

第3章 关系数据库标准语言—— SQL语言



3.1 SQL概述

- » SQL (Structured Query Language结构化查询语言) 是关系数据库的标准语言。目前，关系数据库管理系统都采用SQL语言作为数据库语言。
- » SQL-86: the first standard by ANSI --- American national standard institute, ISO standard.
- » SQL-89: minor revision.
- » **SQL-92: currently supported by most commercial RDBMS.**
- » SQL-99: a major extension of SQL-92, partly supported now.



SQL语言概述

SQL (Structured Query Language) 语言是一种基于关系代数的结构化查询语言，**SQL**是一个通用的、功能极强的关系数据库语言，具有查询功能、数据定义功能、数据操纵、数据控制功能。目前，关系数据库管理系统都采用**SQL**语言作为数据库语言。

SQL语言具有以下特点：

- 综合统一。
- 是一种非过程语言。
- 是一种面向集合的语言。
- 既可独立使用，又可嵌入到宿主语言中使用。



1, SQL的特点-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



2, SQL的特点-高度非过程化

- 非关系数据模型的数据操纵语言“面向过程”，必须指定存取路径。
- SQL只要提出“做什么”，无须了解存取路径。
- 存取路径的选择以及SQL的操作过程由系统自动完成。



3, SQL的特点-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



4, SQL的特点-

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



5, SQL的特点-语言简洁, 易学易用

- SQL功能极强, 完成核心功能只用了9个动词。

表 3.2 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



-
- (1) 了解SQL的特点，掌握SQL的四大功能及使用方法。
 - (2) 掌握数据定义语句、数据查询语句、数据操纵语句的格式和功能。
 - (3) 重点掌握数据查询功能及其使用，掌握完整的SELECT语句的格式和含义。能针对实际问题，熟练地设计简单的SELECT语句。
 - (4) 会在SQL Server环境中编辑、运行SQL语句。



SQL组成

- 数据定义语言（Data Definition Language），用于创建，修改，删除数据库对象，SQL关键字包括CREATE，ALTER，DROP
- 数据查询语言（Data Query Language），用于检索数据，主要SQL关键字为SELECT
- 数据操纵语言（Data Manipulation Language），用于增，删，改数据库对象中的数据，主要SQL关键字INSERT，UPDATE和REVOKE
- 数据控制语言（Data Control Language），用于控制用户对数据库的访问，主要SQL关键字GRANT，DENY 和REVOKE
- 其它语言要素，事务控制，程序化语言



SQL 基本概念:

1、基本表

一个关系对应一个基本表。基本表是独立存在的表，不是由其他的表导出的表。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，索引也存放在存储文件中。存储文件的逻辑结构组成了关系的内模式。存储文件的物理结构对最终用户是隐蔽的。

2、视图

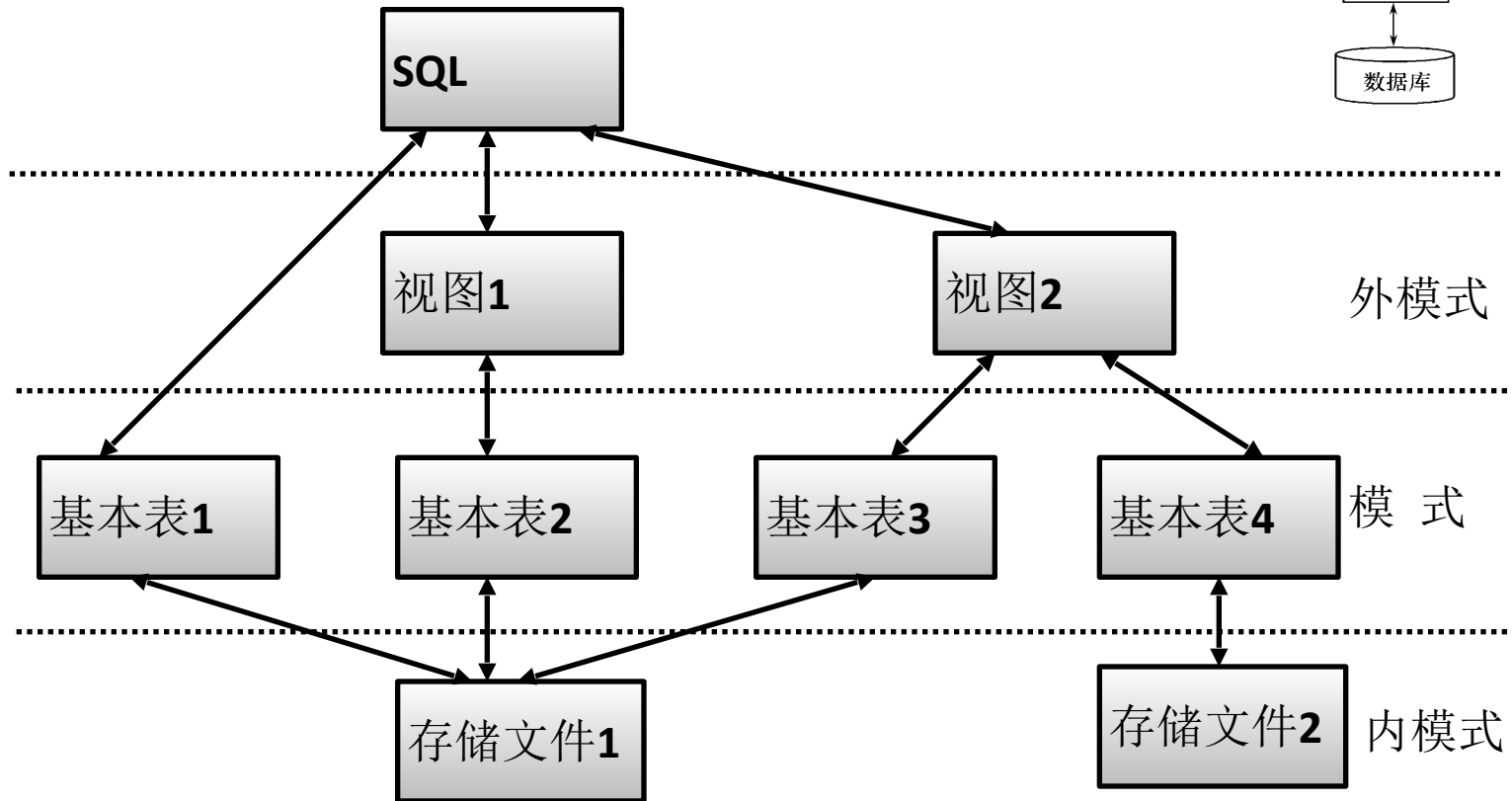
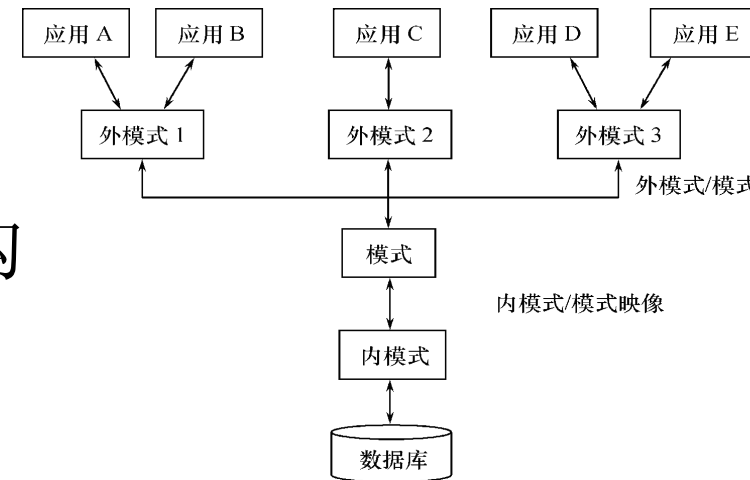
视图是从一个或几个基本表导出的表，是一个虚表。它本身不独立存在于数据库中，数据库中只存放视图的定义而不存放视图对应的数据，这些数据仍存放在导出视图的基本表中。当基本表中的数据发生变化时，从视图中查询出来的数据也随之改变。

视图对于用户来说，就象一个窗口，透过视图用户可以看到数据库中自己感兴趣的内容。



SQL的基本概念

SQL支持关系数据库三级模式结构



3.1 数据定义

❖SQL的数据定义功能:

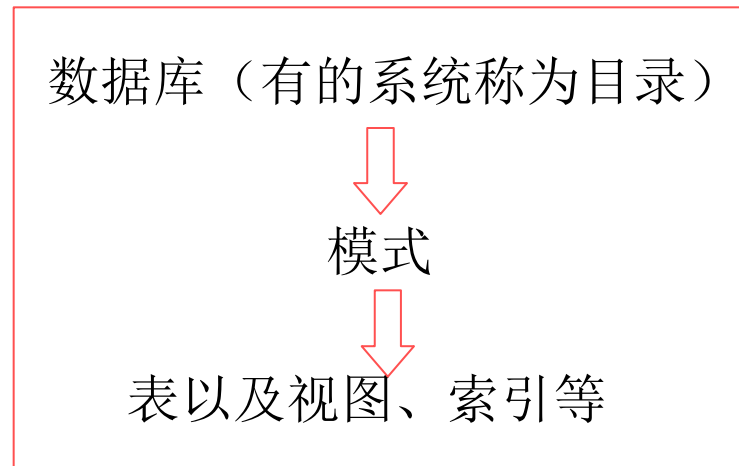
- 模式定义
- 表定义
- 视图和索引的定义

表 3.3 SQL 的数据定义语句

操 作 对 象	操 作 方 式		
	创 建	删 除	修 改
模式	CREATE SCHEMA	DROP SCHEMA	
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	
索引	CREATE INDEX	DROP INDEX	ALTER INDEX



