上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

SJTU

# 数 据 库 技 术

## Database System Technology

郭 捷

(guojie@sjtu.edu.cn)

# 第四章 结构查询语言SQL

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

## 目 录

# 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；

- SQL是一个通用的、功能极强的关系数据库语言；

  SQL语言的功能包括查询、操纵、定义和控制，是一个综合的、通用的关系数据库语言，也是一种高度非过程化语言，只要求用户指出做什么而不需要指出怎么做。SQL集成实现了数据库生命周期的全部操作。

# SQL语言的特点

1. 综合统一

2. 高度非过程化

3. 面向集合的操作方式

4. 以同一种语法结构提供两种使用方法

5. 语言简洁，易学易用

# 一、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：
  - 定义关系模式，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、完整性控制等；
- 用户数据库投入运行后，可根据需要随时逐步修改模式，不影响数据的运行。
- 数据操作符统一。

# 二、高度非过程化

- 非关系数据模型的数据操纵语言“面向过程”，必须制定存取路径。
- 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。

# 三、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。

# 四、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。

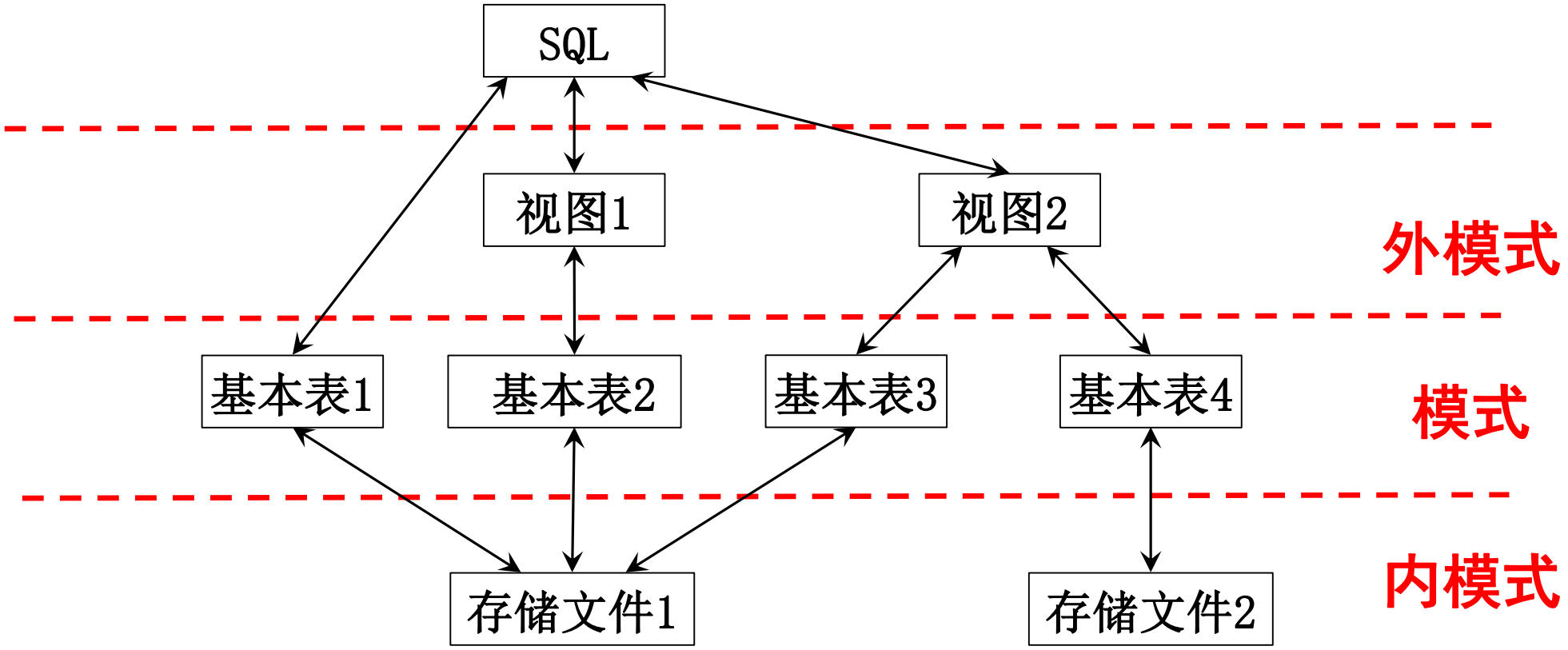
- SQL又是嵌入式语言
  - ✓ 能够嵌入到高级语言（例如C，C++，Java） 程序中，供程序员设计程序时使用。

