第3章 chapter 3

关系数据库标准语言SQL

本章要点

学习、掌握与灵活应用国际标准数据库语言SQL是本章的要求。SQL语言的学习从数据定义(DDL)、数据查询(QUERY)、数据更新(DML)、视图(VIEW)等方面逐步展开,而嵌入式SQL是SQL的初步应用内容。SQL数据查询是本章学习的重点。


思政材料

3.1 SQL语言的基本概念与特点

SQL的全称是结构化查询语言(Structured Query Language),它是国际标准数据库语言,如今,无论是Oracle、SQL Server、MySQL、Sybase、Informix、OceanBase、TiDB、openGauss这样的大型数据库管理系统,还是Access、Visual Foxpro这样的PC上常用的微、小型数据库管理系统,都支持SQL语言。学习本章后,读者应该了解SQL语言的特点,掌握SQL语言的四大功能及其使用方法,重点掌握SQL数据查询功能及其使用方法。

3.1.1 语言的发展及标准化

在20世纪70年代初,E.F.Codd首先提出了关系模型。70年代中期,IBM公司在研制SYSTEM R关系数据库管理系统中研制了SQL语言,最早的SQL语言(称为SEQUEL2)是在1976年11月的IBM Journal of R&D上公布的。

1979年,ORACLE公司首先提供商用的SQL,IBM公司在DB2和SQL/DS数据库系统中也实现了SQL。

1986年10月,美国ANSI采用SQL作为关系数据库管理系统的标准语言(ANSI X3.135—1986),后为国际标准化组织(ISO)采纳为国际标准。

1989年,美国ANSI采纳在ANSI X3.135—1989报告中定义的关系数据库管理系统的SQL标准语言,称为ANSI SQL 89。

1992年,ISO又推出了SQL 92标准,也称为SQL 2,是最重要的一个版本,引入了标准的分级概念。

1999年,ISO/IEC(International Electrotechnical Commission)推出了SQL 1999

(SQL3)。这是变动最大的一个版本,改变了标准符合程度的定义,增加了面向对象特性、正则表达式、存储过程、Java 等支持。

2003 年,ISO/IEC 推出了 SQL 2003,引入了 XML、Window 函数等。

2008 年,SQL 2008 标准发布。这个版本增加了 TRUNCATE TABLE 语句、INSTEAD OF 触发器以及 FETCH 子句等功能。

2011 年,ISO/IEC 发布了 SQL 2011 标准。这个版本主要增加了对时态数据库(temporal database)的支持。

2016 年,SQL 2016 标准发布。SQL 2016 引入了新的 JSON 函数和操作符,增加了行模式识别(RPR)和多态表函数(PTF),具有处理复杂事件的强大功能。

2019 年,ISO/IEC 发布了 SQL 2019 标准。这个版本引进了多维数组(MDA)。

2023 年,ISO 发布了 SQL 2023 标准。这个版本包含 11 个部分,对 SQL 语言进行了全面的增强和扩展。除了增强 SQL 语言和 JSON 相关功能之外,SQL 2023 最大的变化是在 SQL 中直接提供图形查询语言(GQL)功能。

SQL 是一种介于关系代数与关系演算之间的语言,其功能包括查询、操纵、定义和控制 4 方面,是一个通用的、功能极强的关系数据库语言,目前已成为关系数据库的标准语言,广泛应用于各种数据库。