数据库-命名空间

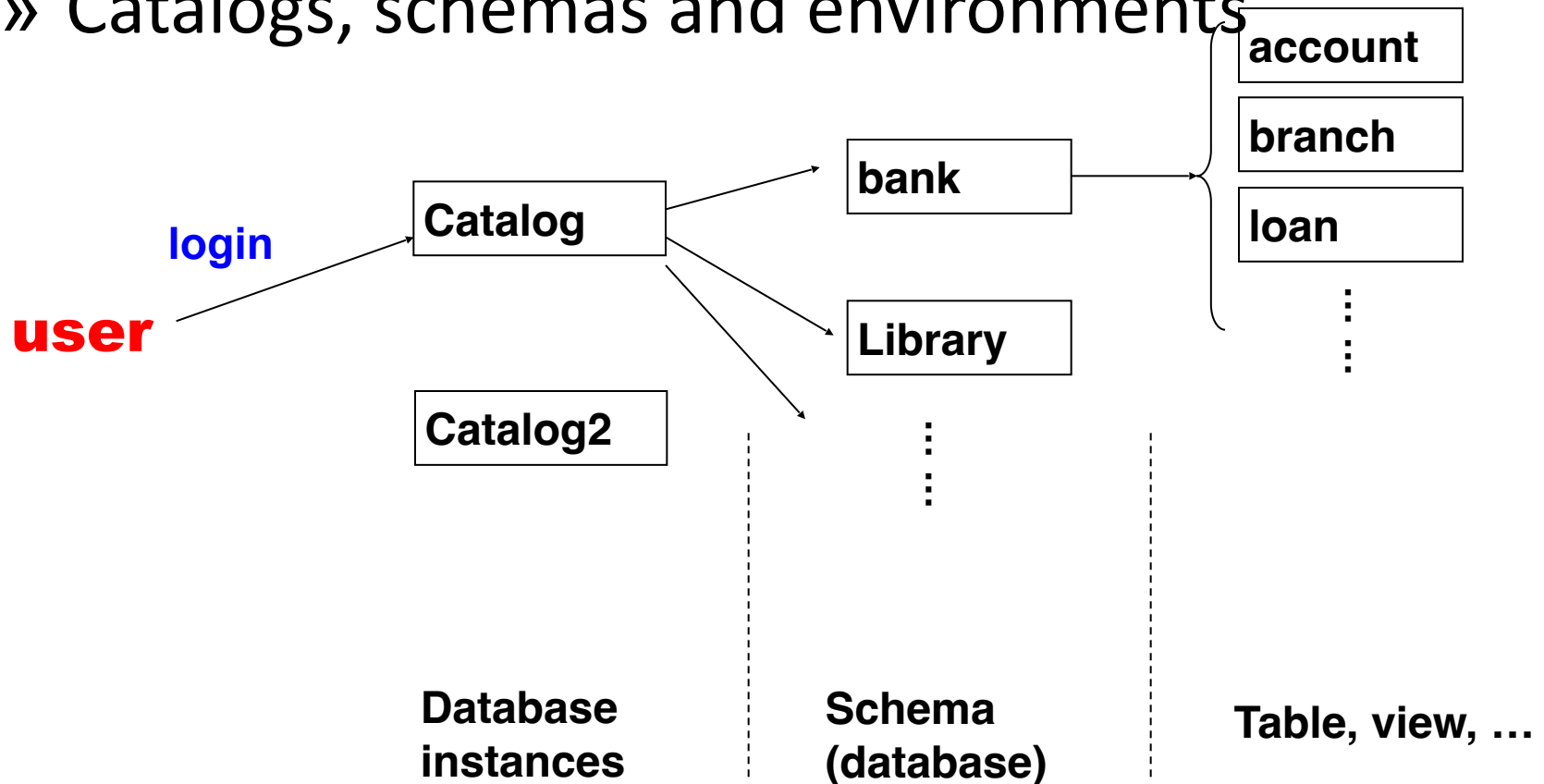


- 现代关系数据库管理系统提供了一个层次化的数据库对象命名机制
 - 一个关系数据库管理系统的实例（Instance）中可以建立多个数据库
 - 一个数据库中 can 建立多个模式
 - 一个模式下通常包括多个表、视图和索引等数据库对象



SQL Data Types and Schemas

» Catalogs, schemas and environments



1. 定义模式

CREATE SCHEMA <模式名> AUTHORIZATION <用户名>;

[例3.1] 为用户WANG定义一个学生-课程模式S-T

CREATE SCHEMA “S-T” AUTHORIZATION WANG;

CREATE SCHEMA AUTHORIZATION WANG;

该语句没有指定<模式名>，<模式名>隐含为<用户名>



定义模式（续）

- 定义模式实际上定义了一个命名空间。
- 在这个空间中可以定义该模式包含的数据库对象，例如基本表、视图、索引等。
- 在CREATE SCHEMA中可以接受CREATE TABLE, CREATE VIEW和GRANT子句。

CREATE SCHEMA <模式名> AUTHORIZATION <用户名>[<表定义子句>|<视图定义子句>|<授权定义子句>];



定义模式（续）

[例3.3]为用户ZHANG创建了一个模式TEST，并且在其中定义一个表TAB1

```
CREATE SCHEMA TEST AUTHORIZATION ZHANG
CREATE TABLE TAB1 ( COL1 SMALLINT,
                     COL2 INT,
                     COL3 CHAR(20),
                     COL4 NUMERIC(10,3),
                     COL5 DECIMAL(5,2)
                     );
```



2. 删除模式

- DROP SCHEMA <模式名> <CASCADE|RESTRICT>

- CASCADE (级联)

- 删除模式的同时把该模式中所有的数据库对象全部删除

- RESTRICT (限制)

- 如果该模式中定义了下属的数据库对象（如表、视图等），则拒绝该删除语句的执行。
- 仅当该模式中没有任何下属的对象时才能执行。



删除模式（续）

[例3.4] DROP SCHEMA ZHANG CASCADE;

- ✓ 删除模式ZHANG
- ✓ 同时该模式中定义的表TAB1也被删除



3.1.2 关系表

表是数据库的重要组成部分，下面主要介绍利用SQL如何建立、修改和删除数据表。

1，表的定义

- 建立用户数据表(**create table**): 定义属性的数据类型; 定义数据表的约束
- 模式与表

2，表的修改表(**alter table**)

- **add**
- **alter**
- **drop**

3，表的删除(**drop table**)



(1) 建立用户数据表

用**CREATE TABLE**语句定义数据表，语句格式：

CREATE TABLE <表名>

(<列定义>[{, <列定义> | <表约束>}])

其中：

❁ <表名>是合法标识符，最多可有**128**个字符，不允许重名。

❁ <列定义>格式： <列名> <数据类型> [**DEFAULT** 默认值] [{<列约束>}]

❁ **DEFAULT**,若某字段设置有默认值，则当该字段未被输入数据时，以该默认值自动填入该字段。

❁ **SQL**语句不区分大小写,但逗号必须是半角字符。



数据类型:

关系数据库中每个关系表中的一列来自同一个域，属于同一种数据类型。当定义数据表时，需要为表中的每列（字段）设置一种数据类型，用来指定字段所存放的数据属于何种数据类型。

◆ 选用哪种数据类型

- 1, 取值范围
- 2, 要做哪些运算



常用数据类型（续）

数据类型	含义
CHAR(<i>n</i>), CHARACTER(<i>n</i>)	长度为 n 的定长字符串
VARCHAR(<i>n</i>), CHARACTERVARYING(<i>n</i>)	最大长度为 n 的变长字符串
CLOB	字符串大对象
BLOB	二进制大对象
INT, INTEGER	长整数（ 4 字节）
SMALLINT	短整数（ 2 字节）
BIGINT	大整数（ 8 字节）
NUMERIC(<i>p</i>, <i>d</i>)	定点数，由 p 位数字（不包括符号、小数点）组成，小数后面有 d 位数字
DECIMAL(<i>p</i>, <i>d</i>), DEC(<i>p</i>, <i>d</i>)	同 NUMERIC
REAL	取决于机器精度的单精度浮点数
DOUBLE PRECISION	取决于机器精度的双精度浮点数
FLOAT(<i>n</i>)	可选精度的浮点数，精度至少为 n 位数字
BOOLEAN	逻辑布尔量
DATE	日期，包含年、月、日，格式为 YYYY-MM-DD
TIME	时间，包含一日的时、分、秒，格式为 HH:MM:SS
TIMESTAMP	时间戳类型
INTERVAL	时间间隔类型



[例]以下命令建立一个名称为**student1**的学生表，其结构为学号（整型），姓名（**8**字节文本），性别（**2**字节文本），出生日期（日期型），班号（**5**字节文本）。

```
CREATE TABLE student1  
  (学号 integer primary key,  
   姓名  varchar(8),  
   性别  varchar(2) default “女”,  
   出生日期 date,  
   班号  varchar(5));
```



(2)定义数据表的约束

SQL Server提供的数据库完整性机制主要包括：约束（**Constraint**）、默认（**Default**）、规则（**Rule**）、触发器（**Trigger**）、存储过程（**Stored Procedure**）等。

约束是**SQL Server**自动强制数据库完整性的方式。对数据表的约束包括：

列约束：是对某一个特定列的约束，包含在列定义中，直接跟在该列的其他定义之后，用空格分隔，不必指定列名。

表约束：与列约束相互独立，不包括在列定义中，通常用于对多个列一起进行约束，与列定义用逗号分隔，定义表约束时必须指出要约束的列的名称。



完整性约束的基本语法格式：

[CONSTRAINT<约束名>]<约束类型>

■ **约束名**：约束不指定名称时，系统会给定一个名称。

■ **约束类型**：在定义完整性约束时，必须指定完整性约束的类型。

在**SQL Server**中可以定义五种类型的完整性约束：

- (1) **NOT NULL**约束
- (2) **UNIQUE**(唯一约束)
- (3) **PRIMARY KEY**(主键约束)
- (4) **FOREIGN KEY**(外键约束)
- (5) **CHECK**约束



(1) NOT NULL约束

NULL值不是0也不是空白，更不是填入字符串“NULL”，而是表示不知道、不确定、没有数据的意思。当某一字段一定要输入值时，可设置为NOT NULL。

该约束只能用于定义列约束。其语法格式：

[CONSTRAINT<约束名>][NOT NULL]

[例]以下命令建立一个名称为**student1**的学生表，其结构为学号（整型，非空），姓名（8字节文本），性别（2字节文本），出生日期（日期型），班号（5字节文本）。对学号字段进行非空约束。

CREATE TABLE student1

(学号 integer CONSTRAINT S_CONS NOT NULL,
姓名 varchar(8),
性别 varchar(2),
出生日期 date,
班号 varchar(5));

注：S_CONS为约束名。当学号为空时，系统给出出错信息。当无NOT NULL约束时，系统缺省为NULL。



UNIQUE约束既可用于列约束，也可用于表约束。

UNIQUE用于定义列约束时，其语法格式：

[CONSTRAINT<约束名>] UNIQUE

[例]建立学生表**student1**，定义姓名为唯一键。

```
CREATE TABLE student1  
  (学号 integer NOT NULL,  
   姓名  varchar(8) CONSTRAINT SN_UNIQ UNIQUE,  
   性别  varchar(2),  
   出生日期 date,  
   班号  varchar(5));
```



(2) UNIQUE(唯一约束)

用于指明基本表在某一系列或多个列的组合上的取值必须唯一。定义了UNIQUE约束的那些列称为唯一键，系统自动为唯一键建立唯一索引，从而保证了唯一键的唯一性。唯一键允许为空，但系统为保证其唯一性，最多只可以出现一个NULL值。

在建立UNIQUE约束时，要考虑以下因素：

- 使用UNIQUE约束的字段允许为NULL值；
- 一个表中可以允许有多个UNIQUE约束；
- 可以把UNIQUE约束定义在多个字段上；
- UNIQUE约束用于强制在指定字段上创建一个UNIQUE索引。



(3) PRIMARY KEY(主键约束)

PRIMARY KEY约束用于定义基本表的主键，起唯一标识作用，其值不能为NULL，也不能重复，以此来保证实体的完整性。

PRIMARY KEY约束与**UNIQUE**约束类似，通过建立唯一索引来保证基本表在主键列取值的唯一性，二者区别：

- 在一个基本表中只能定义一个**PRIMARY KEY**约束，但可以定义多个**UNIQUE**约束。

- 对于指定为**PRIMARY KEY**列的一个列或多个列的组合，其中任何一个列都不能出现NULL值，而对于**UNIQUE**所约束的唯一键，允许NULL。

- 不能为同一个列或一组列既定义**UNIQUE**约束，又定义**PRIMARY KEY**约束。



PRIMARY KEY约束既可用于列约束，也可用于表约束。

PRIMARY KEY用于定义列约束时，其语法格式：

CONSTRAINT <约束名> PRIMARY KEY

[例]建立学生表**student1**，定义学号为主键。

CREATE TABLE student1

(学号 integer CONSTRAINT S_Prim PRIMARY KEY,
姓名 varchar(8),
性别 varchar(2),
出生日期 date,
班号 varchar(5));