# 五. 语言简捷，易学易用

表 1　SQL 语言的动词

| SQL 功　能 | 动　　词 |
| --- | --- |
| 数 据 定 义 | CREATE，DROP，ALTER |
| 数 据 查 询 | SELECT |
| 数 据 操 纵 | INSERT，UPDATE<br>DELETE |
| 数 据 控 制 | GRANT，REVOKE |

# SQL的基本概念




SQL对关系数据库模式的支持

# SQL的基本概念

- 基本表
  - 本身独立存在的表。
  - SQL中一个关系就对应一个基本表。
  - 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
  - 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
  - 从一个或几个基本表导出的表。
  - 数据库中只存放视图的定义而不存放视图对应的数据。
  - 视图是一个虚表。
  - 用户可以在视图上再定义视图。

# 第四章 结构查询语言SQL

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

## 目 录

# “学生-课程”数据库

- “学生-课程”模式S-T ：

学生表：Student(Sno, Sname, Ssex, Sage, Sdept)

课程表：Course(Cno, Cname, Cpno, Ccredit)

学生选课表：SC(Sno, Cno, Grade)

# 学生表（Student)

**Student（学生）**

| 学号<br>Sno | 姓名<br>Sname | 性别<br>Ssex | 年龄<br>Sage | 所在系<br>Sdept |
|---|---|---|---|---|
| 201215121 | 李勇 | 男 | 20 | CS |
| 201215122 | 刘晨 | 女 | 19 | CS |
| 201215123 | 王敏 | 女 | 18 | MA |
| 201215125 | 张立 | 男 | 19 | IS |

# 课程表（Course）

**Course（课程）**

| 课程号<br>Cno | 课程名<br>Cname | 先行课<br>Cpno | 学分<br>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 |

# 学生选课表（SC）

SC （选修关系）

| 学号<br>Sno | 课程号<br>Cno | 成绩<br>Grade |
|---|---|---|
| 201215121 | 1 | 92 |
| 201215121 | 2 | 85 |
| 201215121 | 3 | 88 |
| 201215122 | 2 | 90 |
| 201215122 | 3 | 80 |

# SQL数据定义语句

- SQL的数据定义功能：模式定义、表定义、视图定义和索引定义。

| 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 |


# 01

# 模式的定义与删除

# 数据库模式的定义和删除

1. 定义数据库模式

CREATE SCHEMA〈模式名〉AUTHORIZATION〈用户名〉

[例1] 定义一个学生 - 课程模式 S-T

CREATE SCHEMA “S-T” AUTHORIZATION WANG;

为用户WANG定义了一个模式S-T

# 数据库模式的定义和删除

注意：如果没有指定<模式名>，那么<模式名>隐含为<用户名>。

[例2] CREATE SCHEMA AUTHORIZATION WANG;

<模式名>隐含为用户名WANG

# 数据库模式的定义和删除

- 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>]**

# 数据库模式的定义和删除

[例3] CREATE SCHEMA TEST AUTHORIZATION ZHANG

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

为用户ZHANG创建了一个模式TEST，并在其中定义了一个表TAB1。

# 数据库模式的定义和删除

## 2. 删除数据库模式

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，没有任何下属的对象时才能执行。

# 数据库模式的建立和删除

[例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。


# 02

# 基本表的定义、删除与修改

# 一、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

# 常用的完整性约束

- 主码约束：　PRIMARY　KEY
- 唯一性约束：UNIQUE
- 空值/非空值约束：NULL/NOT NULL
- 参照完整性约束：FOREIGN KEY（<列名>，...）REFERENCES<父表名>（主码列名，...）
- 检查约束：CHECK（<逻辑表达式>）

# 例题

[例5] 建立“学生”表Student，学号是主码，姓名取值唯一。

```
CREATE TABLE Student
(Sno CHAR(9)      PRIMARY KEY,   /* 列级完整性约束条件，主码*/
 Sname CHAR(20) UNIQUE,         /* Sname取唯一值*/
 Ssex CHAR(2),
 Sage SMALLINT,
 Sdept CHAR(20)
);
```