3.1.2 SQL 语言的基本概念

SQL 语言支持关系数据库系统三级模式结构,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。

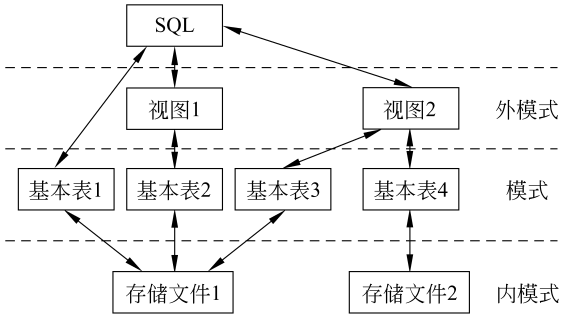

图 3.1 数据库系统三级模式结构

基本表是本身独立存在的表,在 SQL 中,一个关系就对应一个表。一些基本表对应一个存储文件,一个表可以有若干索引,索引也存放在存储文件中。

视图是从基本表或其他视图中导出的表,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。

存储文件的物理结构及存储方式等组成了关系数据库的内模式。对于不同数据库管理系统,其存储文件的物理结构及存储方式等往往是不同的,一般也是不公开的。

视图和基本表是 SQL 语言的主要操作对象,用户可以用 SQL 语言对视图和基本表

进行各种操作。在用户眼中,视图和基本表都是关系表,而存储文件对用户是透明的。

关系数据库系统三级模式结构直观示意图如图 3.2 所示。

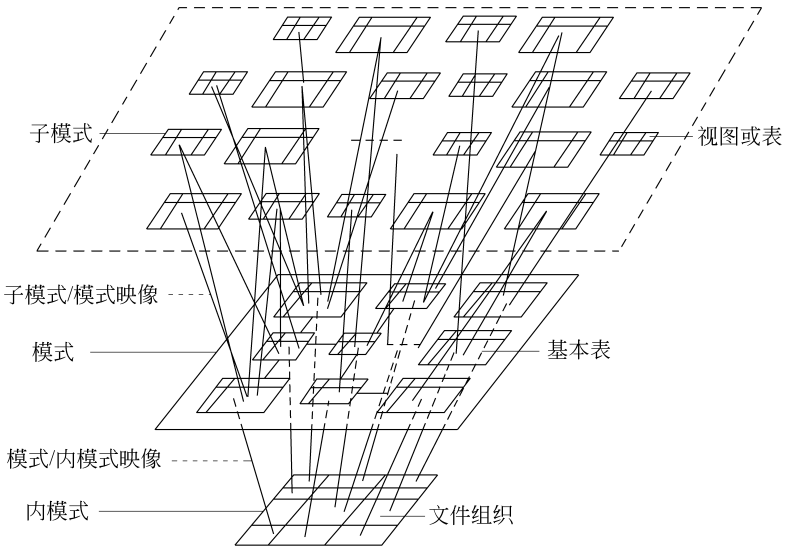

图 3.2 关系数据库系统三级模式结构示意图

3.1.3 SQL 语言的主要特点

SQL 语言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,同时又简捷易学的语言。SQL 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)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(层次模型、网状模型)的数据语言一般不同模式有不同的定义语言,数据操纵语言与各定义语言也不成一体。当用户数据库投入运行后,一般不支持联机实时修改各级模式。

而 SQL 语言则集数据定义语言(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,还可根据需要随时地逐步地修改模式,而不影响数据库的整体正常运行,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,用其完成某项请求,必须指定

存取路径。而用 SQL 语言进行数据操作，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 语句的具体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

SQL 语言采用集合操作方式，不仅查找结果可以是元组的集合（即关系），而且一次插入、删除、更新操作的对象也可以是元组的集合。非关系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 90 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，即如何用多重循环结构按照某条路径一条一条地把学生记录及其所有选课记录读出，并计算、判断后选择出来；而关系数据库中，一条 SELECT 命令就能完成该功能。

4. 以同一种语法结构提供两种使用方式

SQL 语言既是自含式语言，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（如 Java、Python、C#、C、COBOL、FORTRAN、PL/1）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 语言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. 语言简捷，易学易用

SQL 语言功能极强，但由于设计巧妙，语言十分简洁，完成数据查询（SELECT 命令），数据定义（如 CREATE、DROP、ALTER 等命令），数据操纵（如 INSERT、UPDATE、DELETE 等命令），数据控制（如 GRANT、REVOKE 等命令）的四大核心功能只用了 9 个动词。而且 SQL 语言语法简单，接近英语口语，因此易学易用。

3.2 SQL 数据定义

SQL 语言使用数据定义语言（Data Definition Language，DDL）实现其数据定义功能，可对数据库用户、基本表、视图和索引等进行定义、修改和删除。

3.2.1 字段数据类型

当用 SQL 语句定义表时，需要为表中的每一个字段设置一个数据类型，用来指定字段所存放的数据是何种类型的数据。

MySQL 8.0 支持的 SQL 数据类型有 5 种：数字数据类型、日期和时间数据类型、字符串（字符和字节）数据类型、空间数据类型和 JSON 数据类型等。

（1）数字数据类型：BIT[(M)]、TINYINT[(M)]、BOOL、BOOLEAN、SMALLINT

[(M)]、MEDIUMINT[(M)]、INT[(M)]、INTEGER[(M)]、BIGINT[(M)]、SERIAL、DECIMAL[(M[,D])]、DEC[(M[,D])]、NUMERIC[(M[,D])]、FIXED[(M[,D])]、FLOAT[(M,D)]、FLOAT(p)、DOUBLE[(M,D)]。

(2) 日期和时间数据类型：DATE、TIME[(fsp)]、DATETIME[(fsp)]、TIMESTAMP[(fsp)]和YEAR[(4)]。

(3) 字符串数据类型：CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM和SET。

(4) 空间数据类型：MySQL具有与OpenGIS类相对应的空间数据类型。某些空间数据类型保存单个几何图形值，数据类型有GEOMETRY、POINT、LINESTRING、POLYGON。其他空间数据类型保存值的集合，数据类型有MULTIPOINT、MULTILINESTRING、MULTIPOLYGON、GEOMETRYCOLLECTION。

(5) JSON数据类型：JSON。

各种数据类型约定如下。

(1) 对于整数类型，M表示最大显示宽度。对于浮点和定点类型，M是可存储的总位数(精度)。对于字符串类型，M是最大长度。M的最大允许值取决于数据类型。

(2) D适用于浮点和定点类型，并指示小数点后的位数(刻度)。最大可能值为30，但不应大于M－2。

(3) fsp适用于时间、日期时间和时间戳类型，并表示小数秒精度，即小数点后的秒的小数部分，如果给定，fsp值必须为0～6。

(4) p表示以位为单位的精度，但MySQL仅使用此值来确定结果数据类型是使用FLOAT还是DOUBLE。如果p从0到24，则数据类型变为浮点型，没有M或D值。如果p为25～53，则数据类型变为双精度，没有M或D值。

