第3章 关系数据库语言SQL

- 3.1 SQL概述
- 3.2 SQL数据定义
- 3.3 SQL数据查询
- 3.4 SQL数据操纵
- 3.5 视图
- 3.6 SQL数据控制功能
- 3.7 嵌入式SQL

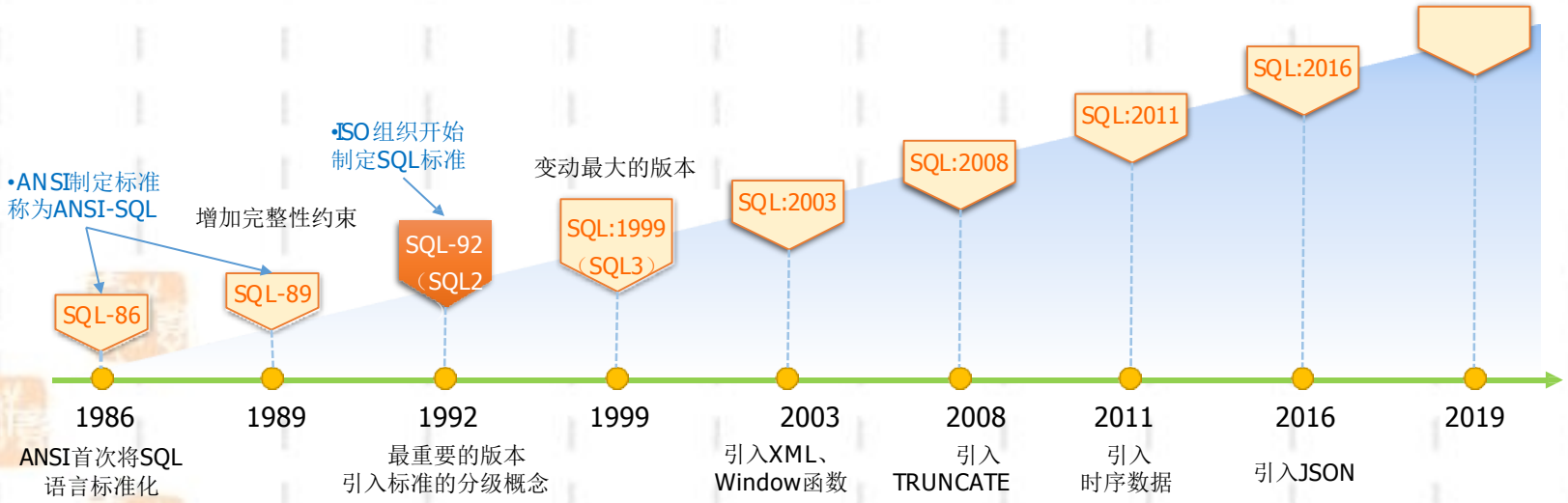
3.1 SQL概述

- SQL(Structured Query Language)是结构化查询语言的简称，是关系数据库的标准语言。
- SQL是一种通用的、功能极强的关系数据库语言，是对关系数据存取的标准接口，也是不同数据库系统之间互操作的基础。
- SQL基于关系代数和元组关系演算，集数据查询、数据操作、数据定义和数据控制功能于一体。
- SQL的功能包括数据插入、查询、更新和删除，数据库模式创建和修改，以及数据访问控制。

SQL起源

- 1972：IBM开始研制System R系统，配置数据库语言SQUARE
  - SQUARE (Specifying Queries As Relational Expressions)
  - 使用大量数学符号
- 1974：Boyce和Chamberlin将SQUARE修改为SEQUEL
  - SEQUEL (Structured English QUEry Language)
  - 去掉了数学符号，以英语单词和结构式语法代替查询
  - 后简称为SQL (Structured Query Language)
- 1970s末起：主流数据库厂商纷纷支持SQL
  - Oracle、DB2、Sybase

SQL发展历程


•ANSI制定标准
称为ANSI-SQL
增加完整性约束
•ISO组织开始
制定SQL标准
变动最大的版本
SQL-86
SQL-89
SQL-92
（SQL2）
SQL:1999
（SQL3）
SQL:2003
SQL:2008
SQL:2011
SQL:2016
1986
1989
1992
1999
2003
2008
2011
2016
2019
ANSI首次将SQL
语言标准化
最重要的版本
引入标准的分级概念
引入XML、
Window函数
引入
TRUNCATE
引入
时序数据
引入JSON

SQL的版本

标准	文档	页数	发布日期
SQL/86	ANSI X3.135-1986， ISO/IEC 9075:1986	103	1986.10
SQL/89（SQL 1）	ANSI X3.135-1989， ISO/IEC 9075:1989	120	1989年
SQL/92（SQL 2）	ANSI X3.135-1992, ISO/IEC 9075:1992	622	1992年
SQL/99（SQL 3）	ISO/IEC 9075:1999	1700	1999年
SQL2003（SQL 4）	ISO/IEC 9075:2003	3600	2003年
SQL2008	ISO/IEC 9075:2008	3777	2006年
SQL2011	ISO/IEC 9075:2011	4000+	2011年
SQL2016	ISO/IEC 9075:2016	5000+	2016年

注：目前，没有任何RDBMS能够支持SQL标准的所有概念和特性

SQL的特点

- 综合统一
- 高度非过程化
- 面向集合的操作方式

SQL的特点(1)

- 综合统一
  - 集数据定义语言(DDL)，数据操纵语言(DML)， 数据控制语言(DCL)功能于一体。
  - 可以独立完成数据库生命周期中的全部活动：
    - 定义和修改、删除关系模式，定义和删除视图，插入数据，建立数据库;
    - 对数据库中的数据进行查询和更新;
    - 数据库重构和维护
    - 数据库安全性、完整性控制，以及事务控制
    - 嵌入式SQL和动态SQL定义
  - 用户数据库投入运行后，可根据需要随时逐步修改模式，不影响数据库的运行。
  - 数据操作符统一。

SQL的特点(2)

- 高度非过程化
  - 非关系数据模型的数据操纵语言"面向过程"，特点是必须指定存取路径。
  - SQL只要提出"做什么(What)"，无须了解具体存取路径。
  - 存取路径的选择以及SQL的操作过程由系统自动完成。

SQL的特点(3)

- 面向集合的操作方式
  - 非关系数据模型采用面向记录的操作方式，操作对象是一条记录。
  - SQL采用集合操作方式
    - 操作对象、查找结果可以是元组的集合。
    - 一次插入、删除、更新操作的对象可以是元组的集合。

SQL的特点(4)

- 以同一种语法结构提供多种使用方式
  - SQL是独立的语言
    - 能够独立地用于联机交互的使用方式。
  - SQL又是嵌入式语言
    - SQL能够嵌入到高级语言（例如C，C++，Java）程序中，供程序员设计程序时使用。

SQL的特点(5)

- 语言简洁，易学易用
  - **SQL功能极强，核心功能只用了9个动词**

表　　SQL 的动词

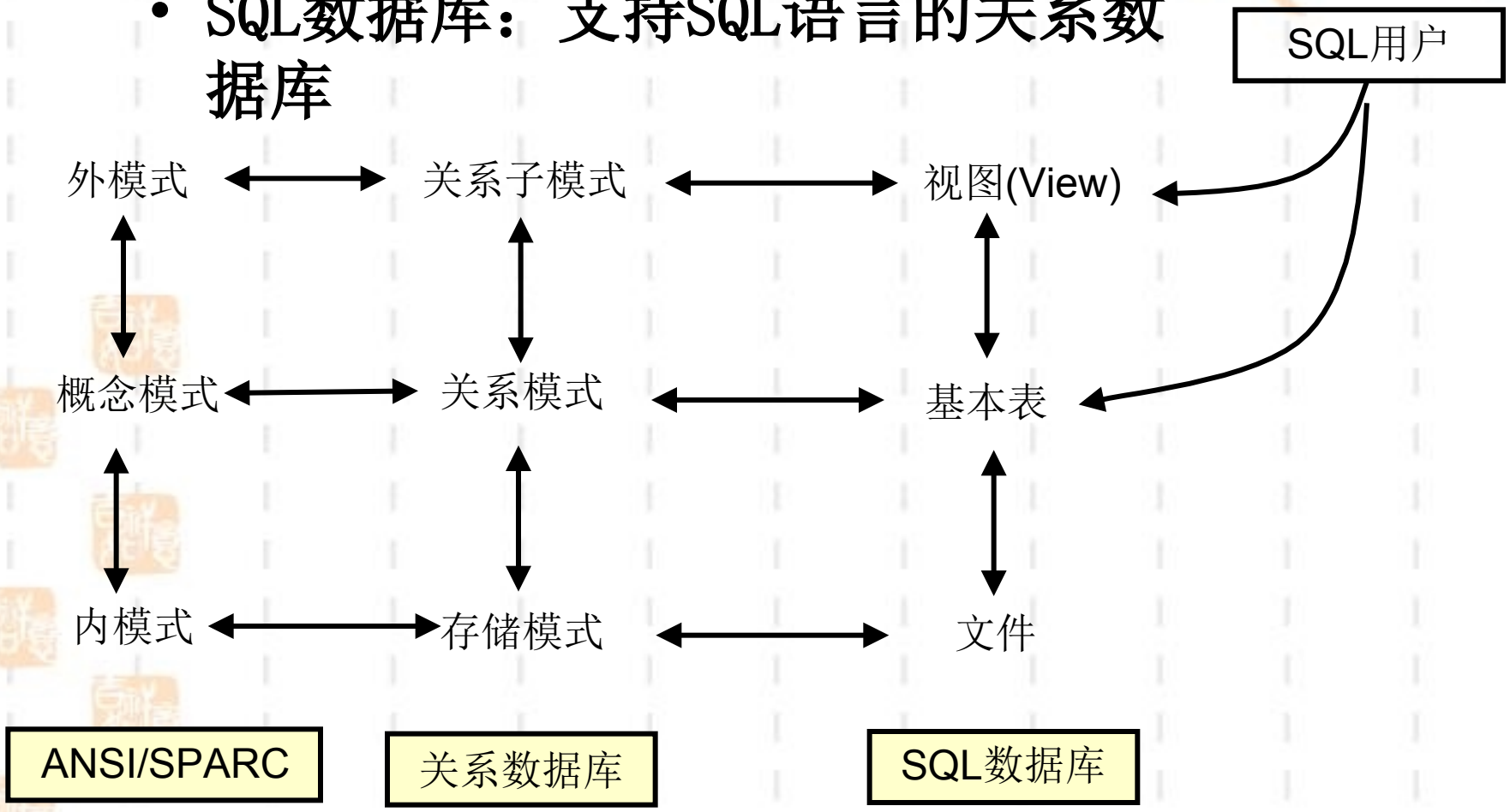
SQL 功　能	动词
数 据 查 询	SELECT
数 据 定 义	CREATE，DROP，ALTER
数 据 操 纵	INSERT，UPDATE，DELETE
数 据 控 制	GRANT，REVOKE