# 例题

[例6] 建立一个“课程”表Course。

```
CREATE TABLE Course
( Cno CHAR(4) PRIMARY KEY,   /* 列级完整性约束条件，主码*/
  Cname CHAR(40) NOT NULL,   /* Cname不能取空值*/
  Cpno CHAR(4) ,
  Ccredit SMALLINT,
  FOREIGN KEY (Cpno) REFERENCES Course(Cno)
);
```

# 例题

[例7] 建立一个“学生选课”表SC。

```
CREATE TABLE SC
(Sno CHAR(9),
 Cno CHAR(4),
 Grade SMALLINT CHECK (Grade>=0 and Grade<=100),
 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*/
);
```

# 二、数据类型

- SQL中域的概念用数据类型来实现。
- 定义表的属性时需要指明其数据类型及长度。
- 选用哪种数据类型：
  - ✓取值范围
  - ✓要做哪些运算

# 数据类型

| 数据类型 | 含义 |
| --- | --- |
| CHAR(n), CHARACTER(n) | 长度为n的定长字符串 |
| VARCHAR(n), CHARACTERVARYING(n) | 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(p, d) | 定点数，由p位数字（不包括符号、小数点）组成，小数点后有d位数字 |
| DECIMAL(p, d) | 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(n) | 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 |

# 三、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*/
  - 方法二：在创建模式语句中同时创建表，如例3。
  - 方法三：设置所属的模式。

# 三、模式与表

- 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。
- RDBMS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。
- 显示当前的搜索路径：SHOW search_path。
- 搜索路径的当前默认值是：$user，PUBLIC。

# 三、模式与表

- DBA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。

过程：RDBMS发现搜索路径中第一个模式名S-T存在，就把该模式作为基本表Student所属的模式。

# 四、修改基本表

ALTER TABLE <表名>

[ ADD [ COLUMN ] <新列名> <数据类型> [ 完整性约束 ] ]

[ ADD <表级完整性约束>]

[ DROP [ COLUMN ] <列名> [ CASCADE|RESTRICT] ]

[ DROP CONSTRAINT <完整性约束名> [ CASCADE|RESTRICT] ]

[ ALTER COLUMN <列名> <数据类型> ];

- <表名>：要修改的基本表
- ADD子句：增加新列、新的列完整性约束条件、新的表完整性约束条件
- DROP CONSTRAINT子句：删除指定的完整性约束条件
- ALTER COLUMN子句：用于修改列名和数据类型

# 四、修改基本表

［例8］向Student表增加“入学时间”列，其数据类型为日期型。

**ALTER TABLE Student ADD S_entrance 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9］将年龄的数据类型由字符型（假设原来的数据类型是字符型）改为整数。

**ALTER TABLE Student ALTER COLUMN Sage INT;**

［例10］增加课程名称必须取唯一值的约束条件。

**ALTER TABLE Course ADD UNIQUE(Cname);**

# 五、删除基本表

DROP TABLE <表名> [RESTRICT | CASCADE];

- RESTRICT：删除表是有限制的。
  - 欲删除的基本表不能被其它表的约束所引用。
  - 如果存在依赖该表的对象，则此表不能被删除。
  - 缺省情况是RESTRICT。

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

# 五、删除基本表

[例11] 删除Student表。

```
DROP TABLE Student CASCADE ;
```

- 基本表定义被删除，数据被删除。
- 表上建立的索引、视图、触发器等一般也将被删除。

# 五、删除基本表

[例12] 若表上建有视图，选择RESTRICT时表不能删除。

```
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
```