(5) 方括号表示类型定义的可选部分。

(6) 关于数据类型的详细说明与使用，请参照MySQL线上资料，如：https://dev.mysql.com/doc/refman/8.0/en/data-types.html。

3.2.2 创建、修改和删除数据表

1. 定义基本表

在SQL语言中，使用语句CREATE TABLE创建数据表，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>是所要定义的基本表的名字，必须是合法的标识符，最多可有128个字符，但本地临时表的表名(名称前有一个编号符#)最多只能包含116个字符。表名不允许重名，一个表可以由一个或多个属性(列)组成。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，当用户在表中操作数据时由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。

关系模型的完整性规则是对关系的某种约束条件。

1）实体完整性

（1）主码（PRIMARY KEY）：在一个基本表中只能定义一个 PRIMARY KEY 约束，对于指定为 PRIMARY KEY 的一个或多个列的组合，其中任何一个列都不能出现空值。PRIMARY KEY 既可用于列约束，也可用于表约束。PRIMARY KEY 用于定义列约束时的语法格式如下：

```
[CONSTRAINT <约束名>] PRIMARY KEY [ CLUSTERED | NONCLUSTERED ] [(column_name
[ ASC | DESC ] [ ,...n ])]
```

说明：[,...n]表示可以重复，下同。

（2）空值（NULL/NOT NULL）：空值不等于 0 也不等于空白，而是表示不知道、不确定、没有意义的意思，该约束只能用于列约束，其语法格式如下：

```
[CONSTRAINT <约束名>] [NULL|NOT NULL]
```

（3）唯一值（UNIQUE）：表示在某一列或多个列的组合上的取值必须唯一，系统会自动为其建立唯一索引。UNIQUE 约束可用于列约束，也可用于表约束，语法格式如下：

```
[CONSTRAINT <约束名>] UNIQUE [ CLUSTERED | NONCLUSTERED ] [(column_name [ ASC |
DESC ] [ ,...n ])]
```

2）参照完整性

FOREIGN KEY 约束指定某一个或一组列作为外部键，其中，包含外部键的表称为从表，包含外部键引用的主键或唯一键的表称为主表。系统保证从表在外部键上的取值是主表中某一个主键或唯一键值，或者取空值，以此保证两个表之间的连接，确保了实体的参照完整性。

FOREIGN KEY 既可用于列约束，也可用于表约束，其语法格式分别为

```
[CONSTRAINT <约束名>] FOREIGN KEY REFERENCES <主表名>(<列名>)
```

或

```
[CONSTRAINT <约束名>] FOREIGN KEY [(<从表列名>[ ,...n ])] REFERENCES <主表名>
[(<主表列名>[ ,...n ])] [ON DELETE {CASCADE | NO ACTION} ] [ ON UPDATE {CASCADE | NO
ACTION}] [NOT FOR REPLICATION]
```

3）用户自定义的完整性约束规则

CHECK 可用于定义用户自定义的完整性约束规则，CHECK 既可用于列约束，也可用于表约束，其语法格式为

```
[CONSTRAINT <约束名>] CHECK [NOT FOR REPLICATION](<条件>)
```

下面以一个“学生-课程”数据库为例来说明，表内容请参见图 3.3。

S

学号 SNO	姓名 SN	性别 SEX	年龄 AGE	系别 DEPT
S1	李涛	男	19	信息
S2	王林	女	18	计算机
S3	陈高	女	21	自动化
S4	张杰	男	17	自动化
S5	吴小丽	女	19	信息
S6	徐敏敏	女	20	计算机

C

课程号 CNO	课程名 CN	学分 CT
C1	C 语言	4
C2	离散数学	2
C3	操作系统	3
C4	数据结构	4
C5	数据库	4
C6	汇编语言	3
C7	信息基础	2

SC

学号 SNO	课程号 CNO	成绩 SCORE
S1	C1	90
S1	C2	85
S2	C1	84
S2	C2	94
S2	C3	83
S3	C1	73
S3	C7	68
S3	C4	88
S3	C5	85
S4	C2	65
S4	C5	90
S4	C6	79
S5	C2	89

图 3.3 “学生-课程”数据库中的三表内容

“学生-课程”数据库中包括如下 3 个表。

(1) “学生”表 S 由学号(SNO)、姓名(SN)、性别(SEX)、年龄(AGE)、系别(DEPT)五个属性组成,可记为 S(SNO,SN,SEX,AGE,DEPT);

(2) “课程”表 C 由课程号(CNO)、课程名(CN)、学分(CT)三个属性组成,可记为 C(CNO,CN,CT);

(3) “学生选课”表 SC 由学号(SNO)、课程号(CNO)、成绩(SCORE)三个属性组成,可记为 SC(SNO,CNO,SCORE)。先创建数据库,并选择为当前数据库。命令为

```
CREATE DATABASE jxgl
GO
USE jxgl
```