当将某些列的组合定义为主键时，其语法格式：

[CONSTRAINT <约束名>] PRIMARY KEY(<列名>[{, <列名>}])

[例]建立学生选课表**SC**，定义学号和课程号为主键。

```
CREATE TABLE SC  
(Sno varchar(5),  
Cno varchar(5) ,  
Score float,  
CONSTRAINT SC_Prim PRIMARY KEY(Sno,Cno)) ;
```



(4) FOREIGN KEY(外键约束)

FOREIGN KEY约束用于指定一个列或一组列作为外键，其中，包含外部键的表称为从表，包含外部键所引用的主键的表称为主表。系统保证从表在外部键上的取值是主表主键的值，或者为空，以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：

[CONSTRAINT <约束名>] FOREIGN KEY REFERENCES <主表名> (<列名>[{, <列名>}])



[例]建立学生选课表**SC**，定义**Sno,Cno**为**SC**的外部键。

CREATE TABLE SC

**(Sno varchar(5) CONSTRAINT S_Fore FOREIGN KEY
REFERENCES S(Sno),**

**Cno varchar(5) CONSTRAINT C_Fore FOREIGN KEY
REFERENCES C(Cno),**

Score float,

CONSTRAINT SC_Prim PRIMARY KEY(Sno,Cno)) ;



(5) CHECK约束

CHECK约束用来检查字段值所允许的范围，如一个字段只能输入整数，限定在0-100，以此来保证域的完整性。

在建立**CHECK**时，需要考虑以下几个因素：

- 一个表中可以定义多个**CHECK**约束；
- 每个字段只能定义一个**CHECK**约束；
- 在多个字段上定义的**CHECK**约束必须为表约束；
- 当执行**INSERT**，**UPDATE**语句时，**CHECK**约束将验证数据。

CHECK约束既可用于列约束，也可用于表约束，其语法：

[CONSTRAINT <约束名>] CHECK(<条件>)



[例]建立学生选课表SC，定义Score的取值范围为0-100。

```
CREATE TABLE SC  
    (Sno varchar(5) NOT NULL CONSTRAINT S_Fore FOREIGN KEY  
REFERENCES S(Sno),  
    Cno varchar(5) NOT NULL CONSTRAINT C_Fore FOREIGN KEY  
REFERENCES C(Cno),  
    Score float CONSTRAINT Score_Chk CHECK(Score>=0 AND  
Score<=100),  
    PRIMARY KEY(Sno,Cno)) ;
```



定义基本表

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>]);

<表名>：所要定义的基本表的名字

<列名>：组成该表的各个属性（列）

<列级完整性约束条件>：涉及相应属性列的完整性约束条件

<表级完整性约束条件>：涉及一个或多个属性列的完整性约束条件

如果完整性约束条件涉及到该表的多个属性列，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。



(3)模式与表

- 每一个基本表都属于某一个模式
- 一个模式包含多个基本表
- 定义基本表所属模式
 - 方法一：在表名中明显地给出模式名

```
Create table "S-T".Student(.....);    /*模式名为 S-T*/  
Create table "S-T".Course(.....);  
Create table "S-T".SC(.....);
```

- 方法二：在创建模式语句中同时创建表
- 方法三：设置所属的模式



模式与表（续）

- 创建基本表（其他数据库对象也一样）时，若没有指定模式，系统根据搜索路径来确定该对象所属的模式
- 关系数据库管理系统会使用模式列表中第一个存在的模式作为数据库对象的模式名
- 若搜索路径中的模式名都不存在，系统将给出错误



模式与表（续）

- 数据库管理员用户可以设置搜索路径，然后定义基本表

```
SET search_path TO "S-T",PUBLIC;
```

```
Create table Student(.....);
```

结果建立了S-T.Student基本表。

关系数据库管理系统发现搜索路径中第一个模式名S-T，就把该模式作为基本表Student所属的模式。



2. 修改数据表-用**ALTER TABLE ...** 修改数据表结构

(1) **ADD**方式：用于增加新列或完整性约束，定义方式同**CREATE TABLE**,语法格式：

ALTER TABLE <表名>

ADD <列定义>|<完整性约束定义>

[例]在**student1**表添加一个“住址”列，可用下列语句

ALTER TABLE student1 **ADD** 住址 varchar(20);

应注意，新增加的列不能定义为“NOT NULL”。数据表在增加一列后，原有记录在新增加的列上的值都被定义为空（NULL）。

[例]在**SC**表中增加完整性约束定义，定义**Score**的取值范围为**0-100**。

ALTER TABLE SC
ADD

CONSTRAINT Score_Chk CHECK(Score>=0 AND Score<=100);



(2) **ALTER**方式-用于修改某些列，语法格式：

ALTER TABLE <表名>

ALTER COLUMN <列名> <数据类型>

[例]把**student1**表姓名列加宽到**10**个字符。

ALTER TABLE student1

ALTER COLUMN 姓名 varchar(10);



(3) **DROP**方式-用于删除完整性约束/列，语法格式：

ALTER **TABLE** <表名>

DROP **CONSTRAINT** <约束名>

[例] 删除student1表中的主键。

```
ALTER   TABLE   student1   DROP   CONSTRAINT  
S_Prim ;
```

DROP COLUMN子句用于删除表中的列

- 如果指定了**CASCADE**短语，则自动删除引用了该列的其他对象
- 如果指定了**RESTRICT**短语，则如果该列被其他对象引用，关系数据库管理系统将拒绝删除该列



4. 删除数据表

用**DROP TABLE...**删除数据表，语句格式：

DROP TABLE <表名>

- **RESTRICT**: 删除表是有限制的。
- **CASCADE**: 删除该表没有限制。删除一个已经存在的表，在表上定义的所有视图和索引也一起被删除。



删除基本表（续）

[例3.12] 若表上建有视图，选择**RESTRICT**时表不能删除;选择**CASCADE**时可以删除表，视图也自动删除。

```
CREATE VIEW IS_Student
```

```
AS
```

```
    SELECT Sno,Sname,Sage
```

```
    FROM  Student
```

```
    WHERE Sdept='IS';
```

```
DROP TABLE Student RESTRICT;
```

--**ERROR**: cannot drop table Student because other objects depend on it



删除基本表（续）

[例3.12续]如果选择**CASCADE**时可以删除表，视图也自动
被删除

```
DROP TABLE Student CASCADE;
```

```
--NOTICE: drop cascades to view IS_Student
```

```
SELECT * FROM IS_Student;
```

```
--ERROR: relation " IS_Student " does not exist
```



删除基本表（续）

DROP TABLE时，**SQL2011** 与 3个**RDBMS**的处理策略比较

序号	标准及主流数据库 的处理方式 依赖基本表 的对象	SQL2011		Kingbase ES		Oracle 12c		MS SQL Server 2012
		R	C	R	C		C	
1	索引	无规定		√	√	√	√	√
2	视图	×	√	×	√	√ 保留	√ 保留	√ 保留
3	DEFAULT, PRIMARY KEY, CHECK （只含该表的列） NOT NULL 等约束	√	√	√	√	√	√	√
4	外码 FOREIGN KEY	×	√	×	√	×	√	×
5	触发器 TRIGGER	×	√	×	√	√	√	√
6	函数或存储过程	×	√	√ 保留	√ 保留	√ 保留	√ 保留	√ 保留

R表示**RESTRICT**，C表示**CASCADE**

‘×’表示不能删除基本表，‘√’表示能删除基本表，‘保留’表示删除基本表后，还保留依赖对象

China university of
Mining and Technology-Beijing



数据定义

1 模式的定义与删除

2 基本表的定义、删除与修改

3 索引的建立与删除

4 数据字典



3.3.3 索引的建立与删除

- 建立索引的目的：加快查询速度
- 关系数据库管理系统中常见索引：
 - 顺序文件上的索引
 - B+树索引
 - 散列（hash）索引
 - 位图索引
- 特点：
 - 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



索引

- 谁可以建立索引
 - 数据库管理员 或 表的属主（即建立表的人）
- 谁维护索引
 - 关系数据库管理系统自动完成
- 使用索引
 - 关系数据库管理系统自动选择合适的索引作为存取路径，用户不必也不能显式地选择索引



1. 建立索引

- 语句格式

CREATE [UNIQUE] [CLUSTER] INDEX <索引名>

ON <表名>(<列名>[<次序>][,<列名>[<次序>]]...);

- <表名>：要建索引的基本表的名字
- 索引：可以建立在表的一列或多列上，各列名之间用逗号分隔
- <次序>：指定索引值的排列次序，升序：ASC，降序：DESC。缺省值：ASC
- UNIQUE：此索引的每一个索引值只对应唯一的数据记录
- CLUSTER：表示要建立的索引是聚簇索引



建立索引（续）

[例3.13] 为学生-课程数据库中的Student, Course, SC三个表建立索引。

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

```
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. 修改索引

- **ALTER INDEX** <旧索引名> **RENAME TO** <新索引名>
 - [例3.14] 将SC表的SCno索引名改为SCSno
ALTER INDEX SCno RENAME TO SCSno;



3. 删除索引

- **DROP INDEX** <索引名>;

删除索引时，系统会从数据字典中删去有关该索引的描述。

[例3.15] 删除Student表的Stusname索引

```
DROP INDEX Stusname;
```



数据定义

1 模式的定义与删除

2 基本表的定义、删除与修改

3 索引的建立与删除

4 数据字典



数据字典

- 数据字典是关系数据库管理系统内部的一组系统表，它记录了数据库中所有定义信息：
 - 关系模式定义
 - 视图定义
 - 索引定义
 - 完整性约束定义
 - 各类用户对数据库的操作权限
 - 统计信息等
- 关系数据库管理系统在执行SQL的数据定义语句时，实际上就是在更新数据字典表中的相应信息。



在数据库school1中定义 student（学生表）、teacher（教师表）、course（课程表）和score（成绩表）4个表。

student: 表

字段名称	
no	
name	
sex	
birthday	

teacher: 表

字段名称	
no	
name	
sex	
birthday	
prof	
depart	

course: 表

字段名称	
cno	
cname	
tno	

score: 表

字段名称	数据类型	说明
no	文本	学号
cno	文本	课程编号
degree	数字	分数

常规

字段大小

格式

输入法模式

输入掩码

标题

默认值

有效性规则

有效性文本

必填字段

允许空字符串

索引

Unicode 压缩

10

输入法开启

学号

是

否

无

是

字段属性

字段名

64 个

格)。

查看不

自



在**student**（学生表）、**teacher**（教师表）、**course**（课程表）和**score**（成绩表）4个表中输入相应的数据。

学号	姓名	性别	出生日期	
101	李军	男	76-02-20	95
103	陆君	男	74-06-03	95
105	匡明	男	75-10-02	95
107	王丽	女	76-01-23	95
108	曾华	男	77-09-01	95
109	王芳	女	75-02-10	95
米				

记录: 1 共有记录数: 6

学号	课程编号	degree
101	3-105	64
101	6-166	85
103	3-105	92
103	3-245	86
105	3-105	88
105	3-245	75
107	3-105	91
107	6-166	79
108	3-105	78
108	6-166	81
109	3-105	76
109	3-245	68
米		0

记录: 1 共有记录数: 12

教师编号	姓名	性别	出生日期	职称	单位
100	李刚	男	68-01-02	副教授	数学系
804	李诚	男	58-12-02	副教授	计算机系
825	王萍	女	72-05-05	助教	计算机系
831	刘冰	女	77-08-14	助教	电子工程系
856	张旭	男	69-03-12	讲师	电子工程系
米					

记录: 1 共有记录数: 5

课程编号	课程名	任课教师编号
3-105	计算机导论	825
3-245	操作系统	804
6-166	数字电路	856
9-888	高等数学	100
米		

记录: 1 共有记录数: 4



第三章 关系数据库标准语言SQL

3.1 SQL概述

3.2 数据定义

3.3 数据查询

3.4 数据更新

3.6 空值的处理

3.7 视图

3.8 小结



3.3 SQL数据查询功能

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，为用户提供了极其简便、快捷和灵活多样的各种查询功能。

SELECT查询语句可实现：

- 可对表中数据进行投影、条件、排序、汇总查询；
- 单表查询、多表的连接查询；



3.3.1 投影查询-选择表中的若干列

在关系代数中最常用的式子是下列表达式：

$$\pi_{A_1, \dots, A_n}(R_1)$$

针对上述表达式，SQL为此设计了**SELECT—FROM**句型，可以选择查询表中的任意列，语句格式：

SELECT 目标列表达式
FROM 表名

“目标列表达式”指出要检索的列的名称，可以为一个列或多个列，当为多个列时，中间用“,”分隔。**FROM**子句指出从什么表中提取数据，如果从多个表中提取数据，表名之间用“,”分隔开。



[例]输出student表中所有记录的名字、性别和班级。