SQL基本概念

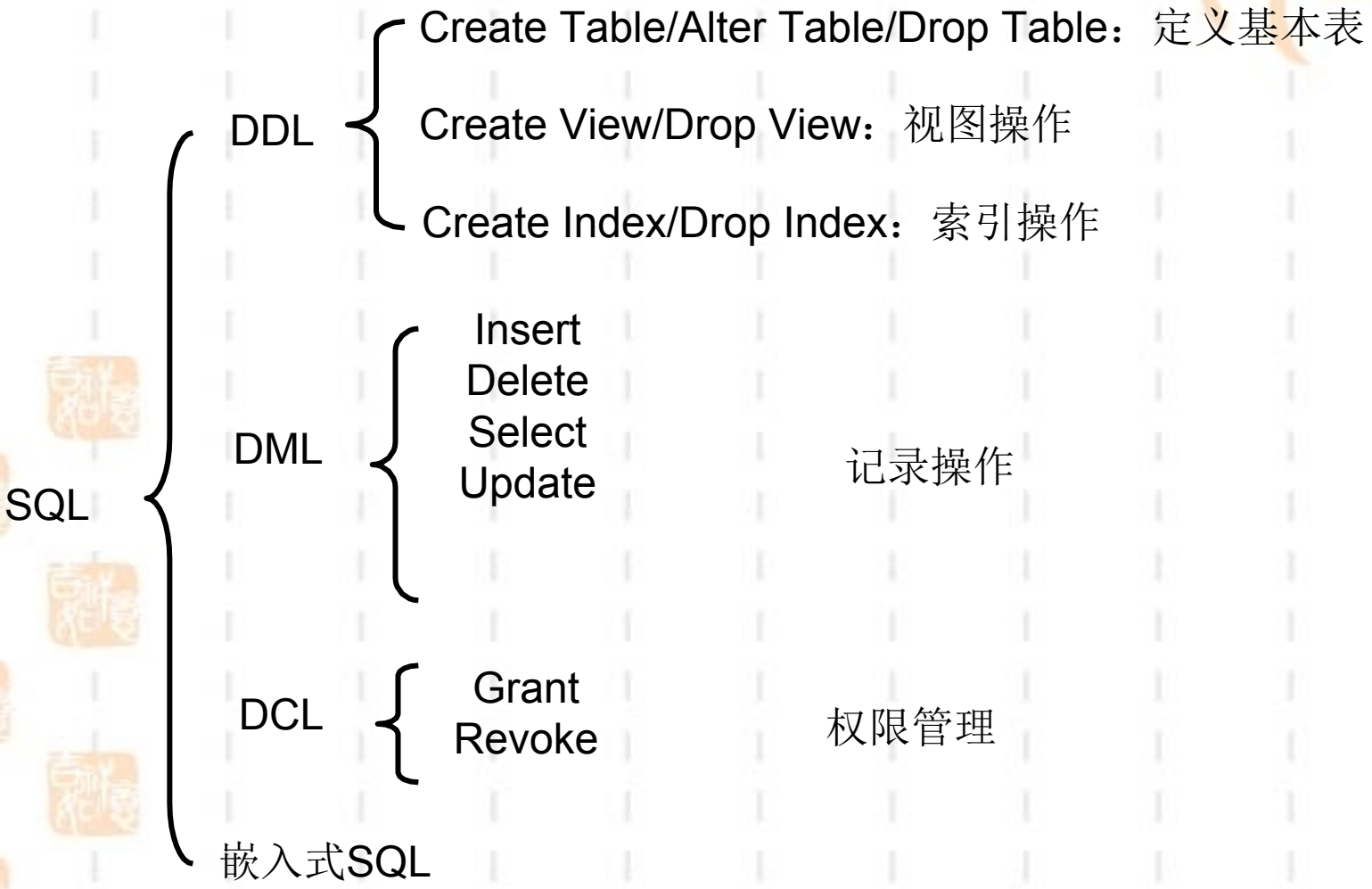
- 基本表(Table)
  - 本身独立存在的表
  - SQL中一个关系对应一个基本表
  - 一个(或多个)基本表对应一个存储文件
  - 一个表可以带若干索引
- 存储文件(File)
  - 逻辑结构组成了关系数据库的内模式。
  - 物理结构是任意的，对用户透明。
- 视图(View)
  - 从一个或几个基本表导出的“虚表”。
  - 数据库中只存放视图的定义而不存放视图对应的数据
  - 用户可以在视图上再定义视图。

SQL数据库三级体系结构

- SQL数据库：支持SQL语言的关系数据库



SQL的组成


SQL
DDL
Create Table/Alter Table/Drop Table：定义基本表
Create View/Drop View：视图操作
Create Index/Drop Index：索引操作
DML
Insert
Delete
Select
Update
记录操作
DCL
Grant
Revoke
权限管理
嵌入式SQL

3.2 SQL数据定义

- SQL数据类型
- 创建基本表：Create Table
- 修改基本表：Alter Table
- 删除基本表：Drop Table

SQL数据类型

- SQL中域的概念用数据类型来实现
- 定义列时需要指明其数据类型及长度
- 选用哪种数据类型需考虑
  - 取值范围
  - 要做哪些运算

SQL数据类型

类型	数据类型举例	说明
binary	binary large object(BLOB)	以16进制格式存储二进制字符串的值。
bit string	bit bit varying	可存储二进制和16进制数据，BIT类型具有可定义的固定长度，而BIT VARYING类型具有可定义的可变长度。
boolean	Boolean	存储真值 TRUE、FALSE或UNKNOWN
character	char character varying(VARCHAR) national character(NCHAR) national character varying (NVARCHAR) character large object(CLOB) national character large object(NCLOB)	可存储字符集中的任意字符组合。可变长度的数据类型允许字符长度变化，而其它数据类型字符长度是固定的。可变长度数据类型会自动删除后继的空格，定长数据类型会自动添加空格以补齐字符定义长度。

SQL数据类型

类型	数据类型举例	说明
numeric	integer(INT) smallint numeric decimal(DEC) float real double precision	存储数字的准确值（整数或小数）或近似值（浮点型）。INT和SMALLINT类型在预定义的精度和范围内存储数字的精确值。DEC和NUMERIC类型在可定义的精度和范围内存储数字的精确值。FLOAT是可定义精度的近似数类型。REAL和DOUBLE PRECISION是具有预定义精度的近似数类型。
temporal	date time time with time zone timestamp timestamp with time zone interval	这些数据类型处理关于时间的值。DATE和TIME分别处理日期和时间。有WITH TIME ZONE后缀的类型常包括一个时区的偏移。TIMESTAMP类型存储按照机器当前运行时间计算出来的值。INTERVAL类型表示时间的间隔。

创建基本表

- 格式：

CREATE TABLE <表名> (

<列定义>

[, <列定义>] …]

……

[<表约束>]);

列定义

- **格式**

  〈列名〉〈列类型〉[DEFAULT 〈默认值〉]
  [[NOT] NULL] [〈列约束〉]

- **列名**
  - 字母开头，可含字母、数字、＃、$、_
  - <=30字符
- **列类型**
  - Char(n)【定长字符串类型】
  - Varchar2(n)【可变长字符串类型】
  - Number【数值型】
  - Date【日期时间型】
  - ……

列定义

- **默认值**
  - 当往表中插入一条新记录时，如果某列上有默认值，并且新记录中未指定该列的值，则自动以默认值填充
  - 例：

    Gender Char(1) DEFAULT ‘F’
- **NOT NULL**
  - 不允许空值，是Check约束的简化

列定义

- **列约束**
  - 格式
    [Constraint〈约束名〉] 〈约束类型〉
  - 必须在每个列定义后定义
  - 只对当前列有效
  - 可以使用4种类型的约束
  - 例：
    - SNO char(8) Constraint PK_Student Primary Key
    - SNO char(8) Primary Key

表约束

- 在全部列定义完成后定义，可以有多个表约束子句
- 多个列上的约束必须使用表约束
- 单列上的约束可以用列约束，也可用表约束

SQL约束

- 主键约束（Primary Key）
  - 实体完整性
- 唯一键约束（Unique Key）
  - 候选键
- 外键约束（Foreign Key）
  - 参照完整性
- 检查约束（Check）
  - 用户定义完整性

4种约束既可以作为列约束，也可以作为表约束

Primary Key约束

- 定义主键：不许有空值，也不可重复
- 一个表只能有一个主键
- 例：

```
Create Table Student(
   Sno  Varchar2(10) Constraint Primary Key,
   Sname Varchar2(20),
   Age Number(3) ,
   Gender Char(1));
```

Unique约束

- 唯一性约束：值不可重复，但可以为空
- 多个列上的约束只能用表约束来实现
- 对空值的处理：
  - 若约束列中有一列不为空，就实施约束；
  - 若所有约束列都为空，则不实施约束
- 例：

```
Create Table Department(
  DNO    Varchar2(10),
  DNAME Varchar2(20),
  SCHOOL Char(20),
  Constraint UQ_D Unique(NAME, SCHOOL)
  )
```

Foreign Key约束

- 外键约束：表中某列值引用其它表的主键列或Unique列
- 例：

```
Create Table Student(--学生表
    SNO  Varchar2(10) Constraint PK_S Primary Key,
    Sname Varchar2(20),
    Age Number(3))
Create Table SC( ——选课表
    SNO  Varchar2(10) Constraint FK_SC References
   Student(SNO), --定义外键
    CNO Varchar2(20),
    Score Number(3) ,
    …
```

Check约束

- 检查约束：用户自定义某些列上的约束
- 例：

```
Constraint CK_S1 Check (age>15)

Constraint CK_S2 Check (Gender In
('M','F'))

Constraint CK_SC Check (Score>=0 and
Score<=100)

Constraint CK_S3 Check (Sname Is Not NULL)
```