例 3.1 建立一个“学生”表 S,它由学号 SNO、姓名 SN、性别 SEX、年龄 AGE、系别 DEPT 五个属性组成,其中学号属性为主键,姓名、年龄与性别不为空,假设姓名具有唯一性,并建立唯一索引,并且性别只能在“男”与“女”中选一个,年龄不能小于 0。

```
CREATE TABLE S
(  SNO CHAR(5) PRIMARY KEY,
   SN VARCHAR(8) NOT NULL,
   SEX CHAR(2) NOT NULL CHECK (SEX IN ('男','女')),
   AGE INT NOT NULL CHECK (AGE>0),
   DEPT VARCHAR(20),
```

```
    CONSTRAINT SN_U UNIQUE(SN)
)
```

例 3.2 建立“课程”表 C,它由课程号(CNO)、课程名(CN)、学分(CT)三个属性组成。CNO 为该表主键,学分大于或等于 1。

```
CREATE TABLE C
(  CNO CHAR(5) NOT NULL PRIMARY KEY,
   CN VARCHAR(20),
   CT INT CHECK(CT>=1))
```

例 3.3 建立“选修”关系表 SC,分别定义 SNO、CNO 为 SC 的外部键,(SNO,CNO)为该表的主键。

```
CREATE TABLE SC
(  SNO CHAR(5) NOT NULL,
   CNO CHAR(5) NOT NULL,
   SCORE NUMERIC(3,0),
   CONSTRAINT S_C_P PRIMARY KEY(SNO,CNO),
   CONSTRAINT S_F FOREIGN KEY(SNO) REFERENCES S(SNO),
   CONSTRAINT C_F FOREIGN KEY(CNO) REFERENCES C(CNO))
```

2. 修改基本表

由于分析设计不到位或应用需求的不断变化等原因,基本表结构的修改也是不可避免的,如增加新列和完整性约束、修改原有的列定义和完整性约束定义等。SQL 语言使用 ALTER TABLE 命令来完成这一功能,其部分语法格式参照下方二维码内容。

例 3.4 向 S 表增加“入学时间”列,其数据类型为日期型。

```
ALTER TABLE S ADD SCOME DATETIME
```

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半字长整数。

```
ALTER TABLE S MODIFY COLUMN AGE SMALLINT
```

修改原有的列定义,会使列中数据做新旧类型的自动转化,有可能会破坏已有数据。

例 3.6 删除例 3.4 中增加的“入学时间”列。

```
ALTER TABLE S DROP COLUMN SCOME
```

例 3.7 禁用或启用 S 中的`AGE`>0 的 CHECK 完整性(设完整性名称为 s_chk_2)。

```
ALTER TABLE S ALTER CONSTRAINT `s_chk_2` NOT ENFORCED;     /*禁用*/
ALTER TABLE S ALTER CONSTRAINT `s_chk_2` ENFORCED;         /*启用*/
```

3. 删除基本表

随着时间的变化,有些基本表无用了,便可将其删除。删除某基本表后,该表中数据及表结构将从数据库中彻底删除,表相关的对象如索引、视图、参照关系等也将同时删除或无法再使用,因此执行删除操作一定要格外小心。删除基本表命令的一般格式为

```
DROP TABLE <表名>
```

例 3.8 删除 S 表。

```
DROP TABLE S
```

注意:删除表需要相应的操作权限,一般只删除自己建立的无用表;执行删除命令后是否真能完成删除操作,还取决于其操作是否违反了完整性约束。

3.2.3 设计、创建和维护索引

1. 索引的概念

在现实生活中,人们经常借用索引的手段实现快速查找,如图书目录、词典索引等。同样道理,数据库中的索引是为了加速对表中元组(或记录)的检索而创建的一种分散存储结构(如 B^+ 树数据结构),它实际上是记录的关键字与其相应地址的对应表。索引是对表或视图而建立的,由索引页面组成。

改变表中的数据(如增加或删除记录)时,索引将自动更新。索引建立后,在查询使用该列时,系统将自动使用索引进行查询。索引是把双刃剑,由于要建立索引页面,索引也会减慢更新数据的速度。索引数目无限制,但索引越多,更新数据的速度越慢。对于仅用于查询的表可多建索引,对于数据更新频繁的表则应少建索引。

按照索引记录的存放位置可分为聚集索引(clustered index)与非聚集索引(non-clustered index)两类。聚集索引是指索引项的顺序与表中记录的物理顺序一致的索引组织;非聚集索引按照索引字段排列记录,该索引中索引的逻辑顺序与磁盘上记录的物理存储顺序不同。在检索记录时,聚集索引会比非聚集索引速度快,一个表中只能有一个聚集索引,而非聚集索引可以有多个。

2. 创建索引

创建索引的语句的一般格式为

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX <索引名>[index_type]
    ON <表名> (<列名>[(length)] | (expr) [ASC|DESC],...) [index_option]
    [algorithm_option | lock_option] ...
index_option: {KEY_BLOCK_SIZE [=] value| index_type | WITH PARSER parser_name |
COMMENT 'string' | {VISIBLE | INVISIBLE} | ENGINE _ATTRIBUTE [=] 'string' |
SECONDARY_ENGINE_ATTRIBUTE [=] 'string'
}
index_type: USING {BTREE | HASH}
```