```
SELECT name,sex,class  
FROM student;
```

也可写为：

```
SELECT name,sex,class FROM student;  
select name,sex,class from student;
```



选择表中的若干列（续）

- 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<列表名>指定为 *

[例3. 18] 查询全体学生的详细记录

```
SELECT *
```

```
FROM Student;
```



查询经过计算的值（续）

SELECT子句的<目标列表达式>不仅可以为表中的属性列，也可以是表达式

[例3.19] 查全体学生的姓名及其出生年份。

```
SELECT Sname,2014-Sage          /*假设当时为2014年*/  
FROM Student;
```

输出结果：

Sname	2014-Sage
李勇	1994
刘晨	1995
王敏	1996
张立	1995



查询经过计算的值（续）

[例3.20] 查询全体学生的姓名、出生年份和所在的院系，要求用小写字母表示系名。

```
SELECT Sname,'Year of Birth: ',2014-Sage,LOWER(Sdept)
FROM Student;
```

输出结果：

Sname	'Year of Birth: '	2014-Sage	LOWER(Sdept)
-------	-------------------	-----------	--------------

李勇	Year of Birth:	1994	cs
刘晨	Year of Birth:	1995	cs
王敏	Year of Birth:	1996	ma
张立	Year of Birth:	1995	is



查询经过计算的值（续）

- 使用列**别名**改变查询结果的列标题：

```
SELECT Sname NAME, 'Year of Birth:' BIRTH,  
2014-Sage BIRTHDAY, LOWER(Sdept) DEPARTMENT  
FROM Student;
```

输出结果：

NAME	BIRTH	BIRTHDAY	DEPARTMENT
李勇	Year of Birth:	1994	cs
刘晨	Year of Birth:	1995	cs
王敏	Year of Birth:	1996	ma
张立	Year of Birth:	1995	is



3.3.2 选择查询-选择表中的若干元组

选择查询就是指定查询条件，只从表提取或显示满足该查询条件的元组。为了选择表中满足查询条件的某些行，可以在SELECT—FROM句型中增加WHERE子句，其格式：

SELECT 列表名
FROM 表名
WHERE 查询条件

其中WHERE子句的查询条件是一个逻辑表达式，它是由多个关系表达式通过逻辑运算符

(.AND.、.OR.、.NOT.) 连接而成的，而关系表达式中可以使用的关系运算符有：

=,<>,>,>=,<,<=,BETWEEN...AND,IN,LIKE.



表3.6 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



① 比较大小

[例3.22] 查询计算机科学系全体学生的名单。

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

[例3.23] 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;
```

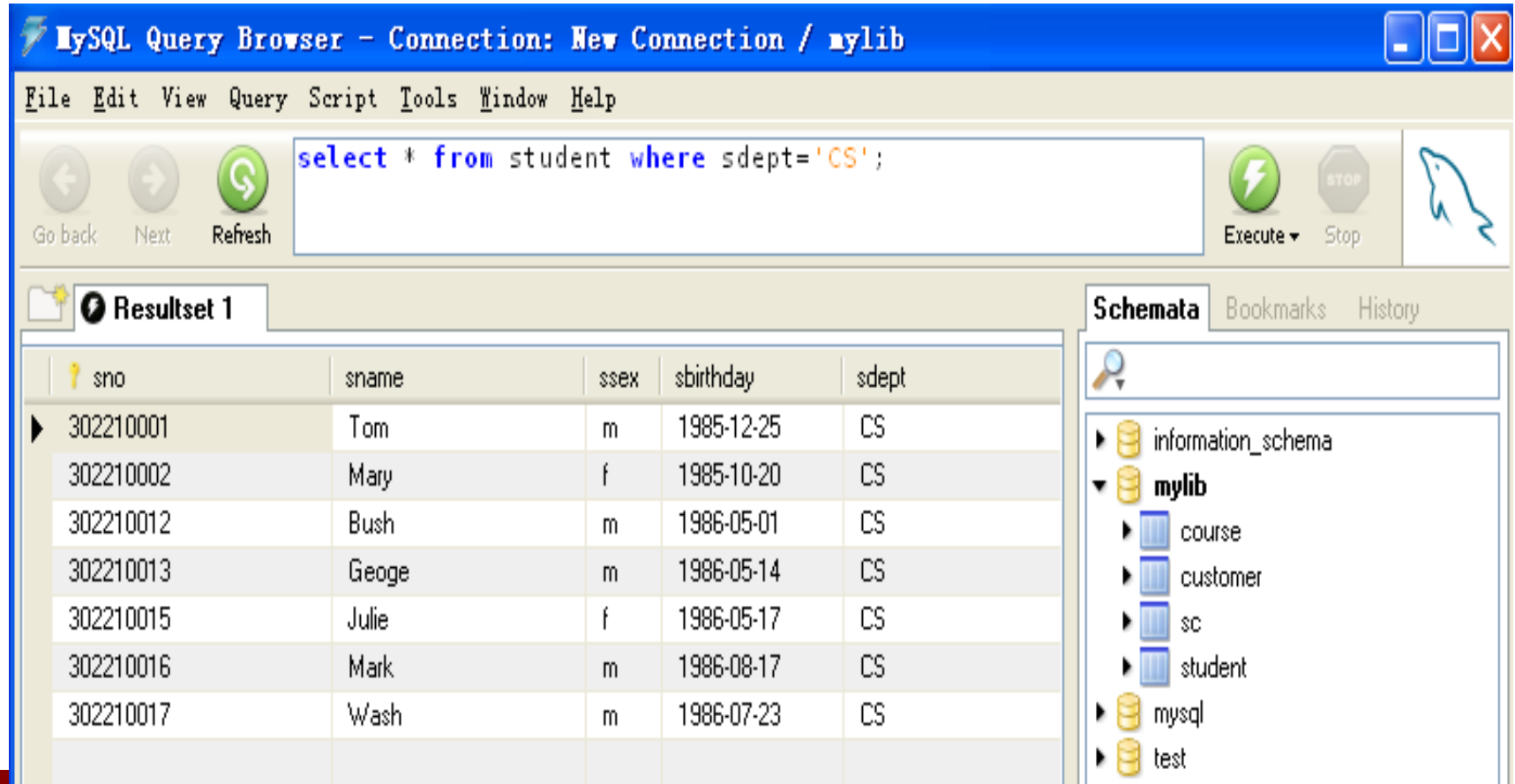
[例3.24] 查询考试成绩有不及格的学生的学号。

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



Basic Structure of Select (Cont.)

❑ Example 1: use **MySQL Query Browser**



The screenshot shows the MySQL Query Browser interface. The title bar reads "MySQL Query Browser - Connection: New Connection / mylib". The menu bar includes File, Edit, View, Query, Script, Tools, Window, and Help. Below the menu bar is a toolbar with "Go back", "Next", "Refresh", "Execute", and "Stop" buttons. The query text area contains the SQL statement: `select * from student where sdept='CS';`. Below the query area, the "Resultset 1" tab is active, displaying a table with the following data:

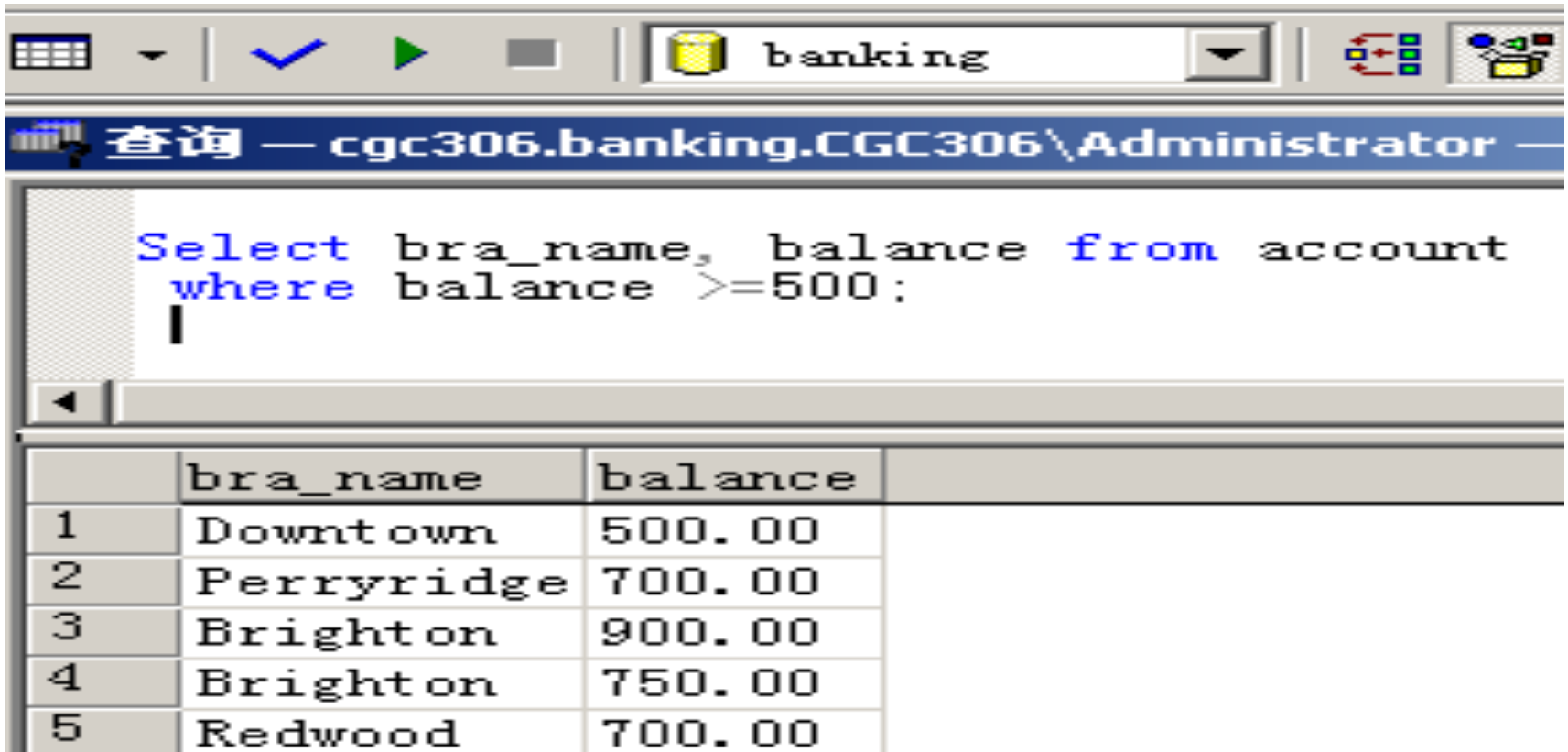
sno	sname	ssex	sbirthday	sdept
302210001	Tom	m	1985-12-25	CS
302210002	Mary	f	1985-10-20	CS
302210012	Bush	m	1986-05-01	CS
302210013	Geoge	m	1986-05-14	CS
302210015	Julie	f	1986-05-17	CS
302210016	Mark	m	1986-08-17	CS
302210017	Wash	m	1986-07-23	CS

On the right side, the "Schemata" tab is active, showing a tree view of the database structure. The "mylib" database is expanded, showing tables: course, customer, sc, and student. Other databases listed include information_schema, mysql, and test.