示例：学生-课程数据库2022-3-9

Student表

学 号 Sno	姓 名 Sname	性 别 SGender	年 龄 Sage	所在系 Sdept
200215121	李勇	男	20	SE
200215122	刘晨	女	19	SE
200215123	王敏	女	18	MA
200515125	张立	男	19	IS

示例：学生-课程数据库

Course表

课程号 Cno	课程名 Cname	先行课 Cpno	学分 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

示例：学生-课程数据库

SC表

学 号 Sno	课程号 Cno	成绩 Grade
200215121	1	92
200215121	2	85
200215121	3	88
200215122	2	90
200215122	3	80

创建学生表Student

主码

```
CREATE TABLE Student
(Sno   CHAR(9) PRIMARY KEY,   /* 列级完整性约束条件*/
 Sname VARCHAR(20) ,
 SGender CHAR(2),
 Sage    SMALLINT,
 Sdept   CHAR(20)
);
```

创建课程表Course

```
CREATE TABLE  Course
(Cno     CHAR(4) PRIMARY KEY,
 Cname   CHAR(40),
 Cpno    CHAR(4) ,
 Ccredit  SMALLINT,
 FOREIGN KEY (Cpno) REFERENCES  Course(Cno)
);
```

先修课

Cpno是外码
被参照表是Course
被参照列是Cno

创建选课表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*/
);
```

修改基本表

- 格式：

```
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>];
```

- <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

修改基本表

- 例3-1：向STUDENT表增加“入学时间” (SCOME)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3-2：将学生姓名SNAME的长度增加到30

```
ALTER TABLE STUDENT
MODIFY SNAME VARCHAR(30);
```

  - 修改原有的列定义有可能会破坏已有数据
- 例3-3：删除关于学号为主键的约束

```
ALTER TABLE STUDENT  DROP PRIMARY KEY;
```

删除基本表

- 格式：

  DROP TABLE <表名>

- 例3-4： 删除STUDENT表

  DROP TABLE STUDENT;

- 基本表定义一旦删除，表中的数据、在此表上建立的索引都将自动被删除掉。

索引

- RDBMS中索引一般采用B+树、HASH索引来实现
  - B+树索引具有动态平衡的优点
  - HASH索引具有查找速度快的特点
- 采用B+树还是HASH索引，由具体的RDBMS决定。
- 索引是关系数据库的内部实现技术，属于内模式的范畴。
- CREATE INDEX语句定义索引时，可以定义索引是唯一索引、非唯一索引或聚簇索引。

创建索引

■格式

CREATE [UNIQUE] [CLUSTER] INDEX <索引名>

ON <表 名> (<列 名> [ASC | DESC] [,<列 名> [ASC|DESC]]…)

■例3-5：

CREATE INDEX SnoIndex ON SC (SNO DESC)

唯一值索引

- 对于已含重复值的属性列不能建UNIQUE索引。
- 对某个列建立UNIQUE索引后，插入新记录时DBMS会自动检查新记录在该列上是否取了重复值,相当于增加了一个UNIQUE约束。

聚簇索引

- 建立聚簇索引后，基表中数据也需要按指定的聚簇属性值的升序或降序存放,即聚簇索引的索引项顺序与表中记录的物理顺序一致。

- 例3-6：在Student表的Sname（姓名）列上建立一个聚簇索引

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

删除索引

- **格式**

  DROP INDEX <索引名>

- **例3-7：**

  DROP INDEX CnameIndex;

3.3 SQL数据查询语言

- 基本查询语句
- 单表查询
- 连接查询
- 嵌套查询
- 集合查询

基本查询语句

- **格式：**

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 <列表达式>,…
[HAVING <条件表达式>]]
[ORDER BY <列表达式> [ASC|DESC],…];

SELECT语句的基本结构

- 在关系代数中最常用的式子是下列表达式：

$$\pi_{A1,\ldots,An}(\sigma_{cond}(R_1\times\ldots\times R_m))$$

- 针对上述表达式，SQL设计了SELECT—FROM—WHERE句型：

SELECT A_1，…，A_n

FROM R_1，…，R_m

WHERE cond

单表查询

- 查询仅涉及一个表
- 选择表中的若干列
- 选择表中的若干元组
- ORDER BY子句
- 聚集函数
- GROUP BY子句

选择表中若干列

- 查询指定列
- 例3-8：查询全体学生的学号与姓名

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

- 例3-9：查询全体学生的姓名、学号、所在系。

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

查询全部列

- 选出所有属性列：
  - 在SELECT关键字后面列出所有列名
  - 将<目标列表达式>指定为 *
- 例3-10：查询全体学生的详细记录

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

或

```
SELECT  *
FROM Student;
```

查询经过计算的值

- SELECT子句的<目标列表达式>可以为：
  - 算术表达式
  - 字符串常量
  - 函数
  - 列别名

查询经过计算的值

- 例3-11：查全体学生的姓名及其出生年份

```
SELECT Sname,  YEAR(Bdate)
FROM Student;
```

- 例3-12：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有系名

```
SELECT Sname,  'Year of Birth:', YEAR(Bdate) ,
  ISLOWER(Sdept)
FROM Student;
```

查询经过计算的值

- 使用列别名改变查询结果的列标题:
- 例3-13:

```
SELECT Sname NAME, 'Year of Birth: '  BIRTH,
   YEAR(Bdate) BIRTHDAY, LOWER(Sdept)  DEPARTMENT
FROM Student;
```

选择表中的若干元组

- 消除取值重复的行：
- 例3-14：查询选修了课程的学生学号(不取消重复行)

```
SELECT Sno    FROM SC;
```

等价于：

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

- 如果指定DISTINCT关键词，则去掉表中重复的行。
- 例3-15：查询选修了课程的学生学号(取消重复行)

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

- 注意:如果没有指定DISTINCT关键词，则缺省为ALL

查询满足条件的元组

表3.4 常用的查询条件

查 询 条 件	谓　词
比　较	=，>，<，>=，<=，!=，<>，!>，!<；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（逻辑运算）	AND，OR，NOT

比较大小

- 例3-16：查询软件学院(SE)全体学生的名单

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

- 例3-17：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;
```

- 例3-18：查询考试成绩有不及格的学生的学号

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

确定范围

- 谓词： BETWEEN … AND …

  NOT BETWEEN … AND …

- 例3-19：查询年龄在20~23岁（包括20岁和23岁）之间的学生的姓名、系别和年龄

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

- 例3-20：查询年龄不在20~23岁之间的学生姓名、系别和年龄

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

确定集合

- 谓词：IN <值表>， NOT IN <值表>
- 例3-21：查询信息系（IS）、数学系（MA）和软件学院（SE）学生的姓名和性别

```
SELECT Sname，SGender
FROM  Student
WHERE Sdept IN ( 'IS', 'MA', 'SE' );
```

- 例3-22：查询既不是信息系、数学系，也不是软件学院的学生的姓名和性别

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

字符匹配

- 谓词：[NOT] LIKE ‘<匹配串>’ [ESCAPE ‘<换码字符>’]

1）匹配串为固定字符串

例3-23：查询学号为200715121的学生的详细情况

```
SELECT *
FROM  Student
WHERE  Sno LIKE ‘200715121’;
```

等价于：

```
SELECT  *
FROM  Student
WHERE Sno = ‘200715121’;
```

字符匹配

2）匹配串为含通配符的字符串

- 例3-24：查询所有姓刘学生的姓名、学号和性别

```
SELECT Sname，Sno，SGender
FROM Student
WHERE  Sname LIKE ‘刘%’;
```

- 例3-25：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__’；
```

字符匹配

- 例3-26：查询名字中第2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：查询所有不姓刘的学生姓名

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

字符匹配

3）使用换码字符将通配符转义为普通字符

- 例3-28：查询DB_Design课程的课程号和学分

```
SELECT Cno，Ccredit
FROM Course
WHERE Cname LIKE ‘DB_Design’ ESCAPE ‘\’;
```

- 例3-29：查询以"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 ‘\’ ;
```

ESCAPE ‘\’表示‘\’为换码字符

涉及空值的查询

■ **谓词：** IS NULL 或 IS NOT NULL

- 例3-30：某些学生选修课程后没有参加考试，所以有选课记录，但没有考试成绩。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  Grade IS NULL;
```

- 例3-31：查所有有成绩的学生学号和课程号

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

- 注意:不能用‘=’代替‘IS’

多重条件查询

- 逻辑运算符：AND和 OR来联结多个查询条件
  - AND的优先级高于OR
  - 可以用括号改变优先级
- 可用来实现多种其他谓词
  - [NOT] IN
  - [NOT] BETWEEN ... AND ...

多重条件查询

例3-32：查询软件工程系年龄在20岁以下的学生姓名

```
SELECT Sname
FROM  Student
WHERE Sdept= ‘SE’  and Sage<20;
```

ORDER BY子句

- 可以按一个或多个属性列排序
- 升序：ASC；降序：DESC；缺省值为升序
- 当排序列含空值时
  - ASC：排序列为空值的元组最后显示
  - DESC：排序列为空值的元组最先显示

ORDER BY子句

- 例3-33：查询选修了3号课程的学生的学号及其成绩，查询结果按分数降序排列

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

- 例3-34：查询全体学生情况，查询结果按所在系的系号升序排列，同一系中的学生按年龄降序排列

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

SQL函数

- SQL函数可以分为标量函数和聚集函数两类。
- 标量函数的运算对象是一个记录行在某个属性上的具体取值，大致可以分为字符、数学、时间、类型转换等几种形式。
- 例如，函数LENGTH()可以计算一个字符串类型数值的长度，ABS()可以计算一个数值类型的绝对值。

聚集函数

- 计数

  COUNT（[DISTINCT|ALL] *）

  COUNT（[DISTINCT|ALL] <列名>）

- 计算总和

  SUM（[DISTINCT|ALL] <列名>）

- 计算平均值

  AVG（[DISTINCT|ALL] <列名>）

- 最大最小值

  MAX（<列名>）

  MIN（<列名>）

聚集函数

- 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;
```