```
algorithm_option: ALGORITHM [=] {DEFAULT | INPLACE | COPY}
lock_option: LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
… / * CREATE INDEX 命令的选项含义略 * /
```

索引可以建在表的一列或多列上，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 ASC(升序)和 DESC(降序)两种，默认值为 ASC。例如，执行下面的 CREATE INDEX 语句：

```
CREATE INDEX StuSN ON S(SN)
```

将会在 S 表的 SN(姓名)列上建立一个索引。

例 3.9 为学生-课程数据库中的 S、C、SC 三个表建立索引。其中，S 表按学号升序建唯一索引，C 表按课程号降序建立索引，SC 表按学号升序和课程号降序建索引。

```
CREATE UNIQUE INDEX S_SNO ON S(SNO)
CREATE INDEX C_CNO ON C(CNO DESC)
CREATE INDEX SC_SNO_CNO ON SC(SNO ASC,CNO DESC)
```

3. 删除索引

删除索引一般格式为

```
DROP INDEX <索引名>ON 表名 [algorithm_option | lock_option] ...
algorithm_option: ALGORITHM [=] {DEFAULT | INPLACE | COPY}
lock_option: LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

例 3.10 删除 S 表的 S_SNO 索引。

```
DROP INDEX S_SNO ON S
```

说明：索引一经建立，就由系统使用和维护它，一般无须用户干预。建立索引是为了减少查询操作的时间，但如果数据增、删、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。删除索引时，系统会同时从数据字典中删去有关该索引的描述。

3.3 SQL 数据查询

3.3.1 SELECT 命令的格式及其含义

数据查询是数据库中最常用的操作命令。SQL 语言提供 SELECT 语句，通过查询操作可以得到所需的信息。SELECT 语句的一般格式为

```
SELECT
  [ALL | DISTINCT | DISTINCTROW ] [HIGH_PRIORITY] [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
  select_expr [, select_expr] ... [into_option]
  [FROM <表名 1 或视图名 1>[[AS] 表别名 1] [,<表名 2 或视图名 2>[[AS] 表别名 2]] ...
[PARTITION partition_list]]
```

```
[WHERE where_condition]
[GROUP BY {<列名> | expr | position}, ... [WITH ROLLUP]]
[HAVING having_condition]
[WINDOW window_name AS (window_spec)[, window_name AS (window_spec)] ...]
[ORDER BY {<列名> | expr | position} [ASC | DESC], ... [WITH ROLLUP]]
[LIMIT {[offset,] row_count | row_count OFFSET offset}] [into_option]
[FOR {UPDATE | SHARE}[OF <表名>[,<表名>] ...] [NOWAIT | SKIP LOCKED] | LOCK IN
SHARE MODE] [into_option]

into_option: {INTO OUTFILE 'file_name' [CHARACTER SET charset_name]
export_options | INTO DUMPFILE 'file_name' | INTO var_name [, var_name] ...}
… /* SELECT 命令的详细选项含义略或见后续子句介绍 */
```

整个 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<列名>的值进行分组(假设只有一列分组列),该属性列值相等的元组为一个组,每个组将产生结果表中的一条记录,通常会对每组作用到集函数。如果 GROUP 子句带 HAVING 短语,则只有满足指定条件的组才给予输出。如果有 ORDER 子句,则结果表还要按 ORDER BY 后<列名>的值的升序或降序排序后(假设只有一列排序列)再输出。

HAVING 子句的分组筛选条件表达式格式基本同 WHERE 子句的可选筛选条件表达式的格式。不同的是 HAVING 子句的条件表达式中出现的属性列名应为 GROUP BY 子句中的分组列名。HAVING 子句的条件表达式中一般要使用到集函数 COUNT、SUM、AVG、MAX 或 MIN 等,因为只有这样才能表达出筛选分组的要求。

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或嵌套查询。一个 SELECT 语句至少需要 SELECT 与 FROM 两个子句,下面以学生-课程数据库(参阅 3.2.2 节)为例说明 SELECT 语句的各种用法。

3.3.2 SELECT 子句的基本使用

1. 查询指定列

例 3.11 查询全体学生的学号与姓名。

```
SELECT SNO,SN
FROM S
```

<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需要改变列的显示顺序。

例 3.12 查询前两位学生的姓名、学号、所在系。

```
SELECT SN,SNO,DEPT
FROM S LIMIT 0,2;  或  FROM S LIMIT 2 OFFSET 0;
```

这时结果表中的列的顺序与基表中不同,是按查询要求,先列出姓名属性,然后再列

出学号和所在系属性。

2. 查询全部列

例 3.13 查询全体学生的详细记录。

```
SELECT * FROM S
```

该 SELECT 语句实际上是无条件地把 S 表的全部信息都查询出来，所以也称为全表查询，这是最简单的一种查询命令形式。它等价于如下命令：

```
SELECT SNO,SN,SEX,AGE,DEPT FROM S
```