Basic Structure of Select

- ❑ Example 2: use Query Analyzer (查询分析器) of SQL Server



The screenshot shows the SQL Server Query Analyzer interface. The title bar indicates the database is 'banking'. The query window contains the following SQL statement:

```
Select bra_name, balance from account  
where balance >=500;
```

Below the query window, the results are displayed in a table with 3 columns: an index, 'bra_name', and 'balance'.

	bra_name	balance
1	Downtown	500.00
2	Perryridge	700.00
3	Brighton	900.00
4	Brighton	750.00
5	Redwood	700.00



② 确定范围

- 谓词: **BETWEEN ... AND ...**
NOT BETWEEN ... AND ...

[例3. 25] 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. 26] 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;
```



③ 确定集合

- 谓词: **IN** <值表>, **NOT IN** <值表>

[例3.27]查询计算机科学系（CS）、数学系（MA）和信息系（IS）学生的姓名和性别。

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

[例3.28]查询既不是计算机科学系、数学系，也不是信息系的学生姓名和性别。

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



④ 字符匹配

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

<匹配串>可以是一个完整的字符串，也可以含有通配符**%**和**_**

- **% (百分号)** 代表任意长度（长度可以为0）的字符串
 - 例如a%b表示以a开头，以b结尾的任意长度的字符串
- **_ (下横线)** 代表任意单个字符。
 - 例如a_b表示以a开头，以b结尾的长度为3的任意字符串



— 匹配串为固定字符串

[例3. 29] 查询学号为201215121的学生的详细情况。

```
SELECT *  
FROM Student  
WHERE Sno LIKE '201215121';
```

等价于：

```
SELECT *  
FROM Student  
WHERE Sno = '201215121';
```



■ 匹配串为含通配符的字符串

[例3.30] 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3.31] 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3.34] 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3.35] 查询以"DB_"开头，且倒数第3个字符为 i 的课程的具体情况。

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

ESCAPE '\ ' 表示“ \ ”为换码字符



⑤ 涉及空值的查询

❖ 谓词: **IS NULL 或 IS NOT NULL**

■ “IS” 不能用 “=” 代替

[例3.36] 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3.37] 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;
```



⑥多重条件查询

- 逻辑运算符：**AND**和 **OR**来连接多个查询条件
 - AND的优先级高于OR
 - 可以用括号改变优先级

[例3.38] 查询计算机系年龄在20岁以下的学生姓名。

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



[例3.27] 查询计算机科学系（CS）、数学系（MA）和信息系（IS）学生的姓名和性别。

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

可改写为：

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



3.3.3 排序查询-ORDER BY子句

排序查询是指查询的结果按照某一系列或某几列字段的升序/降序排列。

SELECT 列表名 FROM 表名 WHERE 查询条件 ORDER BY 列表名

注意：ORDER BY 子句中，列名后加ASC表示按升序排序（默认），列名后加DESC表示按降序排序。

➤大部分情况下ORDER BY 子句必须是SELECT语句中的最后一个子句。



[例]以班号class降序输出student表的所有记录。

```
SELECT *  
FROM student  
ORDER BY class DESC;
```

[例]按cno升序、degree降序显示score表的所有记录。

```
SELECT *  
FROM SCORE  
ORDER BY cno ASC, degree DESC
```



3.3.4 使用合计函数-聚集函数

SQL合计函数主要用于实现数据统计等功能，并将统计结果显示出来。SQL提供的SQL合计函数：

函数名	功 能
AVG	计算一个数值型列的平均值
COUNT	计算指定列中选择的项数，count(*)统计查询输出的行数
MIN	计算指定列中的最小值
MAX	计算指定列中的最大值
SUM	计算指定列中的数值总和



[例]输出student表中95031班的学生人数。

```
SELECT count(*)  
FROM student  
WHERE class='95031' ;
```

[例]输出score表中的最高分。

```
SELECT MAX(degree) FROM SCORE
```

[例]输出编号为3-105的课程的平均分。

```
SELECT AVG(degree)  
FROM SCORE  
WHERE cno='3-105'
```



GROUP BY子句：可以在SELECT—FROM —WHERE句型或SELECT—FROM句型后面加入GROUP BY子句，将查询结果按某一列或多列值分组，值相等的记录为一组。语句格式：

SELECT 列表名 **FROM** 表名 **WHERE** 查询条件 **GROUP BY** 列表名

对查询结果分组的目的是为了**细化合计函数的作用对象**。通常合计函数的作用范围是**整个查询结果**（即满足WHERE子句指定条件的所有记录），当增加GROUP BY子句后，则合计函数的作用范围变为**每组的所有记录**。



GROUP BY子句（续）

【例3.46】 求各个课程号及相应的选课人数。

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

查询结果：

Cno	COUNT(Sno)
1	22
2	34
3	44
4	33
5	48



HAVING短语：如果分组后还要求按一定的条件对这些组进行筛选，最终只输出满足条件的组，则在GROUP BY子句后增加HAVING短语指定筛选条件，HAVING短语总是紧跟在GROUP BY子句后面。语句格式：

➤ **SELECT** 列表名 **FROM** 表名 **WHERE** 查询条件
GROUP BY 列表名 **HAVING** 筛选条件

WHERE子句与HAVING短语的区别在于作用对象不同：

- WHERE子句作用于基本表，从中选出满足条件的元组；
- HAVING短语作用于组，从多组中选择满足条件的组。



[例]输出score表中至少有3名学生选修的课程号以3开头的课程的平均分。

```
SELECT cno AS 课程号 , avg(degree)
FROM score
WHERE cno LIKE '3%'
GROUP BY cno
HAVING count(*)>=3 ;
```

➤显示选课人数:

```
SELECT cno AS 课程号 , avg(degree), count(*)
FROM score WHERE cno LIKE '3%'
GROUP BY cno
HAVING count(*)>=3 ;
```



最常用最基本的SELECT查询语句的格式如下：

```
SELECT 目标列表表达式  
FROM 表名序列  
[ WHERE 条件表达式 ]  
[ GROUP BY 列名序列2 [HAVING 组条件表达式] ]  
[ ORDER BY 列名1 [ASC | DESC ]  
    [,列名2 [ASC | DESC ] ... ]
```

说明：**SQL**语句不区分大小写, 逗号必须是半角字符。
各子句按上述顺序在**SELECT**语句中出现。



其中：

SELECT——指出要查询的项目，通常指列名或表达式，是必需的。

FROM——指明被查询的表，是必需的。

WHERE——说明查询条件，是任选的。

GROUP BY——将表按列的值进行分组，列名之间用逗号分开，是任选的。

ORDER BY——将查询结果排序，是任选的。



SELECT语句的执行顺序：

- 如果有WHERE子句，从FROM子句指定的表名序列中找出满足WHERE子句条件表达式的元组；
- 如果有GROUP子句，则将结果表按列名序列2的值分组，该列名序列2的属性值相等的记录分为一组。
- 通常会在每组中作用聚集函数。如果GROUP子句带有HAVING短语，则只有满足组条件表达式的组才输出。
- 按SELECT子句中的列名序列1，选出元组中的属性值形成结果表。
- 如果有ORDER BY子句，则结果还要按列名的升序/降序排序。

SELECT 列名序列1
FROM 表名序列

[WHERE 条件表达式]

[GROUP BY 列名序列2 [HAVING 组条件表达式]]

[ORDER BY 列名1 [ASC | DESC]

[,列名2 [ASC | DESC] ...]



[例]输出score表中成绩在60~80之间的所有记录。

语句: **SELECT * FROM score**

WHERE degree BETWEEN 60 AND 80

► 关系运算符**BETWEEN...AND...**表示在两者之间

语句: **SELECT * FROM score**

WHERE degree>=60 AND degree<=80

查询结果:

no	no	degree
101	3-105	64
105	3-245	75
107	6-166	79
108	3-105	78
109	3-245	68
109	3-105	76



在条件表达式中，字段为文本型的值一定要用单引号或双引号括起来（半角）。

[例]输出student表中95031班或性别为女的姓名、性别、出生日期、班号，并按class升序、birthday降序排列。

语句： **SELECT** name,sex,birthday,class **FROM** student
WHERE class='95031' OR sex='女'
ORDER BY class **asc** , birthday **desc**;
查询结果：

name	sex	birthday	class
匡明	男	02-Oct-75	95031
王芳	女	10-Feb-75	95031
陆君	男	03-Jun-74	95031
王丽	女	23-Jan-76	95033



SQL语句中聚集函数的使用：

[例]输出student表中95031班的学生人数。

语句： **SELECT** count(*)

FROM student **WHERE** class='95031' ;

语句： **SELECT** count(no)

FROM student **WHERE** class='95031' ;



[例]输出score中学生各科平均分在80分以上的学生的学号和学生平均分，并按平均分的降序排列。

语句: **SELECT** no , avg(degree)
FROM score **GROUP BY** no
HAVING avg(degree)>80
ORDER BY avg(degree) **DESC**;

SCORE表

no	no	degree
101	3-105	64
101	6-166	85
103	3-105	92
103	3-245	86
105	3-105	88
105	3-245	75
107	3-105	91
107	6-166	79
108	3-105	78
108	6-166	81
109	3-105	76
109	3-245	68

查询结果:

no	平均分
103	89
107	85
105	81.5



3.3.4 多表查询

在数据查询中，经常涉及到提取两个或多个表的数据

1. 连接查询
2. 嵌套查询
3. 集合查询
4. 基于派生表的查询



在连接查询中，在**WHERE**子句中指定连接列，并给出连接条件，还可给出查询的条件。在**FROM**子句中指定要连接的表。

最基本的连接查询语句格式如下：

SELECT 列名**1**，列名**2**，...

