个学生，对于 3 号学生所学的每一门课程，该学生都学了。将其改为双重否定形式是：不存在 3 号学生所选修的课程不被某个学生选修，这个学生就是要查找的学生。

```
SELECT DISTINCT Sno
FROM SC X
```

```
WHERE NOT EXISTS (SELECT *
                  FROM SC Y
                 WHERE Sno='3' and NOT EXISTS (SELECT *
                                               FROM SC Z
                                              WHERE  Y.Cno=Z.Cno AND X.Sno=Z.Sno));
```


FROM 子句嵌套子查询

2. FROM 子句嵌套子查询

子查询的另一个作用是在 FROM 子句中当关系使用。在 FROM 列表中,除了可以使用存储关系以外,还可以使用括起来的子查询。由于这个子查询的结果没有名字,必须给它取一个别名,然后就可以像引用 FROM 子句中的关系一样引用子查询。

【例 3.47】 查询输出计算机系平均成绩在 85 分以上的学生学号、姓名和平均成绩。

```
SELECT Student.Sno, Sname, AVGGD
FROM Student, (SELECT Sno, AVG(Grade) AVGGD FROM SC GROUP BY Sno HAVING AVG(Grade)
>85) TEMP
WHERE Student.Sno=TEMP.Sno AND Sdept='计算机系';
```

第一个 FROM 子句嵌套了一个子查询,该子查询找出了平均成绩在 85 分以上的学生学号和平均成绩,用别名 TEMP 表示。WHERE 子句则对学生表 Student 进行筛选后与子查询进行连接。


集合查询

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 语句的结果可以进行集合操作。集合操作主要包括并操作 UNION、交操作 INTERSECT 和差操作 EXCEPT。参加集合操作各个 SELECT 语句的查询结果的列数必须相同,对应的数据类型也必须相同。

【例 3.48】 查询计算机系和数学系的学生信息。

```
SELECT *
FROM Student
WHERE Sdept='计算机系'
UNION
SELECT *
FROM Student
WHERE Sdept='数学系';
```

该查询也可用逻辑运算 OR 来实现,其 SQL 语句为

```
SELECT *
FROM Student
WHERE Sdept='计算机系' OR Sdept='数学系';
```

【例 3.49】 查询计算机系年龄小于 20 岁的学生信息。

```
SELECT *
FROM Student
WHERE Sdept='计算机系'
```