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，也可以是含或不含属性列的表达式，即可以将查询出来的属性列经过一定的计算后列出结果或是个常量表达式的值。

例 3.14 查询全体学生的姓名及其出生年份。

```
SELECT SN, 2005-AGE FROM S
```

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 2005 年)减去学生的年龄，这样，所得的即是学生的出生年份。输出的结果为

```
SN              2005-AGE
--------    ----------
李涛            1986
王林            1987
陈高            1984
张杰            1988
吴小丽          1986
徐敏敏          1985
```

<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 3.15 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 SN, '出生年份: ', 2005-AGE, lower(DEPT) FROM S
```

结果为

```
SN          出生年份:    2005-AGE     lower(DEPT)
--------  ----------  -----------  ----------
李涛          出生年份:    1986         信息
王林          出生年份:    1987         计算机
陈高          出生年份:    1984         自动化
张杰          出生年份:    1988         自动化
吴小丽        出生年份:    1986         信息
徐敏敏        出生年份:    1985         计算机
```

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。如对于上例,定义列别名方法如下。

注意:列别名与表达式间可以直接用空格分隔或用 as 关键字来连接。

```
SELECT SN NAME, '出生年份: ' BIRTH,2005-AGE BIRTHDAY, DEPT as DEPARTMENT
FROM S
```

执行结果为

```
NAME            BIRTH        BIRTHDAY        DEPARTMENT
--------  -----------  ------------  ---------------
李涛            出生年份:     1986            信息
王林            出生年份:     1987            计算机
陈高            出生年份:     1984            自动化
张杰            出生年份:     1988            自动化
吴小丽          出生年份:     1986            信息
徐敏敏          出生年份:     1985            计算机
```

3.3.3 WHERE 子句的基本使用

1. 消除取值重复的行

例 3.16 查询所有选修过课的学生的学号。

```
SELECT SNO FROM SC  或  SELECT ALL SNO FROM SC
```

ALL 是默认值,指定结果集中可以包含重复行,结果类似为

```
SNO
----
S1
S2
S3
S1
…
S3
```

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

```
SELECT DISTINCT SNO FROM SC  --DISTINCT 指定在结果集中只能包含唯一的行
```

执行结果为

```
SNO
----
S1
S2
S3
S4
S5
```

2. 指定 WHERE 查询条件

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 3.1 所示。

表 3.1 常用的查询条件

查询条件	谓 词
比较运算符	=,>,<.>=,<=,!=,<>,!>,!<; Not(上述比较运算符构成的比较关系表达式)
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,NOT

1) 比较运算符

例 3.17 查询计算机系全体学生的名单。

```
SELECT SN
FROM S
WHERE DEPT='计算机'
```

例 3.18 查询所有年龄在 20 岁以下的学生姓名及其年龄。

```
SELECT SN, AGE FROM S WHERE AGE <20
```

或

```
SELECT SN, AGE FROM S WHERE NOT AGE>=20
```

例 3.19 查询考试成绩有不及格的学生的学号。

```
SELECT DISTINCT SNO FROM SC WHERE SCORE< 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

例 3.20 查询年龄在 20 至 23 岁之间(包括 20 和 23)的学生的姓名、系别和年龄。

```
SELECT SN,DEPT,AGE  FROM S WHERE AGE BETWEEN 20 AND 23
```

与 BETWEEN…AND…相对的谓词是 NOT BETWEEN…AND…。

例 3.21 查询年龄不在 20 岁至 23 岁之间的学生姓名、系别和年龄。

```
SELECT SN,DEPT,AGE FROM S WHERE AGE NOT BETWEEN 20 AND 23
```

3) 确定集合

例 3.22 查询信息系、自动化系和计算机系的学生的姓名和性别。

```
SELECT SN,SEX FROM S WHERE DEPT IN ('信息','自动化','计算机')
```

与 IN 相对的谓词是 NOT IN,用于查找属性值不属于指定集合的元组。

例 3.23 查询既不是信息系、自动化系,也不是计算机系的学生的姓名和性别。

```
SELECT SN,SEX FROM S WHERE DEPT NOT IN ('信息','自动化','计算机')
```

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:

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

其含义是查找指定的属性列值与<匹配串>相匹配的元组。<匹配串>可以是一个完整的字符串,也可以含有通配符%、_、[]与[^]等。其含义见表 3.2。ESCAPE <换码字符>的功能说明见例 3.28。

表 3.2 通配符及其含义

通配符	描 述	示 例
%(百分号)	代表零个或多个字符的任意字符串	WHERE title LIKE '%computer%'表达书名任意位置包含单词 computer 的条件
_(下画线)	代表任何单个字符(长度可以为 0)	WHERE au_fname LIKE '_ean'表达以 ean 结尾的所有 4 个字母的名字(Dean、Sean 等)的条件
[ ](中括号)	指定范围(如[a-f])或集合(如[abcdef])中的任何单个字符	WHERE au_lname RLIKE '[C-P]arsen'表达以 arsen 结尾且以介于 C 与 P 之间的任何单个字符开始的作者姓氏的条件,如 Carsen、Larsen、Karsen 等。RLIKE 可换用 regexp
[^]	不属于指定范围(如[^a-f])或不属于指定集合(如[^abcdef])的任何单个字符	WHERE au_lname RLIKE 'de[^l]%'表达以 de 开始且其后的字母不为 l 的所有作者的姓氏的条件。RLIKE 可换用 regexp