# 五、删除基本表

[例12] 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.
```

# 五、删除基本表

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<br>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 | X | √ | X | √ | √<br>保留 | √<br>保留 | √<br>保留 |
| 3 | DEFAULT, PRIMARY KEY, CHECK（只包含该表的列）NOT NULL等约束 | √ | √ | √ | √ | √ | √ | √ |
| 4 | 外码 FOREIGN KEY | X | √ | X | √ | X | √ | X |
| 5 | 触发器 TRIGGER | X | √ | X | √ | √ | √ | √ |
| 6 | 函数或存储过程 | X | √ | √<br>保留 | √<br>保留 | √<br>保留 | √<br>保留 | √<br>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。


# 03

# 索引的建立与删除

# 建立与删除索引

- 建立索引是加快查询速度的有效手段（内模式的范畴）

  顺序文件上的索引、B+树索引、hash索引、位图索引，等等

- 建立索引
  - DBA或表的属主（即建立表的人）根据需要建立。
  - 有些DBMS自动建立以下列上的索引
    - PRIMARY KEY
    - UNIQUE
- 使用索引
  - DBMS自动选择是否使用索引以及使用哪些索引

# 一、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>] ]…);

- 用<表名>指定要建索引的基本表名字
- 索引可以建立在该表的一列或多列上，各列名之间用逗号分隔
- 用<次序>指定索引值的排列次序，升序：ASC，降序：DESC。缺省值：ASC
- UNIQUE表明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

# 一、建立索引

- 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。也即聚簇索引的索引项顺序与表中记录的物理顺序一致。

# 例题

[例13] CREATE CLUSTER INDEX Stusname ON Student(Sname);

- 在Student表的Sname（姓名）列上建立一个聚簇索引。
- 在最经常查询的列上建立聚簇索引以提高查询效率。
- 一个基本表上最多只能建立一个聚簇索引。
- 经常更新的列不宜建立聚簇索引。

# 一、建立索引

- 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。这相当于增加了一个UNIQUE约束

# 例题

[例14] 为学生-课程数据库中的Student，Course，SC三个表建立索引。其中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);
```

# 二、修改索引

ALTER INDEX <旧索引名> RENAME TO <新索引名>；

[例15] 将SC表的SCno索引名改为SCSno。

```
ALTER INDEX Scno RENAME TO SCSno;
```

# 三、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16] 删除Student表的Stusname索引。

```
DROP INDEX Stusname;
```

# 第四章 结构查询语言SQL

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

## 目 录

# 数据查询概述

- 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>… ]|(< 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 ] ];

SELECT语句的执行过程是什么？


# 01

# 单表查询

# 单表查询

查询仅涉及一个表，是一种最简单的查询操作

一、选择表中的若干列

二、选择表中的若干元组

三、对查询结果排序（ORDER BY）

四、使用集函数

五、对查询结果分组（GROUP BY）

# 一、选择表中的若干列

## 1. 查询指定列

［例1］ 查询全体学生的学号与姓名。

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

［例2］ 查询全体学生的姓名、学号、所在系。

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

# 2、查询全部列

[例3] 查询全体学生的详细记录。

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

或

```
SELECT  *
FROM Student;
```

# 3、查询经过计算的值

SELECT子句的<目标列表达式>为表达式

- 算术表达式
- 字符串常量
- 函数
- 列别名

# 3、查询经过计算的值

[例4] 查全体学生的姓名及其出生年份。

```
SELECT Sname，2014-Sage
FROM Student;
```

输出结果：

| Sname | 2014-Sage |
| --- | --- |
| 李勇 | 1994 |
| 刘晨 | 1995 |
| 王名 | 1996 |
| 张立 | 1995 |

# 3、查询经过计算的值

[例5] 查询全体学生的姓名、出生年份和所有系，要求用小写字母表示所有系名。

```
SELECT Sname, '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 |
| 王名 | Year of Birth: | 1996 | ma |
| 张立 | Year of Birth: | 1995 | is |

# 3、查询经过计算的值

[例5.1] 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, ISLOWER(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 |
| 王名 | Year of Birth: | 1996 | ma |
| 张立 | Year of Birth: | 1995 | is |

# 二、选择表中的若干元组

## 1、消除取值重复的行

- 在SELECT子句中使用DISTINCT短语，缺省为ALL

假设SC表中有下列数据

| Sno | Cno | Grade |
|---|---|---|
| 201215121 | 1 | 92 |
| 201215121 | 2 | 85 |
| 201215121 | 3 | 88 |
| 201215122 | 2 | 90 |
| 201215122 | 3 | 80 |

# All 与 DISTINCT

[例6] 查询选修了课程的学生学号。

(1)
```
SELECT Sno
FROM SC;
```

等价于

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

结果：

| Sno |
| --- |
| 201215121 |
| 201215121 |
| 201215121 |
| 201215122 |
| 201215122 |

# All 与 DISTINCT

(2) SELECT DISTINCT Sno

FROM SC;

结果：

| Sno |
| --- |
| 201215121 |
| 201215122 |

# 2、查询满足条件的元组

WHERE子句常用的查询条件：

| 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7］查询计算机科学系全体学生的名单。

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

［例8］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9］查询考试成绩有不及格的学生的学号。

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

# (2) 确定范围

- 使用谓词　BETWEEN … AND …
  NOT BETWEEN … AND …

[例10] 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;
```