- 例3-37：计算1号课程的学生平均成绩

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

聚集函数

- 例3-37：查询选修1号课程的学生最高分数

```
SELECT MAX(Grade)
FROM SC
WHER Cno=‘1’;
```

- 例3-38：查询学生200215012选修课程的总学分数

```
SELECT SUM(Ccredit)
FROM  SC，  Course
WHER Sno=‘200215012’
AND SC.Cno=Course.Cno;
```

GROUP BY子句

- 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。
- 对查询结果分组后，聚集函数将分别作用于每个组。
- 作用对象是查询的中间结果表。
- 按指定的一列或多列值分组，值相等的为一组。

GROUP BY子句

- 例3-39：求各个课程号及相应的选课人数

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

分组和聚集函数

列出每个学生的平均成绩

SNO	CNO	G
s1	c1	84
s1	c2	90
s1	c3	96
s2	c1	80
s2	c2	90
s3	c2	96
s3	c3	88

90

85

92

group by SNO

列出每门课程的平均成绩

SNO	CNO	G
s1	c1	84
s1	c2	90
s1	c3	96
s2	c1	80
s2	c2	90
s3	c2	96
s3	c3	88

90

92

92

group by CNO

HAVING子句

- HAVING子句与WHERE子句的区别：
  - WHERE子句作用于基表或视图，从中选择满足条件的元组
  - HAVING子句作用于组，从中选择满足条件的组

HAVING子句

- 例3-40：查询选修了3门以上课程的学生学号

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

连接查询

- 连接查询：同时涉及多个表的查询
- 连接条件或连接谓词：用来连接两个表的条件
- 一般格式：
  - [<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>
  - [<表名1>.]<列名1> BETWEEN [<表名2>.]<列名2> AND [<表名2>.]<列名3>
  - 连接字段：连接谓词中的列名称
  - 连接条件中的各连接字段类型必须是可比的，但名字不必是相同的

连接查询

- 等值与非等值连接查询
- 自连接
- 外连接
- 复合条件连接

等值与非等值连接查询

- 等值连接：连接运算符为=
- 例3-41：查询每个学生及其选修课程的情况

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

等值与非等值连接查询

- 自然连接：
- 例3-43：对[例3-42]用自然连接完成

```
SELECT  S.Sno，Sname，SGender，Sage，Sdept，Cno，
  Grade
FROM  Student S，SC
WHERE  S.Sno = SC.Sno；
```

自连接

- **自连接**：一个表与其自己进行连接
- 需要给表起别名以示区别
- 由于所有属性名都是同名属性，因此必须使用别名前缀
- 例3-44：查询每一门课的间接先修课（即先修课的先修课）

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

自连接

C1表（Course表）

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

自连接

C2表（Course表）

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

自连接

查询结果：

Cno	Pcno
1	7
3	5
5	6

外连接

- 外连接与普通连接的区别
  - 普通连接操作只输出满足连接条件的元组
  - 外连接操作以指定表为连接主体，将主体表中不满足连接条件的元组一并输出
- 例3-45：改写[例3-44]

```
SELECT Student.Sno，Sname，SGender，Sage，Sdept，Cno，Grade
 FROM  Student S, SC
 Where S.Sno += SC.Sno;
```

外连接

- 左外连接
  - 列出左边关系（如本例Student）中所有的元组
- 右外连接
  - 列出右边关系中所有的元组

复合条件连接

- 复合条件连接：WHERE子句中含多个连接条件
- 例3-46：查询选修2号课程且成绩在90分以上的所有学生

```
SELECT S.Sno, Sname
FROM  Student S, SC
WHERE S.Sno = SC.Sno AND /* 连接谓词*/
      SC.Cno= '2' AND SC.Grade >= 90;
                        /* 其他限定条件 */
```

复合条件连接

- 例3-47：查询每个学生的学号、姓名、选修的课程名及成绩

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

课堂提问

- 什么是SQL基本表？
- 什么是SQL视图（VIEW）？

嵌套查询2022-3-13

- 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短语的条件中的查询称为嵌套查询。

嵌套查询

- 例3-48：查询选修了2号课程的学生姓名

```
SELECT Sname                          /*外层查询/父查询*/
FROM Student
WHERE Sno IN（SELECT Sno    /*内层查询/子查询*/
      FROM SC
      WHERE Cno= ‘2’ ）；
```

嵌套查询

- 子查询的限制
  - 不能使用ORDER BY子句
- 层层嵌套方式反映了SQL语言的结构化
- 有些嵌套查询可以用连接运算替代

嵌套查询求解方法

- 不相关子查询：
  - 子查询的条件不依赖于父查询。
  - 由里向外 逐层处理：即每个子查询在上一级查询处理之前求解，子查询的结果用于建立其父查询的查找条件。

嵌套查询求解方法

- 相关子查询：
  - 子查询的查询条件依赖于父查询。
  - 首先取外层查询中表的第一个元组，根据它与内层查询相关的属性值处理内层查询，若WHERE子句返回值为真，则取此元组放入结果表。
  - 然后再取外层表的下一个元组。
  - 重复这一过程，直至外层表全部检查完为止。

带有IN谓词的子查询

- 例3-49：查询与‘刘晨’在同一个系学习的学生
- 此查询要求可以分步来完成

① 确定‘刘晨’所在系名

```
SELECT   Sdept
FROM     Student
WHERE    Sname=‘刘晨 ’;
```

假定结果为： SE

带有IN谓词的子查询

② 查找所有在SE学习的学生

```
SELECT    Sno，Sname，Sdept
FROM        Student
WHERE   Sdept= ‘SE’;
```

带有IN谓词的子查询

- 将第一步查询嵌入到第二步查询的条件中

```
SELECT Sno，Sname，Sdept
  FROM Student
  WHERE Sdept  IN
                (SELECT Sdept
                 FROM Student
                 WHERE Sname= ‘刘晨’);
```

- 此查询为不相关子查询

带有IN谓词的子查询

- 例3-50：用自连接完成[例3-49]查询

```
SELECT   S1.Sno，S1.Sname，S1.Sdept
FROM      Student S1，Student S2
WHERE   S1.Sdept = S2.Sdept
AND      S2.Sname = ‘刘晨’；
```

带有IN谓词的子查询

- 例3-51：查询选修了课程名为‘信息系统’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= ‘信息系统’
                )
         );
```

③ 最后在Student关系中取出Sno和Sname

② 然后在SC关系中找出选修了3号课程的学生学号

① 首先在Course关系中找出‘信息系统’的课程号，为3号

带有IN谓词的子查询

- 例3-52：用连接查询实现[例3-51]：

```
SELECT Sno，Sname
FROM Student S，SC，Course C
WHERE S.Sno = SC.Sno
AND   SC.Cno = C.Cno
AND   C.Cname='信息系统';
```

带有比较运算符的子查询

- 当能确切知道内层查询返回单值时，可用比较运算符（>，<，=，>=，<=，!=或< >）
- 与ANY或ALL量词配合使用

带有比较运算符的子查询

- 例3-52：假设一个学生只可能在一个系学习，并且必须属于一个系，则在[例3-45]可以用 = 代替IN ：

```
SELECT Sno，Sname，Sdept
FROM    Student
WHERE Sdept    =
                (SELECT Sdept
                 FROM    Student
                 WHERE Sname= ‘刘晨’)；
```

带有比较运算符的子查询

- 例3-53：找出每个学生超过他选修课程平均成绩的课程号

```
SELECT Sno，  Cno
FROM  SC  x
WHERE Grade >=(SELECT AVG(Grade)
               FROM  SC y
               WHERE x.sno=y.sno
               );
```

相关子查询

带有比较运算符的子查询

- 执行过程：

1. 从外层查询中取出SC的一个元组x，将元组x的Sno值（如200215121）传送给内层查询。

```
SELECT AVG(Grade)
FROM SC y
WHERE y.Sno='200215121';
```

2. 执行内层查询，得到平均值（如88），用该值代替内层查询，得到外层查询：

```
SELECT Sno,  Cno
FROM  SC x
WHERE Grade >=88;
```

带有比较运算符的子查询

3. 执行这个查询，得到

（200215121，1）

（200215121，3）

4. 外层查询取出下一个元组重复做上述1至3步骤，直到外层的SC元组全部处理完毕。结果为:

（200215121，1）

（200215121，3）

（200215122，2）

带有ANY（SOME）或ALL量词的子查询

- 谓词语义
  - ANY（SOME）：某一个值
  - ALL：所有值

带有ANY（SOME）或ALL谓词的子查询

- 需要配合使用比较运算符

> 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	大于等于子查询结果中的某个值
>= ALL	大于等于子查询结果中的所有值
<= ANY	小于等于子查询结果中的某个值
<= ALL	小于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

=ALL 等于子查询结果中的所有值（通常没有实际意义）

!=（或<>）ANY	不等于子查询结果中的某个值
!=（或<>）ALL	不等于子查询结果中的任何一个值

带有ANY（SOME）或ALL谓词的子查询

- 例3-54：查询其他系中比软件工程系某一学生年龄小的学生姓名和年龄

```
SELECT Sname，Sage
FROM   Student
WHERE Sage < ANY (SELECT  Sage
                  FROM    Student
                  WHERE Sdept = 'SE')
AND Sdept <> 'SE' ;  /*父查询块中的条件 */
```

带有ANY（SOME）或ALL谓词的子查询

- 执行过程：
  1. RDBMS执行此查询时，首先处理子查询，找出SE系中所有学生的年龄，构成一个集合(20，19)
  2. 处理父查询，找所有不是SE系且年龄小于20或19的学生

带有ANY（SOME）或ALL谓词的子查询

- 例3-55：查询其他系中比软件工程系所有学生年龄都小的学生姓名及年龄。
- 方法一：用ALL谓词

```
SELECT Sname，Sage
FROM Student
WHERE Sage < ALL (SELECT Sage
                  FROM Student
                  WHERE Sdept= 'SE')
  AND Sdept <> 'SE';
```

带有ANY（SOME）或ALL谓词的子查询

- 方法二：用聚集函数