例 3.24 查询所有姓“李”的学生的姓名、学号和性别。

```
SELECT SN, SNO, SEX FROM S WHERE SN LIKE '李%'
```

例 3.25 查询姓“欧阳”且全名为三个汉字的学生的姓名。

```
SELECT SN FROM S WHERE SN LIKE '欧阳_'
```

例 3.26 查询名字中第二字为“涛”字的学生的姓名和学号。

```
SELECT SN, SNO FROM S WHERE SN LIKE '_涛%'
```

例 3.27 查询所有不姓“吴”的学生姓名。

```
SELECT SN, SNO, SEX FROM S WHERE SN NOT LIKE '吴%'
```

如果用户要查询的匹配字符串本身就含有“%”或“_”字符,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 查询 DB_Design 课程的课程号和学分。

```
SELECT CNO, CT FROM C WHERE CN LIKE "DB\_Design"; #使用缺省的换码字符\
SELECT CNO, CT FROM C WHERE CN LIKE 'DB$_Design' ESCAPE '$';
SELECT CNO, CT FROM C WHERE CN LIKE CONCAT("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"_"或"%"不再具有通配符的含义,而是取其本身含义,即被转义为普通的"_"或"%"字符。

注意:ESCAPE 定义的换码字符'$'可以换成其他字符(\除外,因为\为缺省的换码字符)。

5) 涉及空值的查询

例 3.29 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,下面查询缺少成绩的学生的学号和相应的课程号。

```
SELECT SNO, CNO FROM SC WHERE SCORE IS NULL
```

注意:这里的 IS 不能用等号("=")代替。

例 3.30 查询所有成绩记录的学生学号和课程号。

```
SELECT SNO, CNO FROM SC WHERE SCORE IS NOT NULL
```

6) 多重条件查询

逻辑运算符 AND、OR 和 NOT 可用来联结多个查询条件。优先级 NOT 最高,接着是 AND,OR 优先级最低,但用户可以用括号改变运算的优先顺序。

例 3.31 查询计算机系年龄在 20 岁以下的学生姓名。

```
SELECT SN FROM S WHERE DEPT='计算机' AND AGE<20
```

例 3.32 IN 谓词实际上是多个 OR 运算符的缩写,因此"查询信息系、自动化系和计算机系的学生的姓名和性别"一题,也可以用 OR 运算符写成如下等价形式:

```
SELECT SN, SEX FROM S
WHERE DEPT='计算机' OR DEPT='信息' OR DEPT='自动化'
```

或

```
SELECT SN, SEX FROM S
WHERE NOT(DEPT<>'计算机' AND DEPT<>'信息' AND DEPT<>'自动化')
```

3.3.4 常用集函数及统计汇总查询

为了进一步方便用户,增强检索功能,SQL 提供了许多集函数,如表 3.3 所示。

表 3.3 常用集函数

COUNT({[ALL|DISTINCT] expression}|*)	返回组中项目的数量。Expression 一般是指<列名>,下同。COUNT(*)表示对元组(或记录)计数
SUM([ALL|DISTINCT] expression)	返回表达式中所有值的和,或只返回 DISTINCT 值的和。SUM 只能用于数字列。空值将被忽略
AVG([ALL|DISTINCT] expression)	返回组中值的平均值。空值将被忽略
MAX([ALL|DISTINCT]expression)	返回组中值的最大值。空值将被忽略或为最小值
MIN([ALL|DISTINCT] expression)	返回组中值的最小值。空值将被忽略或为最小值

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而统计或汇总。

例 3.33 查询学生总人数。

```
SELECT COUNT(*) FROM S
```

例 3.34 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 3.35 计算 C1 课程的学生人数、最高成绩、最低成绩及平均成绩。

```
SELECT COUNT(*),MAX(SCORE),MIN(SCORE),AVG(SCORE)
FROM SC WHERE CNO='C1'
```

3.3.5 分组查询

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,即整个查询结果为一组对应统计产生一个函数值。否则,集函数将作用于每一个组,即每一组分别统计,分别产生一个函数值。

例 3.36 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,然后对每一组作用集函数 COUNT 以求得该组的学生人数,执行结果为

```
CNO   COUNT(SNO)
----  ----------
C1    3
C2    4
C3    1
C4    1
C5    2
C6    1
C7    1
```

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

例 3.37 查询有 3 人以上学生(包括 3 人)选修的课程的课程号及选修人数。

```
SELECT CNO,COUNT(SNO) FROM SC GROUP BY CNO HAVING COUNT(*)>=3
```

结果为

```
CNO    COUNT(SNO)
----   --------
C1     3
C2     4
```

例 3.38 对(Sage,Ssex)、(Sage)值的每个不同值统计学生人数,并还能统计出总人数。