# (2) 确定范围

[例11] 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

使用谓词　　IN <值表>,　NOT IN <值表>

<值表>：用逗号分隔的一组取值

[例12]查询信息系（IS）、数学系（MA）和计算机科学系（CS）学生的姓名和性别。

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

# (3) 确定集合

[例13]查询既不是信息系、数学系，也不是计算机科学系的学生的姓名和性别。

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

# (4) 字符串匹配

- [NOT] LIKE ‘<匹配串>’ [ESCAPE ‘<换码字符>’]
  - 查找指定的属性值与<匹配串>相匹配的元组。
  - 匹配串：固定字符串或含通配符的字符串
  - 当匹配模板为固定字符串时，
    - 用 = 运算符取代 LIKE 谓词
    - 用 != 或 < >运算符取代 NOT LIKE 谓词

# 1) 匹配串为固定字符串

[例14] 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';
```

# 通配符

- %（百分号）：代表任意长度（长度可以为0）的字符串
  - 例：a%b表示以a开头，以b结尾的任意长度的字符串。

    如 acb，addgb，ab 等都满足该匹配串

- _（下横线）：代表任意单个字符
  - 例：a_b表示以a开头，以b结尾的长度为3的任意字符串。

    如 acb，afb等都满足该匹配串

# 2) 匹配串为含通配符的字符串

[例15] 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%’；     /*%表示任意长度*/
```

[例16] 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_ _’；/*_代表任意单个字符*/
```

# 2) 匹配串为含通配符的字符串

[例17] 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18] 查询所有不姓刘的学生姓名。

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

# 3) 使用换码字符将通配符转义为普通字符

[例19] 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20] 查询以“DB_”开头，且倒数第3个字符为i的课程情况。

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

# (5) 涉及空值的查询

- 使用谓词 IS NULL 或 IS NOT NULL
- “IS NULL” 不能用 “= NULL” 代替

[例21] 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学号和相应课程号。

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

# (5) 涉及空值的查询

[例22] 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

- 用逻辑运算符 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 …

# (6) 多重条件查询

[例23] 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;
```

# (6) 多重条件查询

改写[例12] 查询信息系（IS）、数学系（MA）和计算机科学系（CS）学生的姓名和性别。

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

可改写为：

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

# 三、对查询结果排序

## 使用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

# 三、对查询结果排序

［例24］ 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;
```

# 三、对查询结果排序

[例25] 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;
```

# 四、使用聚集函数

- 5类主要聚集函数：
  - 计数

    COUNT（[DISTINCT|ALL] *）　　（统计元组个数）

    COUNT（[DISTINCT|ALL] <列名>）　　（统计一列中值的个数）
  - 计算一列值的总和

    SUM（[DISTINCT|ALL] <列名>）
  - 计算一列值的平均值

    AVG（[DISTINCT|ALL] <列名>）
  - 求一列值中的最大最小值

    MAX（[DISTINCT|ALL] <列名>）

    MIN（[DISTINCT|ALL] <列名>）

# 四、使用聚集函数

[例26] 查询学生总人数。

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

[例27] 查询选修了课程的学生人数。

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

[例28] 计算1号课程的学生平均成绩。

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

# 四、使用聚集函数

[例29] 查询选修1号课程的学生最高分数。

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

[例30] 查询学生201215012选修课程的总学分数。

```
SELECT SUM(Ccredit)
FROM SC, Course
WHERE Sno='201215012' AND SC.Cno=Course.Cno;
```

注意：WHERE子句中不能用聚集函数作为条件表达式。

# 五、对查询结果分组

- 使用GROUP BY子句分组，按某一列或多列的值分组，值相等的为一组。
- 细化聚集函数的作用对象
  - 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