```
SELECT Sname，Sage
FROM Student
WHERE Sage < (SELECT MIN(Sage)
              FROM Student
              WHERE Sdept= 'SE')
  AND Sdept <>'SE';
```

带有ANY（SOME）或ALL谓词的子查询

表3.5 ANY（或SOME），ALL谓词与聚集函数、IN谓词的等价转换关系

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

带有EXISTS谓词的子查询

1. EXISTS谓词
   - 存在量词∃
   - 带有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”
     - 若内层查询结果非空，则外层的WHERE子句返回真值
     - 若内层查询结果为空，则外层的WHERE子句返回假值
   - 由EXISTS引出的子查询，其目标列表达式通常都用* ，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
2. NOT EXISTS谓词
   - 若内层查询结果非空，则外层的WHERE子句返回假值
   - 若内层查询结果为空，则外层的WHERE子句返回真值

带有EXISTS谓词的子查询

- 例3-56：查询所有选修了1号课程的学生姓名
- 思路分析：
  - 本查询涉及Student和SC关系
  - 在Student中依次取每个元组的Sno值，用此值去检查SC关系
  - 若SC中存在这样的元组，其Sno值等于此Student.Sno值，并且其Cno= ‘1’，则取此Student.Sname送入结果关系

带有EXISTS谓词的子查询

- 用嵌套查询

```
SELECT Sname
FROM Student S
WHERE EXISTS(SELECT *
             FROM SC
             WHERE Sno=S.Sno
             AND Cno= '1');
```

带有EXISTS谓词的子查询

- 用连接运算

```
SELECT Sname
FROM Student S, SC
WHERE S.Sno=SC.Sno
AND SC.Cno = '1';
```

带有EXISTS谓词的子查询

- 例3-57：查询没有选修1号课程的学生姓名

```
SELECT Sname
FROM Student S
WHERE NOT EXISTS(
    SELECT *
    FROM SC
   WHERE Sno = S.Sno
   AND Cno='1');
```

带有EXISTS谓词的子查询

- 不同形式的查询间的替换
  - 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
  - 所有带IN谓词、比较运算符、ANY和ALL谓词的子查询都能用带EXISTS谓词的子查询等价替换
- 用EXISTS/NOT EXISTS实现全称量词
  - SQL语言中没有全称量词∀ （For all）
  - 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：

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

带有EXISTS谓词的子查询

- 例3-58：查询与‘刘晨’在同一个系学习的学生可以用带EXISTS谓词的子查询替换：

```
SELECT Sno，Sname，Sdept
FROM Student S1
WHERE EXISTS (SELECT *
              FROM Student S2
              WHERE S2.Sdept = S1.Sdept
                AND S2.Sname = ‘刘晨’)；
```

用EXISTS/NOT EXISTS实现逻辑蕴函

- SQL语言中没有蕴函(Implication)逻辑运算
- 可以利用谓词演算将逻辑蕴函谓词等价转换为：

$$p \rightarrow q \equiv \neg p \vee q$$

带有EXISTS谓词的子查询

- 例3-59：查询至少选修了学生200915122选修的全部课程的学生号码
- 解题思路：
  - 用逻辑蕴函表达：查询学号为x的学生，对所有的课程y，只要200915122学生选修了课程y，则x也选修了y
  - 形式化表示：

    用P表示谓词“学生200915122选修了课程y”

    用q表示谓词“学生x选修了课程y”

    则上述查询为：$(\forall y)\ p \rightarrow q$

带有EXISTS谓词的子查询

- 等价变换：

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

- 变换后语义：不存在这样的课程y，学生200915122选修了y，而学生x没有选

带有EXISTS谓词的子查询

- 用NOT EXISTS谓词表示：

```
SELECT DISTINCT Sno
FROM SC SCX
WHERE NOT EXISTS
    (SELECT *
     FROM SC SCY
     WHERE SCY.Sno = '200915122'
      AND  NOT EXISTS( SELECT *
                       FROM SC SCZ
                       WHERE  SCZ.Sno=SCX.Sno
                       AND    SCZ.Cno=SCY.Cno
                     )
);
```

带有EXISTS谓词的子查询

- 例3-60：查询选修了全部课程的学生姓名

```
SELECT Sname
FROM Student S
WHERE
NOT EXISTS(
      SELECT *
      FROM Course C
      WHERE NOT EXIST(
            SELECT *
            FROM SC
            WHERE Sno= S.Sno AND Cno= C.Cno)
                        );
```

集合查询

- 集合操作的种类
  - 并操作UNION
  - 交操作INTERSECT
  - 差操作EXCEPT
- 参加集合操作的各查询结果的列数必须相同；对应项的数据类型也必须相同

集合查询

- 例3-61：查询软件工程系的学生及年龄不大于19岁的学生
- 方法一：

```
SELECT *
FROM Student
WHERE Sdept= 'SE'
UNION ALL
SELECT *
FROM Student
WHERE Sage<=19;
```

- UNION：将多个查询结果合并时去掉重复元组。
- UNION ALL：将多个查询结果合并时保留重复元组

集合查询

- 方法二：

```
SELECT  DISTINCT  *
FROM Student
WHERE Sdept= ‘SE’  OR  Sage<=19;
```

集合查询

- 例3-62：查询选修了C001课程并且成绩高于90分的学生学号

```
SELECT Sno
FROM SC
WHERE Cno='C001'
   INTERSECT
SELECT Sno
FROM SC
WHERE Grade>=90;

SELECT Sno
FROM SC
WHERE Cno='C001' and Grade>=90;
```

集合查询

- 例3-63：查询软件学院(SE)的学生与年龄不大于19岁的学生的交集

```
SELECT *
FROM Student
WHERE Sdept='SE'
INTERSECT
SELECT *
FROM Student
WHERE Sage<=19
```

集合查询

- 实际上就是查询软件学院中年龄不大于19岁的学生:

```
SELECT *
FROM Student
WHERE Sdept= 'SE' AND  Sage<=19;
```

集合查询

- 例3-64：查询选修课程1的学生集合与选修课程2的学生集合的交集

```
SELECT Sno
FROM SC
WHERE Cno='1'
INTERSECT
SELECT Sno
FROM SC
WHERE Cno='2';
```

集合查询

- 实际上是查询既选修了课程1又选修了课程2　的学生

```
SELECT Sno
 FROM SC
 WHERE Cno='1'
 AND Sno IN (SELECT Sno
                FROM SC
                WHERE Cno='2');
```

集合查询

- 例3-65：查询软件学院(SE)的学生与年龄不大于19岁的学生的差集

```
SELECT *
FROM Student
WHERE Sdept='SE'
EXCEPT
SELECT  *
FROM Student
WHERE Sage <=19;
```

集合查询

- 实际上是查询软件学院(SE)中年龄大于19岁的学生

```
SELECT *
FROM Student
WHERE Sdept= 'SE' AND Sage>19;
```

集合查询

- 例3-66：查询至少选修了“SE－02”或“SE－04”课程的学生学号：

```
(SELECT   SNO
 FROM     SC
 WHERE    CNO ＝ ‘SE－02’ )
UNION
(SELECT   SNO
 FROM     SC
 WHERE    CNO＝‘SE－04’)
```

3.4 SQL数据操纵功能

- 数据插入
- 数据删除
- 数据修改

数据插入

- 格式

  INSERT　INTO　<表名>［(<列名>，…,<列名>)］

  VALUES　(<列值>,…,<列值>)

数据插入

- 例3-67：将数据库课程加入到Course表中：

```
INSERT  INTO  Course(CNO, Cname, Teacher)
VALUES （‘SE－06’，‘数据库系统原理’，‘李华’）;
```

- 例3-68：在SC表中增加记录（‘01055123’，‘SE－06’），成绩暂缺：

```
INSERT  INTO  SC (SNO,CNO)
VALUES  （‘01055123’，‘SE－06’）;
```

数据插入

- SQL3允许采用上述的单记录插入方式一次向表中插入多个记录，其形式如下：

```
INSERT  INTO  <表名> [(<列名>,…,<列名>)]
VALUES  (<列值>,…,<列值>), …,
        (<列值>,…,<列值>);
```

- 另一种多记录插入的方式是子查询结果插入，即将某个表中的若干个记录按照一定的查询条件作为一个查询的结果集插入到另一个表中，形式如下：

```
INSERT  INTO  <表名> [(<列名>,…,<列名>)]
<子查询>;
```

数据插入

- 例3-69： 将平均成绩高于80分的男生学号及平均成绩存入表S_Grade(SNO, Avg_Grade)中，其中SNO表示学号，Avg_Grade表示平均成绩：

```
INSERT  INTO  S_Grade(SNO, Avg_Grade)
SELECT  SNO, AVG(Grade)
FROM    SC
WHERE   SNO  IN
       (SELECT   SNO
        FROM     Student
        WHERE    Gender=’男’)
GROUP BY SNO
HAVING   AVG(Grade) >80;
```

数据删除

- **格式：**

  DELETE　FROM　〈表名〉

  [WHERE〈条件表达式〉]

- **如果该语句指定WHERE条件，则只有满足条件的那些记录才会被删除，条件可以是简单的算术比较表达式，也可以是带有子查询的复杂形式。**

数据删除

- 例3-70：删除课程表Course中的全部记录：

```
DELETE   FROM  Course;
```

- 例3-71：删除Student表中学号为'01055123'的学生记录：

```
DELETE   FROM  Student
WHERE  SNO＝'01055123';
```

数据删除

- 例3-72：将“C语言”课程成绩不存在的记录删除：

```
DELETE   FROM  SC
WHERE   (Grade  IS  NULL)
AND  CNO  IN
          (SELECT  CNO
           FROM    Course
           WHERE   Cname ＝’C语言’);
```

数据修改

- **格式：**

  UPDATE　<表名>

  SET　<列名>＝<列值表达式>[，<列名>＝<列值表达式>…]

  [WHERE <条件表达式>]；

数据修改

- **例3-73：将雇员表EMP中每名员工的工资增加100元：**

```
UPDATE EMP
SET    Salary＝Salary+100;
```

- **例3-74：将“01055071”同学选修的“SE－03”课程成绩置为NULL：**

```
UPDATE    SC
SET       Grade＝NULL
WHERE     SNO＝'01055071'
AND       CNO＝'SE－03';
```

3.5 SQL视图

- SQL视图是不存储具体数据，而仅在数据目录中存放其定义的“虚表”，它提供了一种间接访问基本表中数据的便捷方式，使用户可以更有效，更安全的访问系统中存储的相关数据。
- 视图可以通过基本表或其它视图导出，因而有着许多和基本表类似的性质。
- 本节介绍视图的使用方法及应该注意的问题，具体包括视图的定义、删除、查询、更新，以及视图的应用特点等几方面内容。

视图的定义

- 可以通过以下方式建立视图：

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

- 视图名称后指定的列名表部分不是必须的，如果不给出这些具体的列名，则视图的各列与子查询中SELECT语句所指定的列完全相同。
- 如果在子查询的SELECT语句中使用了函数或运算表达式，那么必须在视图名后给出全部的列名表，因为函数和运算表达式是不能用来表示一个列的名称的。

视图的定义

- 例3-75：建立全部男同学的视图：