FROM 表**1**，表**2**，...

WHERE 连接条件 [**and** 查询条件]



在连接查询的多个表中，通常具有公共列（如 **no**），两个表的连接通常在公共列上进行。

在连接查询语句中，对于多个表的公共列，为了区别是哪个表的公共列，在语句中通过**表名前缀**指定公共列。如“**student.no**”，“**score.no**”。对于非公共列的列名可以加表的前缀，也可以不加。



[例]输出所有学生的name,cno和degree列，并按学号升序排序。

语句： **SELECT** student.name, score.cno, score.degree
FROM student, score
WHERE student.no=score.no
ORDER BY student.no

也可写： **SELECT** name, cno, degree
FROM student, score
WHERE student.no=score.no
ORDER BY student.no



student

score

no	name	sex	birthday	class
101	李军	男	20-Feb-76	95033
103	陆君	男	03-Jun-74	95031
105	匡明	男	02-Oct-75	95031
107	王丽	女	23-Jan-76	95033
108	曾华	男	01-Sep-77	95033
109	王芳	女	10-Feb-75	95031

no	no	degree
101	3-105	64
101	6-166	85
103	3-105	92
103	3-245	86
105	3-105	88
105	3-245	75
107	3-105	91
107	6-166	79
108	3-105	78
108	6-166	81
109	3-105	76
109	3-245	68

查询结果

student.no	name	sex	birthday	class	score.no	no	degree	name	no	degree
101	李军	男	20-Feb-76	95033	101	3-105	64	李军	3-105	64
101	李军	男	20-Feb-76	95033	101	6-166	85	李军	6-166	85
103	陆君	男	03-Jun-74	95031	103	3-245	86	陆君	3-245	86
103	陆君	男	03-Jun-74	95031	103	3-105	92	陆君	3-105	92
105	匡明	男	02-Oct-75	95031	105	3-245	75	匡明	3-245	75
105	匡明	男	02-Oct-75	95031	105	3-105	88	匡明	3-105	88
107	王丽	女	23-Jan-76	95033	107	3-105	91	王丽	3-105	91
107	王丽	女	23-Jan-76	95033	107	6-166	79	王丽	6-166	79
108	曾华	男	01-Sep-77	95033	108	3-105	78	曾华	3-105	78
108	曾华	男	01-Sep-77	95033	108	6-166	81	曾华	6-166	81
109	王芳	女	10-Feb-75	95031	109	3-245	68	王芳	3-245	68
109	王芳	女	10-Feb-75	95031	109	3-105	76	王芳	3-105	76



China University of Mining and Technology-Beijing

一条**SQL**语句可以同时完成选择和连接查询，这时**WHERE**子句是由连接谓词和选择谓词组成的复合条件

[例]输出95031班女学生的name,sex,class,cno和degree列，并按学号升序排序。

语句： **SELECT** student.name, student.sex,
 student .class,score.cno, score.degree
FROM student, score
WHERE student.no=score.no and
 class="95031" and sex="女"
ORDER BY score.no



student

no	name	sex	birthday	class
101	李军	男	20-Feb-76	95033
103	陆君	男	03-Jun-74	95031
105	匡明	男	02-Oct-75	95031
107	王丽	女	23-Jan-76	95033
108	曾华	男	01-Sep-77	95033
109	王芳	女	10-Feb-75	95031

score

no	no	degree
101	3-105	64
101	6-166	85
103	3-105	92
103	3-245	86
105	3-105	88
105	3-245	75
107	3-105	91
107	6-166	79
108	3-105	78
108	6-166	81
109	3-105	76
109	3-245	68

student.no	name	sex	birthday	class	score.no	no	degree
101	李军	男	20-Feb-76	95033	101	6-166	85
101	李军	男	20-Feb-76	95033	101	3-105	64
103	陆君	男	03-Jun-74	95031	103	3-245	86
103	陆君	男	03-Jun-74	95031	103	3-105	92
105	匡明	男	02-Oct-75	95031	105	3-245	75
105	匡明	男	02-Oct-75	95031	105	3-105	88
107	王丽	女	23-Jan-76	95033	107	6-166	79
107	王丽	女	23-Jan-76	95033	107	3-105	91
108	曾华	男	01-Sep-77	95033	108	6-166	81
108	曾华	男	01-Sep-77	95033	108	3-105	78
109	王芳	女	10-Feb-75	95031	109	3-245	68
109	王芳	女	10-Feb-75	95031	109	3-105	76



查询结果

China unive
Mining and

name	sex	class	no	degree
王芳	女	95031	3-245	68
王芳	女	95031	3-105	76

在多个表的连接查询中，表的公共列或非公共列前通常加**表名前缀**来指定是哪个表中的列，因此经常要反复写表名，较麻烦。为了简化输入，允许在查询中使用表的别名，以缩写表名。

可以在**FROM**子句中为表定义一个临时别名，然后在查询的各子句中可以用别名代替表名。

FROM子句指定别名的格式

FROM 表名1 别名1, 表名2 别名2, ...



[例]输出所有学生的name,cno和degree列，并按学号升序排序。

语句： **SELECT** student.name, score.cno, score.degree
FROM student, score
WHERE student.no=score.no
ORDER BY score.no

也可使用表的别名写为：

语句： **SELECT** x.name, y.cno, y.degree
FROM student x, score y
WHERE x.no=y.no
ORDER BY y.no



[例]输出所有学生的name,cname和degree列，并按no升序，课程号的降序排序。
(三个表的连接查询)

语句: **SELECT** name, cname, degree
FROM student x, course y,score z
WHERE x.no=z.no and y.cno=z.cno
ORDER BY x.no , y.cno **desc**

查询结果:

name	cname	degree
李军	数字电路	85
李军	计算机导论	64
陆君	操作系统	86
陆君	计算机导论	92
匡明	操作系统	75
匡明	计算机导论	88
王丽	数字电路	79
王丽	计算机导论	91
曾华	数字电路	81
曾华	计算机导论	78
王芳	操作系统	68
王芳	计算机导论	76



连接查询（续）

1.等值与非等值连接查询

2.自身连接

3.外连接



自身连接:在数据查询中有时需要将同一个表进行连接,这种连接称为自身连接.进行自身连接就如同两个分开的表一样,可以把一个表的某行与同一表中的另一行连接起来.

◆自身连接: 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 3.52]查询每一门课的间接先修课（即先修课的先修课）

```
SELECT FIRST.Cno, SECOND.Cpno  
FROM Course FIRST, Course SECOND  
WHERE FIRST.Cpno = SECOND.Cno;
```



自身连接（续）

FIRST表（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

SECOND表（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



自身连接（续）

查询结果：

Cno	Pcno
1	7
3	5
5	6



[例]输出选学3-105课程的并且成绩高于109号学生该门课程成绩的所有学生记录。

语句: **SELECT** x.no, x.cno, x.degree

FROM score x, score y

WHERE x.cno='3-105' AND x.degree > y.degree AND
Y.cno='3-105' AND y.no='109'

score

no	cno	degree
101	3-105	64
101	6-166	85
103	3-105	92
103	3-245	86
105	3-105	88
105	3-245	75
107	3-105	91
107	6-166	79
108	3-105	78
108	6-166	81
109	3-105	76
109	3-245	68



连接查询（续）

1.等值与非等值连接查询

2.自身连接

3.外连接



3. 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
 - 左外连接
 - 列出左边关系中所有的元组
 - 右外连接
 - 列出右边关系中所有的元组



外连接（续）

[例 3. 53] 改写[例 3.49]

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



外连接（续）

执行结果：

Student.Sn	Sname	Ssex	Sage	Sdept	Cno	Grade
201215121	李勇	男	20	CS	1	92
201215121	李勇	男	20	CS	2	85
201215121	李勇	男	20	CS	3	88
201215122	刘晨	女	19	CS	2	90
201215122	刘晨	女	19	CS	3	80
201215123	王敏	女	18	MA	NULL	NULL
201215125	张立	男	19	IS	NULL	NULL



3.3.4 多表查询

1. 连接查询
2. 嵌套查询
3. 集合查询
4. 基于派生表的查询



嵌套查询（续）

- 嵌套查询概述
 - 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

```
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（续）

- 1.带有IN谓词的子查询
- 2.带有比较运算符的子查询
- 3.带有ANY（SOME）或ALL谓词的子查询
- 4.带有EXISTS谓词的子查询



1. 带有IN谓词的子查询

[例 3.55] 查询与“刘晨”在同一个系学习的学生。

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



带有IN谓词的子查询（续）

用自身连接完成[例 3.55]查询要求

```
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（续）

[例 3.56] 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关系中找到选
修了3号课程的学生学号

① 首先在Course关系中找到
“信息系统”的课程号，为3号



带有IN谓词的子查询（续）

用连接查询实现[例 3.56]：

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



- 1.带有IN谓词的子查询
- 2.带有比较运算符的子查询
- 3.带有ANY（SOME）或ALL谓词的子查询
- 4.带有EXISTS谓词的子查询



2. 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<>）。

在[例 3.55]中，由于一个学生只可能在一个系学习，则可以用 = 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（续）

[例 3.57]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 y.Sno=x.Sno);
```

相关子查询



嵌套查询

- 1.带有**IN**谓词的子查询
- 2.带有比较运算符的子查询
- 3.带有**ANY（SOME）**或**ALL**谓词的子查询
- 4.带有**EXISTS**谓词的子查询



使用**ANY**或**ALL**谓词时必须同时使用比较运算
语义为：

> 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



\leq ANY 小于等于子查询结果中的某个值

\leq 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）

\neq （或 \neq ）ANY 不等于子查询结果中的某个值

\neq （或 \neq ）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（续）

[例 3.58] 查询非计算机科学系中比计算机科学系任意一个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= ' CS ')
AND Sdept <> 'CS ' ;      /*父查询块中的条件 */
```



带有ANY (SOME) 或ALL谓词的子查询 (续)

结果:

Sname	Sage
王敏	18
张立	19

执行过程:

- (1) 首先处理子查询, 找出CS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CS系且年龄小于 20 或 19的学生



带有ANY (SOME) 或ALL谓词的子查询 (续)

用聚集函数实现[例 3.58]

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



带有ANY（SOME）或ALL谓词的子查询（续）

[例 3.59] 查询非计算机科学系中比计算机科学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= ' CS ')
AND Sdept <> ' CS ';
```



带有ANY (SOME) 或ALL谓词的子查询 (续)

方法二：用聚集函数

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



带有ANY（SOME）或ALL谓词的子查询（续）

表3.7 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



嵌套查询

- 1.带有**IN**谓词的子查询
- 2.带有比较运算符的子查询
- 3.带有**ANY**（**SOME**）或**ALL**谓词的子查询
- 4.带有**EXISTS**谓词的子查询



- EXISTS谓词

- 存在量词 \exists
- 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

- 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



[例 3.60]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表
- 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送入结果表

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



[例 3.61] 查询没有选修1号课程的学生姓名。

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



带有EXISTS谓词的子查询（续）

[例 3.55]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');
```



带有EXISTS谓词的子查询（续）

[例 3.62] 查询选修了全部课程的学生姓名。

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

❖ 参见爱课程网数据库系统概论数据查询节动画
《EXISTS子查询》



带有EXISTS谓词的子查询（续）

[例 3.63]查询至少选修了学生201215122选修的全部课程的学生号码。

解题思路：

■ 用逻辑蕴涵表达：查询学号为x的学生，对所有的课程y，只要201215122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201215122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（续）