```
SELECT Sage,Ssex,count(*) FROM Student
GROUP BY Sage,Ssex with rollup
```

注意:①有 GROUP BY 子句,才能使用 HAVING 子句;②有 GROUP BY 子句,则 SELECT 子句中只能出现 GROUP BY 子句中的分组列名与集函数;③同样,使用 HAVING 子句条件表达时,也只能使用分组列名与集函数。有 GROUP BY 子句时,SELECT 子句或 HAVING 子句中使用非分组列名是错误的。

3.3.6 查询的排序

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(通常是元组添加到表中的先后顺序)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,其中升序 ASC 为默认值。

例 3.39 查询选修了 C3 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, SCORE
FROM SC
WHERE CNO='C3'
ORDER BY SCORE DESC
```

前面已经提到,可能有些学生选修了 C3 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,空值(NULL)一般被认为是最小值。

例 3.40 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 S ORDER BY DEPT,AGE DESC
```

3.3.7 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,它们共同关联着提供有用的信息。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,而广义笛卡儿积一般不常用。

1. 等值与非等值连接查询

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>

当比较运算符为"="时,称为**等值连接**。使用其他运算符的连接称为**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,另一个是整数型就不允许了,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,查找满足连接条件的元组,每找到一个满足条件的元组,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,然后再从头开始顺序扫描或按索引扫描表2,查找满足连接条件的元组,每找到一个满足条件的元组,就将表1中的第二个元组与该元组拼接起来,形成结果表中一个元组。重复上述操作,直到表1全部元组都处理完毕为止。

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S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S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 *
FROM S, SC
WHERE S.SNO=SC.SNO  --若省略 WHERE 即为 S 与 SC 两表的广义笛卡儿积操作
```

连接运算中有两种特殊情况,一种称为**广义笛卡儿积连接**,另一种称为**自然连接**。

广义笛卡儿积连接是不带连接谓词的连接。两个表的广义笛卡儿积连接即两表中元组的交叉乘积,也即其中一表中的每一元组都要与另一表中的每一元组做拼接,因此结果表往往很大。

如果是按照两个表中的相同属性进行等值连接,且目标列中去掉了重复的属性列,但保留了所有不重复的属性列,则称为**自然连接**。

例3.42 自然连接S和SC表。

```
SELECT S.SNO, SN, SEX, AGE, DEPT, CNO, SCORE
FROM S, SC
WHERE S.SNO=SC.SNO
```

在本查询中,由于SN、SEX、AGE、DEPT、CNO和SCORE属性列在S与SC表中是唯一的,因此引用时可以去掉表名前缀。而SNO在两个表都出现了,因此引用时必须加上表名前缀,以明确属性所属的表。该查询的执行结果不再出现SC.SNO列。

2. 自身连接

连接操作不仅可以在两个表之间进行，也可以是一个表与其自己进行连接，这种连接称为表的**自身连接**。

例 3.43 查询比李涛年龄大的学生的姓名、年龄和李涛的年龄。

要查询的内容均在同一表 S 中，可以将表 S 分别取两个别名，一个是 X，一个是 Y。将 X，Y 中满足比李涛年龄大的行连接起来。这实际上是同一表 S 的大于连接。

完成该查询的 SQL 语句为

```
SELECT X.SN AS 姓名,X.AGE AS 年龄,Y.AGE AS 李涛的年龄
FROM S AS X, S AS Y
WHERE X.AGE>Y.AGE AND Y.SN='李涛'
```

结果为

```
姓名      年龄    李涛的年龄
------  -----  ------------
陈高      21        19
徐敏敏    20        19
```

注意：SELECT 语句的可读性可通过为表指定别名来提高，别名也称为相关名称或范围变量。指派表的别名时，可以使用也可以不使用 AS 关键字，如上 SQL 命令也可表示为

```
SELECT X.SN 姓名, X.AGE 年龄, Y.AGE 李涛的年龄
FROM S X,S Y
WHERE X.AGE>Y.AGE AND Y.SN='李涛'
```

3. 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，如在例 3.41 和例 3.42 的结果表中没有关于学生 S6 的信息，原因在于她没有选课，在 SC 表中没有相应的元组。但是有时想以 S 表为主体列出每个学生的基本情况及其选课情况，若某个学生没有选课，则只输出其基本情况信息，其选课信息为空值即可，这时就需要使用外连接([Outer] Join)。外连接的运算符通常为 *，有的关系数据库中也用＋，使它出现在＝左边或右边。如下 MySQL 中使用类英语的表示方式([Outer] Join)来表达外连接。

这样，可以将例 3.42 改写如下：

```
SELECT S.SNO, SN, SEX, AGE, DEPT, CNO, SCORE
FROM S LEFT Outer JOIN SC ON S.SNO=SC.SNO
```

结果为

```
SNO  SN      SEX  AGE  DEPT        CNO  SCORE
---  ------  ---  ---  ----------  ----  ------
S1   李涛     男    19   信息          C1   90
S1   李涛     男    19   信息          C2   85
```