```
CREATE  VIEW  S_MALE
AS  SELECT  *
      FROM    Student
      WHERE   Gender＝’男’
WITH  CHECK  OPTION;
```

视图的定义

- 例3-76：建立一个包含软件学院学生学号、选修课程数及其平均成绩的视图：

```
CREATE  VIEW  S_AG (Sno, CNT_C, AVG_G)
AS  SELECT  Sno, COUNT(Cno), AVG(Grade)
      FROM    SC
      WHERE   Cno  LIKE  'SE%'
      GROUP   BY  Sno;
```

视图的定义

- ✓ **例3-77：建立选修了课程“操作系统”的学生学号、姓名、及成绩的视图：**

```
CREATE  VIEW  S_C
AS  SELECT  S.SNO, Sname, Grade
      FROM   Student S,  SC, Course C
      WHERE   S.SNO＝SC.SNO  AND  C.CNO＝SC.CNO
                AND  Cname＝’操作系统’；
```

视图的删除

- **删除视图的语句如下：**

  DROP　VIEW　<视图名>;

- **例3-78：删除视图S_MALE：**

  DROP　VIEW　S_MALE;

视图的查询

- 由于视图中并不存储实际的物理数据记录，因此处理视图查询时，系统会首先将视图查询语句中的WHERE条件与视图定义中的WHERE条件进行有效合并，然后再转变为对基本表的查询，这一过程称为查询合并。

视图的查询

- **例3-79：在视图S_MALE上查询“李华”同学的学号及所在班级**

```
SELECT  SNO, Class
FROM S_MALE
WHERE Sname＝’李华’;
```

- **视图S_MALE建立在基本表Student上，因此该查询将转换为对基本表Student的查询，如下所示：**

```
SELECT   SNO, Class
FROM     Student
WHERE    Gender＝’男’AND  Sname＝’李华’;
```

视图的查询

- 例3-81：在视图S_AG中查询学号为’01055107’的软件学院同学的选课情况：

```
SELECT SNO, COUNT(Cno) CNT_C, AVG(Grade) AVG_G
FROM S_AG
WHERE SNO=‘01055107’;
```

- 视图S_AG建立在基本表SC上，因此该查询将转换为对基本表SC的查询，如下所示：

```
SELECT SNO,COUNT(Cno) CNT_C, AVG(Grade) AVG_G
FROM SC
WHERE Cno LIKE ‘SE%’ AND SNO=‘01055107’
GROUP BY Sno;
```

视图的更新

- 视图的更新（包括INSERT、UPDATE、DELETE等操作）是一件比较复杂的事情。

- 视图的更新最终还是会落实为对基本表的更新，但并不是所有的视图记录都能够唯一的对应到基本表中的一条记录，因此视图的更新通常会具有较大的限制。

视图的更新

- **例3-82：**在视图S_MALE中将学号为‘01055029’的学生所在班级更新为‘软件12’：

```
UPDATE   S_MALE
SET    Class＝‘软件12’
WHERE    SNO＝‘01055029’;
```

视图的更新

- **例3-83：**在视图S_MALE中插入记录（‘01055072’，‘软件13’，’王晓斌’）：

```
INSERT   INTO  S_MALE
VALUES  （‘01055072’，‘软件13’，’王晓斌’,’女’）;
```

- 该操作同样也可以正常执行，系统会将其转换为如下的基本表操作：

```
INSERT INTO  student
VALUES（‘01055072’，‘软件13’，‘王晓斌’，‘男’）;
```

视图的更新

- **例3-84：**将视图S_AG中学号为‘01055016’的同学的选课数更新为12，并将其平均成绩更新为81.7分：

```
UPDATE   S_AG
SET      CNT_C＝12, AVG_G＝81.7
WHERE    SNO＝’01055016’;
```

- **问题**:上述查询是否可以执行?

- **例3-85：**在视图S_C中插入新记录(‘01055097’，’张明’，92)：

```
INSERT    INTO   S_C
VALUES   (‘01055097’，‘张明’，92);
```

- **问题**:上述查询是否可以执行?

视图的更新

- **总结视图更新的特点及限制如下：**

（1）行列子集视图是可以更新的。一个例外情况是，如果基本表并没有为视图定义中不包含的那些属性列指定缺省值，同时这些列又不允许取值为NULL，那么这样的行列子集视图是不能执行INSERT操作的。

（2）如果视图定义中使用了DISTINCT、GROUP BY分组或者是聚集函数，那么这样的视图是不能更新的。

（3）如果视图定义中使用了多表联接或者含有嵌套查询，那么通常情况下这样的视图是不能更新的。

视图的应用

- 视图对应于数据库三层模式结构的外模式，视图这个概念的提出极大的方便了关系数据库系统的使用，具体的说，视图应用具有以下的几个优点：

（1）提高了数据逻辑独立性

（2）简化了用户应用

（3）提供了一定的数据安全保护功能

3.5 SQL数据控制功能

- 由DBMS提供统一的数据控制功能是数据库系统的特点之一。
- 数据控制亦称为数据保护，包括数据的安全性控制、完整性控制、并发控制和恢复。
- 这里主要介绍SQL的数据控制功能。

权限与角色

- 数据库上的权限主要包括两种，一种是用户级的权限，另一种是表级的权限。
- 用户级权限是指DBA可以单独授予某个用户在数据库上可进行何种操作的权限。
- SQL标准支持的是表级权限，这是指DBA可以单独控制每个基本表和视图的存取，可能会经常用到的表级权限包括：表（包括基本表和视图）的查询(SEELCT)、更新(INSERT、UPDATE、DELETE)和引用(REFERENCE)等。

权限与角色2022-3-16

- 除了可以给一个特定的用户授予权限外，SQL还支持角色的概念。
- 角色并不对应于某个具体的用户，而是对于一类具有共同特征的用户的总称，这样作的目的是为了便于管理。
- SQL中定义角色的语法如下：

```
CREATE  ROLE <角色名>
[WITH  ADMIN  {<当前用户名>|<当前角色名>}];
```

权限的授予和收回

- 将表级权限授予某个授权ID的语法格式如下：

  GRANT 〈权限〉 ON TABLE 〈表名〉

  TO 〈授权ID〉, …, 〈授权ID〉[WITH GRANT OPTION];

- 其中权限包括SELECT、UPDATE、INSERT、DELETE、REFERENCE、ALL PRIVILEGES等，ALL PRIVILEGES表示授予所有的权限，此外对于SELECT和UPDATE还可以将权限指定到具体的某些属性列上。
- 如果使用了WITH GRANT OPTION，则表明该权限可以转授。

权限的授予和收回

- **例3-80：将表SC上的SELECT和UPDATE(GRADE)特权授予用户“Wang”，同时允许该用户将权限授予其它用户：**

```
GRANT SELECT, UPDATE(GRADE) ON TABLE SC
TO Wang WITH GRANT OPTION;
```

- 收回用户某种特权的SQL语句如下：

```
REVOKE <权限> ON TABLE <表名>
FROM <授权ID>, …, <授权ID>;
```

- **例3-81：将用户Wang在SC表上的UPDATE权限收回：**