■ 等价变换：

$$\begin{aligned}(\forall y) \text{ p} \rightarrow \text{q} &\equiv \neg (\exists y (\neg (\text{p} \rightarrow \text{q}))) \\ &\equiv \neg (\exists y (\neg (\neg \text{p} \vee \text{q}))) \\ &\equiv \neg \exists y (\text{p} \wedge \neg \text{q})\end{aligned}$$

- 变换后语义：不存在这样的课程y，学生201215122选修了y，而学生x没有选。



带有EXISTS谓词的子查询（续）

■ 用NOT EXISTS谓词表示：

```
SELECT DISTINCT Sno
```

```
FROM SC SCX
```

```
WHERE NOT EXISTS
```

```
  (SELECT *
```

```
   FROM SC SCY
```

```
   WHERE SCY.Sno = ' 201215122 ' AND
```

```
         NOT EXISTS
```

```
         (SELECT *
```

```
          FROM SC SCZ
```

```
          WHERE SCZ.Sno=SCX.Sno AND
```

```
                SCZ.Cno=SCY.Cno));
```



3.3.4 多表查询

1. 连接查询
2. 嵌套查询
3. 集合查询
4. 基于派生表的查询



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（续）

[例 3.64] 查询计算机科学系的学生及年龄不大于19岁的学生。

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

- UNION: 将多个查询结果合并起来时，系统自动去掉重复元组
- UNION ALL: 将多个查询结果合并起来时，保留重复元组



集合查询（续）

[例3.66] 查询计算机科学系的学生与年龄不大于19岁的学生的交集。

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



集合查询（续）

[例 3.66] 实际上就是查询计算机科学系中年龄不大于19岁的学生。

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



集合查询（续）

[例 3.67]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 '  
INTERSECT  
SELECT Sno  
FROM SC  
WHERE Cno='2 ';
```



集合查询（续）

[例3.67]也可以表示为：

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



集合查询（续）

[例 3.68] 查询计算机科学系的学生与年龄不大于19岁的学生的差集。

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



集合查询（续）

[例3.68]实际上是查询计算机科学系中年龄大于19岁的学生

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



3.3.4 多表查询

1. 连接查询
2. 嵌套查询
3. 集合查询
4. 基于派生表的查询



基于派生表的查询

- 子查询不仅可以出现在**WHERE**子句中，还可以出现在**FROM**子句中，这时子查询生成的临时派生表（**Derived Table**）成为主查询的查询对象

[例3.57]找出每个学生超过他自己选修课程平均成绩的课程号

```
SELECT Sno, Cno
FROM SC, (SELECT Sno, Avg(Grade)
          FROM SC
          GROUP BY Sno)
      AS Avg_sc(avg_sno,avg_grade)
WHERE SC.Sno = Avg_sc.avg_sno
      and SC.Grade >=Avg_sc.avg_grade
```



基于派生表的查询（续）

- 如果子查询中没有聚集函数，派生表可以不指定属性列，子查询**SELECT**子句后面的列名为其缺省属性。

[例3.60]查询所有选修了1号课程的学生姓名，可以用如下查询完成：

```
SELECT Sname
FROM   Student,
       (SELECT Sno FROM SC WHERE Cno=' 1 ') AS SC1
WHERE  Student.Sno=SC1.Sno;
```



3.4.6 SELECT语句的一般格式

SELECT [ALL|DISTINCT]

<目标列表表达式> [别名] [, <目标列表表达式> [别名]] ...

FROM <表名或视图名> [别名]

[, <表名或视图名> [别名]] ...

|(<SELECT语句>)[AS]<别名>

[**WHERE** <条件表达式>]

[**GROUP BY** <列名1>[**HAVING**<条件表达式>]]

[**ORDER BY** <列名2> [ASC|DESC]];



第三章 关系数据库标准语言SQL

3.1 SQL概述

3.2 数据定义

3.3 数据查询

3.4 数据更新

3.6 空值的处理

3.7 视图

3.8 小结



3.4 数据更新

3.4.1 插入数据

3.4.2 修改数据

3.4.3 删除数据



3.4.1 插入数据

- 两种插入数据方式
 - 插入元组
 - 插入子查询结果
 - 可以一次插入多个元组



1. 插入元组

- 语句格式

INSERT

INTO <表名> [(<属性列1>[,<属性列2 >...])]

VALUES (<常量1> [,<常量2>]...);

- 功能

- 将新元组插入指定表中



插入元组（续）

[例3.69]将一个新学生元组（学号：201215128;姓名：陈冬;性别：男;所在系：IS;年龄：18岁）插入到Student表中。

```
INSERT
```

```
INTO Student (Sno,Sname,Ssex,Sdept,Sage)
```

```
VALUES ('201215128','陈冬','男','IS',18);
```



插入元组（续）

[例3.71] 插入一条选课记录（ '200215128','1 '）。

```
INSERT  
INTO SC(Sno,Cno)  
VALUES ('201215128 ',' 1 ');
```

关系数据库管理系统将在新插入记录的**Grade**列上自动地赋空值。

或者：

```
INSERT  
INTO SC  
VALUES (' 201215128 ',' 1 ',NULL);
```



2. 插入子查询结果

- 语句格式

INSERT

INTO <表名> [(<属性列1> [,<属性列2>...]]

子查询;

- 子查询

- SELECT子句目标列必须与INTO子句匹配
 - 值的个数
 - 值的类型



插入子查询结果（续）

[例3.72] 对每一个系，求学生的平均年龄，并把结果存入数据库
第一步：建表

```
CREATE TABLE Dept_age  
  ( Sdept    CHAR(15)                /*系名*/  
    Avg_age  SMALLINT);              /*学生平均年龄*/
```

第二步：插入数据

```
INSERT  
INTO Dept_age(Sdept,Avg_age)  
  SELECT Sdept, AVG(Sage)  
FROM   Student  
GROUP BY Sdept;
```



插入子查询结果（续）

- 关系数据库管理系统在执行插入语句时会检查所插元组是否破坏表上已定义的完整性规则
 - 实体完整性
 - 参照完整性
 - 用户定义的完整性
 - NOT NULL约束
 - UNIQUE约束
 - 值域约束



3.4 数据更新

3.4.1 插入数据

3.4.2 修改数据

3.4.3 删除数据



3.4.2 修改数据

- 语句格式

UPDATE <表名>

SET <列名>=<表达式>[,<列名>=<表达式>]...

[WHERE <条件>];

- 功能

- 修改指定表中满足WHERE子句条件的元组
- SET子句给出<表达式>的值用于取代相应的属性列
- 如果省略WHERE子句，表示要修改表中的所有元组



修改数据（续）

- 三种修改方式
 - 修改某一个元组的值
 - 修改多个元组的值
 - 带子查询的修改语句



1. 修改某一个元组的值

[例3.73] 将学生201215121的年龄改为22岁

```
UPDATE Student
```

```
SET Sage=22
```

```
WHERE Sno=' 201215121 ';
```



2. 修改多个元组的值

[例3.74] 将所有学生的年龄增加1岁。

```
UPDATE Student
```

```
SET Sage= Sage+1;
```



3. 带子查询的修改语句

[例3.75] 将计算机科学系全体学生的成绩置零。

```
UPDATE SC
SET   Grade=0
WHERE Sno IN
      (SELETE Sno
       FROM   Student
       WHERE  Sdept= 'CS' );
```



修改数据（续）

- 关系数据库管理系统在执行修改语句时会检查修改操作是否破坏表上已定义的完整性规则
 - 实体完整性
 - 参照完整性
 - 用户定义的完整性
 - NOT NULL约束
 - UNIQUE约束
 - 值域约束



3.4 数据更新

3.4.1 插入数据

3.4.2 修改数据

3.4.3 删除数据



3.4.3 删除数据

- 语句格式

DELETE

FROM <表名>

[WHERE <条件>];

- 功能
 - 删除指定表中满足WHERE子句条件的元组
- WHERE子句
 - 指定要删除的元组
 - 缺省表示要删除表中的全部元组，表的定义仍在字典中



删除数据（续）

- 三种删除方式
 - 删除某一个元组的值
 - 删除多个元组的值
 - 带子查询的删除语句



1. 删除某一个元组的值

[例3.76] 删除学号为201215128的学生记录。

DELETE

FROM Student

WHERE Sno= 201215128 ';



2. 删除多个元组的值

[例3.77] 删除所有的学生选课记录。

DELETE

FROM SC;



3. 带子查询的删除语句

[例3.78] 删除计算机科学系所有学生的选课记录。

```
DELETE  
FROM SC  
WHERE Sno IN  
    (SELETE Sno  
     FROM Student  
     WHERE Sdept= 'CS') ;
```



第三章 关系数据库标准语言SQL

3.1 SQL概述

3.2 数据定义

3.3 数据查询

3.4 数据更新

3.5 空值的处理

3.6 视图

3.7 小结



3.5 空值的处理

- 空值就是“不知道”或“不存在”或“无意义”的值。
- 一般有以下几种情况：
 - 该属性应该有一个值，但目前不知道它的具体值
 - 该属性不应该有值
 - 由于某种原因不便于填写



1. 空值的产生

- 空值是一个很特殊的值，含有不确定性。对关系运算带来特殊的问题，需要做特殊的处理。
- 空值的产生

[例 3.79]向SC表中插入一个元组，学生号是"201215126"，课程号是"1"，成绩为空。

```
INSERT INTO SC(Sno,Cno,Grade)
VALUES('201215126 ','1',NULL); /*该学生还没有考试成绩，取空值*/
```

或

```
INSERT INTO SC(Sno,Cno)
VALUES(' 201215126 ','1');          /*没有赋值的属性，其值为空值*/
```



空值的产生（续）

[例3.80] 将Student表中学生号为"201215200"的学生所属的系改为空值。

```
UPDATE Student  
SET Sdept = NULL  
WHERE Sno='201215200';
```



2. 空值的判断

- 判断一个属性的值是否为空值，用IS NULL或IS NOT NULL来表示。

[例 3.81] 从Student表中找出漏填了数据的学生信息

```
SELECT *  
FROM Student  
WHERE Sname IS NULL OR Ssex IS NULL OR  
Sage IS NULL OR Sdept IS NULL;
```



3. 空值的约束条件

- 属性定义（或者域定义）中
 - 有NOT NULL约束条件的不能取空值
 - 码属性不能取空值



4. 空值的算术运算、比较运算和逻辑运算

- 空值与另一个值（包括另一个空值）的算术运算的结果为空值
- 空值与另一个值（包括另一个空值）的比较运算的结果为UNKNOWN。
- 有UNKNOWN后，传统二值（TRUE，FALSE）逻辑就扩展成了三值逻辑



空值的算术运算、比较运算和逻辑运算(续)

表3.8 逻辑运算符真值表

x	y	x AND y	x OR y	NOT x
T	T	T	T	F
T	U	U	T	F
T	F	F	T	F
U	T	U	T	U
U	U	U	U	U
U	F	F	U	U
F	T	F	T	T
F	U	F	U	T
F	F	F	F	T

T表示TRUE, F表示FALSE, U表示UNKNOWN



空值的算术运算、比较运算和逻辑运算（续）

[例3.82] 找出选修1号课程的不及格的学生。