# 使用GROUP BY子句分组

[例31] 求各个课程号及相应的选课人数。

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

结果

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

# 使用HAVING短语筛选最终输出结果

[例32] 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;
```

注意：WHERE子句作用于基本表或视图，选择满足条件的元组；
HAVING短语作用于组，从分好的组中选择满足条件的组；

# 使用HAVING短语筛选最终输出结果

[例32]1　查询平均成绩大于等于90分的学生学号和平均成绩。

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

注意：WHERE子句不能用聚集函数作为条件表达式；

# 使用HAVING短语筛选最终输出结果

[例32]1　查询平均成绩大于等于90分的学生学号和平均成绩。

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

注意：聚集函数只能用于SELECT子句和GROUP BY中的HAVING子句。


# 02

# 连接查询

# 连接查询

- 同时涉及多个表的查询称为连接查询；
- 用来连接两个表的条件称为连接条件或连接谓词 ；
  - 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，但不必相同！
- 一般格式：
  - [<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>
    比较运算符：=、>、<、>=、<=、!=（或<>）等
  - [<表名1>.]<列名1> BETWEEN [<表名2>.]<列名2> AND [<表名2>.]<列名3>

# 连接查询

SQL中连接查询的主要类型

- 等值连接(含自然连接)
- 非等值连接查询
- 自身连接查询
- 外连接查询
- 复合条件连接查询

# 一、等值与非等值连接查询

- 等值连接：连接运算符为 = 。

[例33] 查询每个学生及其选修课程的情况。

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

# 查询结果：

| Student.Sno | Sname | Ssex | Sage | Sdept | SC.Sno | Cno | Grade |
|---|---|---|---|---|---|---|---|
| 201215121 | 李勇 | 男 | 20 | CS | 201215121 | 1 | 92 |
| 201215121 | 李勇 | 男 | 20 | CS | 201215121 | 2 | 85 |
| 201215121 | 李勇 | 男 | 20 | CS | 201215121 | 3 | 88 |
| 201215122 | 刘晨 | 女 | 19 | CS | 201215122 | 2 | 90 |
| 201215122 | 刘晨 | 女 | 19 | CS | 201215122 | 3 | 80 |

- 任何子句中引用表1和表2中同名属性时，都必须加表名前缀。引用唯一属性名时可以加也可以省略表名前缀。

# 连接操作的执行过程

- 嵌套循环连接算法(NESTED-LOOP)
  - 首先在表1中找到第一个元组，然后从头开始扫描表2，逐一查找满足连接件的元组，找到后就将表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全部查找完后，再找表1中第二个元组，然后再从头开始扫描表2，逐一查找满足连接条件的元组，找到后就将表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中的全部元组都处理完毕。

# 连接操作的执行过程

- 索引连接(INDEX-JOIN)
  - 对表2按连接字段Sno建立索引。
  - 对表1中的每个元组，依次根据其连接字段值查询表2的索引，从中找到满足条件的元组，找到后就将表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。

# 二、自然连接

- 等值连接的一种特殊情况，把目标列中重复的属性列去掉。

[例34] 对[例33]用自然连接完成。

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

# 三、自身连接

- 一个表与其自己进行连接，称为表的自身连接；
- 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35]　查询每一门课的间接先修课（即先修课的先修课）

# 三、自身连接

FIRST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 |

# 三、自身连接

SECOND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 |

# 三、自身连接

[例35] 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;
```

查询结果：

| Cno | Cpno |
|---|---|
| 1 | 7 |
| 3 | 5 |
| 5 | 6 |

# 四、外连接（Outer Join）

- 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悬浮元组一并输出；

# 四、外连接（Outer Join）

［例36］查询每个学生及其选修课程的情况，包括没有选修课程的学生
----用外连接操作（改写例33）

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

# 四、外连接（Outer Join）

| Student.Sno | 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 |

# 四、外连接（Outer Join）

- 左外连接
  - 列出左边关系（如本例Student）中所有元组；
- 右外连接
  - 列出右边关系（如本例SC）中所有元组；

# 五、复合条件连接

- WHERE子句中含多个连接条件时，称为复合条件连接

[例37]查询选修2号课程且成绩在90分以上的所有学生的学号和姓名

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

# 多表连接

[例38] 查询每个学生的学号、姓名、选修的课程名及成绩。

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

结果：

| Student.Sno | Sname | Cname | Grade |
|---|---|---|---|
| 201215121 | 李勇 | 数据库 | 92 |
| 201215121 | 李勇 | 数学 | 85 |
| 201215121 | 李勇 | 信息系统 | 88 |
| 201215122 | 刘晨 | 数学 | 90 |
| 201215122 | 刘晨 | 信息系统 | 80 |