```
REVOKE UPDATE ON TABLE SC FROM Wang;
```

3.7 嵌入式SQL

- SQL语言提供了两种使用方式：
  - 一种是在终端交互式方式下使用，前面介绍的就是做为独立语言由用户在交互环境下使用的SQL语言。
  - 另一种是将SQL语言嵌入到某种高级语言如PL/1、COBOL、FORTRAN、C中使用，利用高级语言的过程性结构来弥补SQL语言在实现复杂应用方面的不足，这种方式下使用的SQL语言称为嵌入式SQL（Embedded SQL），而嵌入SQL的高级语言称为主语言或宿主语言。

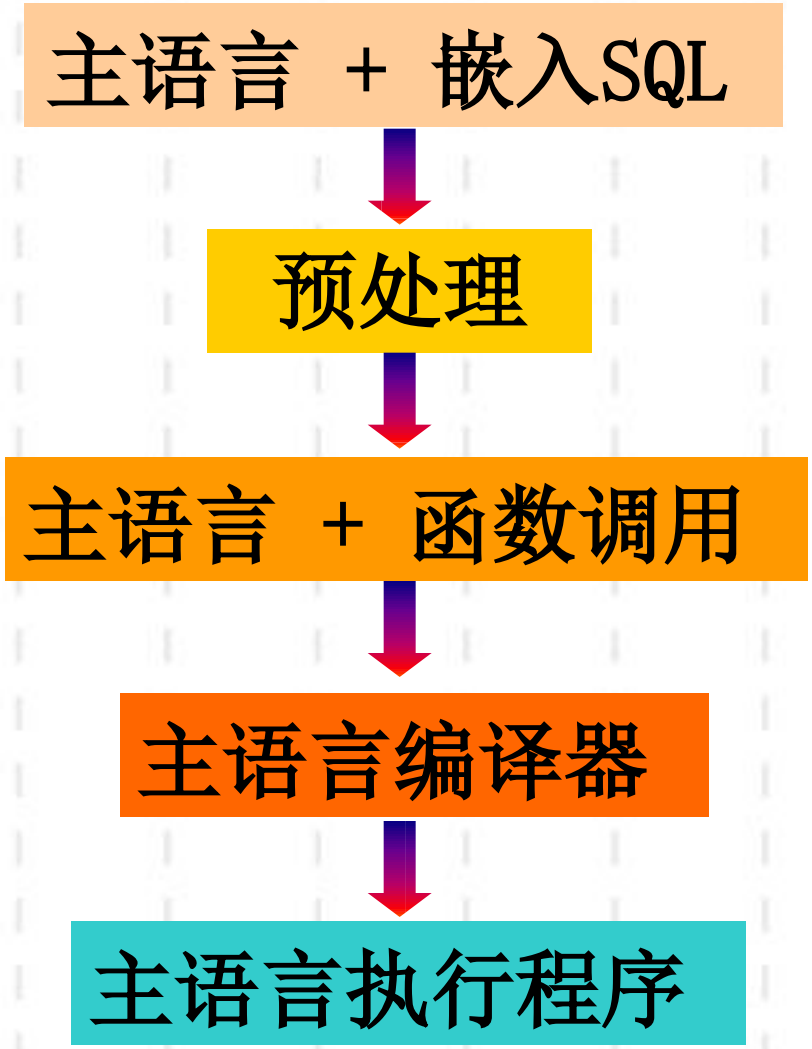
嵌入式SQL语句

- 在嵌入式SQL中，为了能够区分SQL语句与主语言语句，所有SQL语句都必须加前缀EXEC SQL。
- SQL语句的结束标志则随主语言的不同而不同，例如在PL/1和C中以分号（;）结束，在COBOL中以END-EXEC结束。
- 这样，以C或PL/1作为主语言的嵌入式SQL语句的一般形式为：
- EXEC SQL <SQL语句>;

嵌入式SQL语句

- 嵌入SQL语句根据其作用的不同，可分为可执行语句和说明性语句两类。
- 可执行语句又分为：数据定义、数据控制、数据操纵三种。
- 在宿主程序中，任何允许出现可执行的高级语言语句的地方，都可以写可执行SQL语句。
- 任何允许出现说明性高级语言语句的地方，都可以写说明性SQL语句。

嵌入式SQL的运行过程



嵌入式SQL与主语言之间的通信

- 数据库工作单元与源程序工作单元之间通信主要包括：

1. 向主语言程序传递SQL语句的执行状态信息，使主语言能够据此控制程序流程。
2. 主语言程序向SQL语句提供参数。
3. 将SQL语句查询数据库的结果交主语言程序进一步处理。

嵌入式SQL与主语言之间的通信

- 在嵌入式SQL中，向主语言程序传递SQL执行状态信息主要用SQL通信区（SQL Communication Area，简称SQLCA）实现。
- 主语言程序向SQL语句输入数据主要用主变量（host variable）实现。
- SQL语句向主语言程序输出数据主要用主变量和游标（cursor）实现。

SQL通信区

- SQL语句执行后，系统要反馈给应用程序若干信息，主要包括描述系统当前工作状态和运行环境的各种数据，这些信息将送到SQL通信区SQLCA中。
- 应用程序从SQLCA中取出这些状态信息，据此决定接下来执行的语句。
- SQLCA是一个预定义的数据结构，在应用程序中用

  EXEC SQL INCLUDE SQLCA;

  加以定义。
- SQLCA中有一个存放每次执行SQL语句后返回代码的变量SQLCODE。
- 应用程序每执行完一条SQL 语句之后都应该测试一下SQLCODE的值，以了解该SQL语句执行情况并做相应处理。

SQL通信区

- 如果SQLCODE等于预定义的常量SUCCESS，则表示SQL语句成功，否则表示错误代码。
- 例如,在执行删除语句DELETE后，根据不同的执行情况，SQLCA中有下列不同的信息：
  - 违反数据保护规则，操作拒绝
  - 没有满足条件的行，一行也没有删除
  - 成功删除，并有删除的行数（SQLCODE=SUCCESS）
  - 无条件删除警告信息
  - 由于各种原因，执行出错

主变量

- 嵌入式SQL语句中可以使用主语言的程序变量来输入或输出数据。我们把在SQL语句中使用的主语言程序变量简称为主变量。
- 主变量根据其作用的不同，分为输入主变量和输出主变量。
- 输入主变量由应用程序对其赋值，SQL语句引用。
- 输出主变量由SQL语句对其赋值或设置状态信息，返回给应用程序。

主变量

- 一个主变量有可能既是输入主变量又是输出主变量。
- 利用输入主变量，可以指定向数据库中插入的数据，可以将数据库中的数据修改为指定值，可以指定执行的操作，可以指定WHERE子句或HAVING子句中的条件。
- 利用输出主变量，可以得到SQL语句的结果数据和状态。

主变量

- 一个主变量可以附带一个任选的指示变量（Indicator Variable）。
- 所谓指示变量是一个整型(Integer)变量，用来“指示”所指主变量的值或条件。
- 输入主变量可以利用指示变量赋空值，输出主变量可以利用指示变量检测出是否空值，值是否被截断。

主变量

- 使用主变量及指示变量的方法是，所有主变量和指示变量必须在SQL语句

  EXEC SQL BEGIN DECLARE SECTION;

  与

  EXEC SQL END DECLARE SECTION;

- 之间进行说明

主变量

- 主变量被说明之后，就可以在SQL语句中任何能够使用表达式的地方出现，为了与数据库对象名（表名、视图名、列名等）区别，SQL语句中的主变量名前要加冒号（:）作为标志。
- 同样，SQL语句中的指示变量前也必须加冒号标志，并且要紧跟在所指主变量之后。
- 而在SQL语句之外，主变量和指示变量均可以直接引用，不必加冒号。

游标

- SQL语言与主语言具有不同数据处理方式。
- SQL语言是面向集合的，一条SQL语句原则上可以产生或处理多条记录。
- 主语言是面向记录的，一组主变量一次只能存放一条记录。
- 所以仅使用主变量并不能完全满足SQL语句向应用程序输出数据的要求，为此嵌入式SQL引入了游标的概念，用游标来协调这两种不同的处理方式。

游标

- 游标是系统为用户开设的一个数据缓冲区，存放SELECT语句执行后返回的结果集。
- 每个游标区都有一个名字,用户可以用SQL语句逐一从游标中获取记录，并赋给主变量，交由主语言程序进一步处理。
- 按照是否使用游标，可以将嵌入式SQL语句区分为。

不使用游标的嵌入式SQL语句

1)说明性语句

- 交互式SQL中没有说明性语句，说明性语句是专为在嵌入式SQL中说明主变量等而设置的，主要有两条语句：

  EXEC SQL BEGIN DECLARE SECTION;

  和

  EXEC SQL END DECLARE SECTION;

- 两条语句必须配对出现，相当于一个括号，两条语句中间是主变量的说明。

不使用游标的嵌入式SQL语句

2）数据定义语句，这一类语句不会返回结果集，所以不需要使用游标，例如，定义一个基本表或删除一个基本表的语句等，删除基本表SC可以写成如下形式：

```
EXEC SQL DROP TABLE SC;
```

- 数据定义语句中不允许使用主变量，例如下列语句是错误的：

```
EXEC SQL DROP TABLE :table_name;
```

不使用游标的嵌入式SQL语句

3）数据控制语句，这一类SQL语句用于用户及角色的管理，授予用户（角色）权限或收回其权限，不返回结果集，不需要使用游标；

不用游标的SQL语句

4)查询结果为单记录的SELECT语句

- 在嵌入式SQL中，查询结果为单记录的SELECT语句需要用INTO子句指定查询结果的存放地点。该语句的一般格式为：

```
EXEC SQL
SELECT [ALL|DISTINCT]
                        <目标列表达式>[,<目标列表达式>]...
INTO <主变量>[<指示变量>][,<主变量>[<指示变量>]]...
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]];
```

- 该语句对交互式SELECT语句的扩充就是多了一个INTO子句，把从数据库中找到的符合条件的记录，放到INTO子句指出的主变量中去，其他子句的含义不变。

不使用游标的嵌入式SQL语句

- 例3-82:根据学生号码查询学生信息。假设已将要查询的学生的学号赋给了主变量givensno：

```
srtcpy(givensno, ‘200801689”);
EXEC SQL SELECT Sno , Sname, SGender, Sage, Sdept
    INTO  :Hsno :isno, :Hname :isname, :HGender,:Hage,:Hdept
    FROM  Student
    WHERE Sno=:givensno;
```

- 上面的SELECT语句中Hsno, Hname, HGender, Hage, Hdept和givensno均是主变量，并均已在前面的程序中说明过了。

不用游标的SQL语句

5）非CURRENT形式的UPDATE语句

- 在UPDATE语句中，SET子句和WHERE子句中均可以使用主变量，其中SET子句中还可以使用指示变量
- 例5： 将全体学生001号课程的考试成绩增加若干分，设增加的分数已赋给主变量raise：

```
int raise;
raise = 10;
EXEC SQL UPDATE SC
        SET Grade=Grade+:raise
        WHERE Cno = ‘001”;
```

- 该操作实际上是一个集合操作，DBMS会修改所有学生的001号课程的Grade属性列。

不用游标的SQL语句

- 例3-83： 修改某个学生001号课程的成绩。假设该学生的学号已赋给主变量givensno，修改后的成绩已赋给主变量newgrade：

```
EXEC SQL UPDATE SC
        SET Grade=:newgrade
       WHERE Sno=:givensno; and CNO= ‘001’ ;
```

不用游标的SQL语句

6)非CURRENT形式的DELETE语句

- DELETE语句的WHERE子句中可以使用主变量指定删除条件。
- 例3-84：某个学生退学了，现要将有关他的所有选课记录删除掉，假设该学生的姓名已赋给主变量stdname：

```
EXEC SQL DELETE FROM SC
        WHERE Sno = ( SELECT Sno
                      FROM Student
                      WHERE
Sname=:stdname);
```

不用游标的SQL语句

- 另一种等价实现方法为：

```
EXEC SQL DELETE   FROM SC
              WHERE :stdname =
                   (SELECT Sname
                      FROM Student
                     WHERE   Studnet.Sno =
        SC.sno);
```

- 显然第一种方法更直接，从而也更高效些。如果该学生选修了多门课程，执行上面的语句时，DBMS会自动执行集合操作，即把他选修的所有课程都删除掉

不用游标的SQL语句

7) INSERT语句

- INSERT语句的VALUES子句中可以使用主变量和指示变量。
- 例3-85:某个学生新选修了某门课程，将有关记录插入SC表中。假设学生的学号已赋给主变量stdno，课程号已赋给主变量couno