```
SELECT Sno  
FROM SC  
WHERE Grade < 60 AND Cno='1';
```

查询结果不包括缺考的学生



空值的算术运算、比较运算和逻辑运算（续）

[例 3.83] 选出选修1号课程的不及格的学生以及缺考的学生。

```
SELECT Sno  
FROM SC  
WHERE Grade < 60 AND Cno='1'  
UNION  
SELECT Sno  
FROM SC  
WHERE Grade IS NULL AND Cno='1'
```

或者

```
SELECT Sno  
FROM SC  
WHERE Cno='1' AND (Grade<60 OR Grade IS NULL);
```



第三章 关系数据库标准语言SQL

3.1 SQL概述

3.2 数据定义

3.3 数据查询

3.4 数据更新

3.5 空值的处理

3.6 视图

3.7 小结



3.6 视图

- 视图
 - 虚表，是从一个或几个基本表（或视图）导出的表
 - 只存放视图的定义，不存放视图对应的数据
 - 基表中的数据发生变化，从视图中查询出的数据也随之改变



3.6 视图

3.6.1 定义视图

3.6.2 查询视图

3.6.3 更新视图

3.6.4 视图的作用



3.6.1 定义视图

1.建立视图

2.删除视图



1. 建立视图

- 语句格式

CREATE VIEW

<视图名> [(<列名> [,<列名>]...)]

AS <子查询>

[**WITH CHECK OPTION**];

- 子查询可以是任意的**SELECT**语句，是否可以含有**ORDER BY**子句和**DISTINCT**短语，则决定具体系统的实现



建立视图（续）

- 组成视图的属性列名：全部省略或全部指定
 - 全部省略：
 - 由子查询中SELECT目标列中的诸字段组成
 - 明确指定视图的所有列名：
 - 某个目标列是聚集函数或列表表达式
 - 多表连接时选出了几个同名列作为视图的字段
 - 需要在视图中为某个列启用新的更合适的名字



建立视图（续）

[例3.84] 建立信息系学生的视图。

```
CREATE VIEW IS_Student  
AS  
SELECT Sno,Sname,Sage  
FROM Student  
WHERE Sdept= 'IS';
```



建立视图（续）

[例3.84] 建立信息系学生的视图。

```
CREATE VIEW IS_Student  
AS  
SELECT Sno,Sname,Sage  
FROM Student  
WHERE Sdept= 'IS'  
with check option;
```



建立视图（续）

- **WITH CHECK OPTION**
 - 对视图进行UPDATE，INSERT和DELETE操作时要保证更新、插入或删除的行为满足视图定义中的谓词条件（即子查询中的条件表达式）
- 定义IS_Student视图时加上了WITH CHECK OPTION子句，对该视图进行插入、修改和删除操作时，RDBMS会自动加上Sdept='IS'的条件。



建立视图（续）

- 基于多个基表的视图

[例3.86] 建立信息系选修了1号课程的学生视图（包括学号、姓名、成绩）。

```
CREATE VIEW IS_S1(Sno,Sname,Grade)
AS
SELECT Student.Sno,Sname,Grade
FROM Student,SC
WHERE Sdept= 'IS' AND
      Student.Sno=SC.Sno AND
      SC.Cno= '1';
```



建立视图（续）

- 基于视图的视图

[例3.87] 建立信息系选修了1号课程且成绩在90分以上的学生的视图。

```
CREATE VIEW IS_S2  
AS  
SELECT Sno,Sname,Grade  
FROM IS_S1  
WHERE Grade>=90;
```



建立视图（续）

- 带表达式的视图

[例3.88] 定义一个反映学生出生年份的视图。

```
CREATE VIEW BT_S(Sno,Sname,Sbirth)
AS
SELECT Sno,Sname,2014-Sage
FROM Student;
```



建立视图（续）

- 分组视图

[例3.89] 将学生的学号及平均成绩定义为一个视图

```
CREAT VIEW S_G(Sno,Gavg)
```

```
AS
```

```
SELECT Sno,AVG(Grade)
```

```
FROM SC
```

```
GROUP BY Sno;
```



2. 删除视图

- 语句的格式:

DROP VIEW <视图名>[CASCADE];

- 该语句从数据字典中删除指定的视图定义
- 如果该视图上还导出了其他视图，使用CASCADE级联删除语句，把该视图和由它导出的所有视图一起删除
- 删除基表时，由该基表导出的所有视图定义都必须显式地使用DROP VIEW语句删除



删除视图（续）

[例3.91] 删除视图BT_S和IS_S1

```
DROP VIEW BT_S; /*成功执行*/
```

```
DROP VIEW IS_S1; /*拒绝执行*/
```

要删除IS_S1，需使用级联删除：

```
DROP VIEW IS_S1 CASCADE;
```



3.7 视图

3.7.1 定义视图

3.7.2 查询视图

3.7.3 更新视图

3.7.4 视图的作用



3.7.2 查询视图

- 用户角度：查询视图与查询基本表相同
- 关系数据库管理系统实现视图查询的方法
 - 视图消解法 (View Resolution)
 - 进行有效性检查
 - 转换成等价的对基本表的查询
 - 执行修正后的查询



查询视图（续）

[例3.92] 在信息系学生的视图中找出年龄小于20岁的学生。

```
SELECT Sno,Sage  
FROM IS_Student  
WHERE Sage<20;
```

视图消解转换后的查询语句为：

```
SELECT Sno,Sage  
FROM Student  
WHERE Sdept= 'IS' AND Sage<20;
```



查询视图（续）

- 视图消解法的局限
 - 有些情况下，视图消解法不能生成正确的查询。

[例3.94]在S_G视图中查询平均成绩在90分以上的学生学号和平均成绩

```
SELECT *  
FROM S_G  
WHERE Gavg>=90;
```

S_G视图的子查询定义：

```
CREATE VIEW S_G (Sno,Gavg)  
AS  
SELECT Sno,AVG(Grade)  
FROM SC  
GROUP BY Sno;
```



查询视图（续）

错误：

```
SELECT Sno, AVG(Grade)
FROM SC
WHERE AVG(Grade)>=90
GROUP BY Sno;
```

正确：

```
SELECT Sno,AVG(Grade)
FROM SC
GROUP BY Sno
HAVING AVG(Grade)>=90;
```



建立视图（续）

目前多数关系数据库对行列子集视图的查询均能进行正确转换。

```
CREATE VIEW IS_Student  
AS  
SELECT Sno,Sname,Sage  
FROM Student  
WHERE Sdept= 'IS';
```



3.7 视图

3.7.1 定义视图

3.7.2 查询视图

3.7.3 更新视图

3.7.4 视图的作用



更新视图（续）

[例3.95] 将信息系学生视图IS_Student中学号”201215122”的学生姓名改为”刘辰”。

```
UPDATE IS_Student  
SET Sname= '刘辰'  
WHERE Sno= ' 201215122 ';
```

转换后的语句：

```
UPDATE Student  
SET Sname= '刘辰'  
WHERE Sno= ' 201215122 ' AND Sdept= 'IS';
```



更新视图（续）

[例3.96] 向信息系学生视图IS_S中插入一个新的学生记录，其中学号为“201215129”，姓名为“赵新”，年龄为20岁

```
INSERT  
INTO IS_Student  
VALUES('201215129','赵新',20);
```

转换为对基本表的更新：

```
INSERT  
INTO Student(Sno,Sname,Sage,Sdept)  
VALUES('200215129','赵新',20,'IS');
```



更新视图（续）

[例3.97]删除信息系学生视图IS_Student中学号为“201215129”的记录

DELETE

FROM IS_Student

WHERE Sno= ' 201215129 ';

转换为对基本表的更新:

DELETE

FROM Student

WHERE Sno= ' 201215129 ' AND Sdept= 'IS';



更新视图（续）

- 更新视图的限制：一些视图是不可更新的，因为对这些视图的更新不能唯一地有意义地转换成对相应基本表的更新

例：例3.89定义的视图S_G为不可更新视图。

```
UPDATE S_G  
SET      Gavg=90  
WHERE Sno= '201215121';
```

这个对视图的更新无法转换成对基本表SC的更新



更新视图（续）

- **DB2对视图更新的限制：**
 - 若视图是由两个以上基本表导出的，则此视图不允许更新。
 - 若视图的字段来自字段表达式或常数，则不允许对此视图执行INSERT和UPDATE操作，但允许执行DELETE操作。
 - 若视图的字段来自聚集函数，则此视图不允许更新。
 - 若视图定义中含有GROUP BY子句，则此视图不允许更新。
 - 若视图定义中含有DISTINCT短语，则此视图不允许更新。
 - 若视图定义中有嵌套查询，并且内层查询的FROM子句中涉及的表也是导出该视图的基本表，则此视图不允许更新。



3.7 视图

3.7.1 定义视图

3.7.2 查询视图

3.7.3 更新视图

3.7.4 视图的作用



3.7.4 视图的作用

- 视图能够简化用户的操作
- 视图使用户能以多种角度看待同一数据
- 视图对重构数据库提供了一定程度的逻辑独立性
- 视图能够对机密数据提供安全保护
- 适当的利用视图可以更清晰的表达查询



视图的作用（续）

- 视图对重构数据库提供了一定程度的逻辑独立性

— 数据库重构：

例：学生关系Student(Sno,Sname,Ssex,Sage,Sdept)

“垂直”地分成两个基本表：

SX(Sno,Sname,Sage)

SY(Sno,Ssex,Sdept)



视图的作用（续）

通过建立一个视图Student:

```
CREATE VIEW
```

```
Student(Sno,Sname,Ssex,Sage,Sdept)
```

```
AS
```

```
SELECT SX.Sno,SX.Sname,SY.Ssex,SX.Sage,SY.Sdept
```

```
FROM SX,SY
```

```
WHERE SX.Sno=SY.Sno;
```

使用用户的外模式保持不变，用户的应用程序通过视图仍然能够查找数据



视图的作用（续）

- 视图对重构数据库提供了一定程度的逻辑独立性(续)
 - 视图只能在一定程度上提供数据的逻辑独立性
 - 由于对视图的更新是有条件的，因此应用程序中修改数据的语句可能仍会因基本表结构的改变而改变。
- 视图能够对机密数据提供安全保护
 - 对不同用户定义不同视图，使每个用户只能看到他有权看到的数据



视图的作用（续）

- 适当的利用视图可以更清晰的表达查询
 - 经常需要执行这样的查询“对每个同学找出他获得最高成绩的课程号”。可以先定义一个视图，求出每个同学获得的最高成绩

```
CREATE VIEW VMGRADE
```

```
AS
```

```
SELECT Sno, MAX(Grade) Mgrade
```

```
FROM SC
```

```
GROUP BY Sno;
```



视图的作用（续）

然后用如下的查询语句完成查询：

```
SELECT SC.Sno,Cno  
FROM SC,VMGRADE  
WHERE SC.Sno=VMGRADE.Sno AND  
SC.Grade=VMGRADE .Mgrade;
```



小结

- **SQL**可以分为数据定义、数据查询、数据更新、数据控制四大部分
- **SQL**是关系数据库语言的工业标准。大部分数据库管理系统产品都能支持**SQL92**,但是许多数据库系统只支持**SQL99**、**SQL2008**和**SQL2011**的部分特征，至今尚没有一个数据库系统能够完全支持**SQL99**以上的标准。