```
gradeid = -1;
EXEC SQL INSERT
    INTO SC(Sno, Cno, Grade)
    VALUES(:stdno, :couno, :gr :gradeid);
```

- 由于该学生刚选修课程，尚未考试，因此成绩列为空，所以本例中用指示变量指示相应的主变量为空值。

使用游标的SQL语句

- 一般情况下，SELECT语句查询结果都是多条记录，而高级语言一次只能处理一条记录，因此需要以游标机制作为桥梁，将多条记录一次一条送至宿主程序处理，从而把对集合的操作转换为对单个记录的处理。

使用游标的SQL语句

- 使用游标的步骤：

1)说明游标：用DECLARE语句为一条SELECT语句定义游标
DECLARE语句的一般形式为：

EXEC SQL DECLARE <游标名>
CURSOR FOR <SELECT语句>;

- 其中SELECT语句可以是简单查询，也可以是复杂的连接查询和嵌套查询。
- 定义游标仅仅是一条说明性语句，这时DBMS并不执行SELECT指定的查询操作。

使用游标的SQL语句

2） 打开游标:用OPEN语句将上面定义的游标打开
OPEN语句的一般形式为：

EXEC SQL OPEN <游标名>[USING …];

- 打开游标实际上是执行相应的SELECT语句，把所有满足查询条件的记录从指定表取到缓冲区中，这时游标处于活动状态，指针指向查询结果集第0条记录。

使用游标的SQL语句

3)推进游标指针并取当前记录:用FETCH语句把游标指针向前推进一条记录，同时将缓冲区中的当前记录取出来出来送至主变量供主语言进一步处理，FETCH语句的一般形式为：

```
EXEC SQL FETCH <游标名>
INTO <主变量>[<指示变量>][,<主变量>[<指示变量>]]...;
```

- 其中主变量必须与SELECT语句中的目标列表达式具有一一对应关系。
- 为进一步方便用户处理数据，现在许多关系数据库管理系统对FETCH语句做了扩充，允许用户向任意方向以任意步长移动游标指针，而不仅仅是把游标指针向前推进一行。

使用游标的SQL语句

4) 关闭游标:用CLOSE语句关闭游标，释放结果集占用的缓冲区及其他资源，CLOSE语句的一般形式为：

EXEC SQL CLOSE <游标名>;

游标被关闭后，就不再和原来的查询结果集相联系。但被关闭的游标可以再次被打开，与新的查询结果相联系。

使用游标的嵌入式SQL语句

- 例3-86:查询学生选课信息并在终端上显示，学生学号由用户输入：

```
EXEC SQL INCLUDE SQLCA;                    /*定义SQL通信区*/
EXEC SQL BEGIN DECLARE SECTION;
…  /* 说明宿主变量 UID,PWD,Hsno, Hcno, Hgrade, GradeID */
EXEC SQL END DECLARE SECTION;
srtcpy(UID,” Zhang”);
srtcpy(PWD,” tiger”);
EXEC SQL CONNECT(:UID , :PWD);
printf(“请输入学号：\n”);
gets(Hsno);
```

使用游标的嵌入式SQL语句

```
...
EXEC SQL  DECLARE  SC_CURSOR  CURSOR FOR /*定义游标*/
        SELECT  CNO, GRADE
        FROM    SC
        WHERE  SNO=:Hsno;
EXEC SQL OPEN  SC_CURSOR ;                    /*打开游标*/
```

使用游标的嵌入式SQL语句

```
While(1){
   EXEC SQL FETCH SC_CURSOR
         INTO :Hcno, :Hgrade :GradeID;    /*读取数据*/
     if ( sqlca.sqlcode <> SUCCESS)  break;
                                          /*如果出错则中断程序*/
     ......                               /*显示查询结果*/
   }
   EXEC SQL CLOSE SC_CURSOR;              /*关闭游标*/
```

使用游标的嵌入式SQL语句

- 更新语句中使用游标一般会带有CURRENT语句说明更新记录的位置。
- 例3-87:查询课程信息，根据用户需要修改某些元组的TEACHER字段：

```
EXEC SQL INCLUDE SQLCA;                    /*定义SQL通信区*/
EXEC SQL BEGIN DECLARE SECTION;
  /* 说明宿主变量 dept, deptname, Hcno, HCname, HTeacher,
     NewTeacher */
  …
EXEC SQL END DECLARE SECTION;
  …
```

使用游标的嵌入式SQL语句

```
printf(“请输入系名代号：\n”);
gets(dept);
deptname = “\’ ”;
strcat(deptname,dept);
strcat(deptname, “\%\’ ”);
EXEC SQL  DECLARE  C_CURSOR  CURSOR FOR  /*定义游标*/
 SELECT  CNO, Cname, Teacher
 FROM   Course
 WHERE  CNO  LIKE  :deptname
 FOR UPDATE OF TEACHER ;
```

使用游标的嵌入式SQL语句

```
EXEC SQL OPEN C_CURSOR;                                    /*打开游标*/
 While(1)
 { EXEC SQL FETCH C_CURSOR
        INTO :Hcno, :Hcname, :HTeacher; /*读取数据*/
  if (SQLCA.SQLCODE <> SUCCESS) break; /*如出错则中断程序*/
  printf("%s, %s, %s", Hcno, Hcname, HTeacher);
  printf("确定要修改Teacher属性吗?");
  scanf("%c",&flag);
```

使用游标的嵌入式SQL语句

```
if (flag == 'y' || flag == 'Y')
    { printf("请输入新的Teacher属性: ");
      scanf("%s", &NewTeacher);
      EXEC SQL UPDATE Course /*修改当前记录的TEACHER属性*/
      SET TEACHER=:NewTeacher
      WHERE CURRENT OF C_CURSOR;
    }
  ...
}
EXEC SQL CLOSE C_CURSOR;
```

动态SQL

- 前面节中介绍的嵌入式SQL语句为编程提供了一定的灵活性，使用户可以在程序运行过程中根据实际需要输入WHERE子句或HAVING子句中某些变量的值。
- 这些SQL语句的共同特点是，语句中主变量的个数与数据类型在预编译时都是确定的，只有主变量的值是程序运行过程中动态输入的，我们称这类嵌入式SQL语句为静态SQL语句。

动态SQL

- 静态SQL语句提供的编程灵活性在许多情况下仍显得不足，有时候我们需要编写更为通用的程序。
- 例如，查询学生选课关系SC，任课教师想查选修某门课程的所有学生的学号及其成绩，班主任想查某个学生选修的所有课程的课程号及相应成绩，学生想查某个学生选修某门课程的成绩，也就是说查询条件是不确定的，要查询的属性列也是不确定的，这时无法用一条静态SQL语句实现。

动态SQL

- 实际上，如果在预编译时下列信息:
  - SQL语句正文
  - 主变量个数
  - 主变量的数据类型
  - SQL语句中引用的数据库对象(例如列、索引、基本表、视图等)。
- 不能确定，我们就必须使用动态SQL技术。

动态SQL

- 动态SQL方法允许在程序运行过程中临时“组装”SQL语句，主要有三种形式：
  1）语句可变
  2）条件可变
  3）数据库对象、查询条件均可变

语句可变

- 可接收完整的SQL语句，即允许用户在程序运行时临时输入完整的SQL语句。

条件可变

- 对于非查询语句，条件子句有一定的可变性。例如删除学生选课记录，既可以是因某门课临时取消，需要删除有关该课程的所有选课记录，也可以是因为某个学生退学，需要删除该学生的所有选课记录。
- 对于查询语句，SELECT子句是确定的，即语句的输出是确定的，其他子句（如WHERE子句、HAVING短语）有一定的可变性。
- 例如，查询学生人数，可以是查某个系的学生总数，查某个性别的学生人数，查某个年龄段的学生人数，查某个系某个年龄段的学生人数等等，这时SELECT子句的目标列表达式是确定的（COUNT(*)），但WHERE子句的条件是不确定的。

数据库对象、查询条件均可变

- 对于查询语句，SELECT子句中的列名、FROM子句中的表名或视图名、WHERE子句和HAVING短语中的条件等等均可由用户临时构造，即语句的输入和输出可能都是不确定的。例如我们前面查询学生选课关系SC的例子。
- 对于非查询语句，涉及的数据库对象及条件也是可变的。
- 这几种动态形式几乎可覆盖所有的可变要求。

动态SQL语句

- 为了实现上述三种可变形式，SQL提供了相应的语句，包括:EXECUTE IMMEDIATE、PREPARE、EXECUTE、DESCRIBE等。
- 动态SQL预备语句：

  EXEC SQL PREPARE〈语句名〉

  FROM〈共享变量或字符串〉
- 这里共享变量或字符串的值应是一个完整的SQL语句。这个语句可以在程序运行时由用户输入或组合起来。
- 此时，这个语句并不执行。

动态SQL语句

- 动态SQL执行语句

  EXEC SQL EXECUTE〈语句名〉

  动态SQL语句使用时，还可以有两点改进：

(1) 当预备语句中组合而成的SQL语句只需执行一次时，那么预备语句和执行语句可合并成一个语句：

  EXEC SQL EXECUTE IMMEDIATE〈共享变量或字符串〉

(2) 当预备语句中组合而成的SQL语句的条件值尚缺时，可以在执行语句中用USING短语补上：

EXEC SQL EXECUTE < 语句名 > USING〈主变量〉

动态SQL语句

- 例3-88:下面两个C语言的程序段说明了动态SQL语句的使用技术

①
```
EXEC SQL BEGIN DECLARE SECTION;
char[120] query;
EXEC SQL END DECLARE SECTION;
scanf（" %s"，query）；   /* 从键盘输入一个SQL语句 */
//strcpy(query,"delete from student where sno=?");
EXEC SQL PREPARE que FROM :query;
EXEC SQL EXECUTE que using :givenson;
…
```

- 这个程序段表示从键盘输入一个SQL语句到字符数组中；字符指针query指向字符串的第1个字符，如果执行语句只做一次，那么程序段最后两个语句可合并成一个语句：

```
EXEC SQL EXECUTE IMMEDIATE :query;
```

动态SQL语句

```
BEGIN DECLARE SECTION;
 char *query =
      " UPDATE sc SET grade = grade * 1.1 WHERE cno =? " ;
 char[5] cno = "C4" ;
END DECLARE SECTION;
  EXEC SQL PREPARE dynprog FROM :query;

  EXEC SQL EXECUTE dynprog USING :cno;
```

- 这里第一个char语句表示用户组合成一个SQL语句，但有一个值（课程号）还不能确定，因此用“？”表示
- 第二个语句是动态SQL预备语句
- 第三个语句（char语句）表示取到了课程号值
- 第四个语句是动态SQL执行语句，“？”值到共享变量cno中取

作业

- 3.2
- 3.3
- 3.4
- 3.5
- 3.7