# 03

# 嵌套查询

SJTU

# 嵌套查询

- 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；

例：
```
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

- 允许多层嵌套，但子查询有限制：
  - 子查询的SELECT语句中不能使用ORDER BY子句
  - ORDER BY子句只能对最终查询结果排序
- 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；

# 引出子查询的谓词

- 带有IN谓词的子查询
- 带有比较运算符的子查询
- 带有ANY或ALL谓词的子查询
- 带有EXISTS谓词的子查询

# 一、带有IN谓词的子查询

［例39］ 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 ’;
```

结果为：  Sdept  CS

# 一、带有IN谓词的子查询

② 查找所有在CS系学习的学生。

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

结果为：

| Sno | Sname | Sdept |
|---|---|---|
| 201215121 | 李勇 | CS |
| 201215122 | 刘晨 | CS |

# 构造嵌套查询

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 ’ )；
```

注意：此查询为不相关子查询，DBMS求解该查询时分步去做。

# 嵌套查询分类

- 不相关子查询

  -- 子查询的查询条件不依赖于父查询。

  - 处理过程：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

# 带有IN谓词的子查询

- 用自身连接完成本查询要求

```
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

[例40]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，结果为3号

# 带有IN谓词的子查询

- 用连接查询实现上例：

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

# 二、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谓词配合使用

# 二、带有比较运算符的子查询

例：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39]可以用 = 代替IN ：

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

# 二、带有比较运算符的子查询

注意：子查询一定要跟在比较符之后

错误的例子：

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

# 嵌套查询分类

- 相关子查询

  -- 子查询的查询条件依赖于父查询。

处理过程：

- 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。

# 相关子查询

[例41] 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

可能的执行过程：

1. 从外层查询中取出SC的一个元组x，将元组x的Sno值（201215121）传送给内层查询。

```
SELECT AVG(Grade)
FROM   SC y
WHERE y.Sno='201215121';
```

2. 执行内层查询，得到值88（近似值），用该值代替内层查询，得到外层查询：

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

# 相关子查询

3. 执行这个查询，得到：

（201215121，1）

（201215121，3）

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:

（201215121，1）

（201215121，3）

（201215122，2）

# 三、带有ANY或ALL谓词的子查询

谓词语义

- ANY：某一个值
- ALL：所有值

| | |
|---|---|
| >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（无意义） |
| !=（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 |

# 三、带有ANY或ALL谓词的子查询

[例42] 查询其他系中比计算机系某一个（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 ’ ;  /* 注意这是父查询块中的条件 */
```

# 三、带有ANY或ALL谓词的子查询

结果

| Sname | Sage |
| --- | --- |
| 王敏 | 18 |
| 张立 | 19 |

执行过程：

1. DBMS执行此查询时，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(19，20)；

2. 处理父查询，找所有不是CS系且年龄小于 19 或 20的学生

# 三、带有ANY或ALL谓词的子查询

- ANY和ALL谓词有时可以用聚集函数实现
  - ANY与ALL与聚集函数（MIN，MAX）的对应关系

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

# 三、带有ANY或ALL谓词的子查询

[例42']：查询其他系中比计算机系某一个（任意一个）学生年龄小的学生姓名和年龄

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

注意：用聚集函数实现子查询通常比直接用ANY或ALL查询效率要高，因为前者通常能够减少比较次数。

# 三、带有ANY或ALL谓词的子查询

[例43] 查询其他系中比计算机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 ’ ;
```

# 三、带有ANY或ALL谓词的子查询

方法二：用集函数

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

# 四、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，则返回真值
    - 若内层查询结果为空，则返回假值
  - 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

# 四、带有EXISTS谓词的子查询

[例41] 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。

# 四、带有EXISTS谓词的子查询

[例44] 查询所有选修了1号课程的学生姓名。

- 用嵌套查询

```
SELECT Sname
FROM Student
WHERE EXISTS
        (SELECT *          /*相关子查询*/
         FROM SC
         WHERE Sno=Student.Sno
         AND Cno= ' 1 ');
```

# 四、带有EXISTS谓词的子查询

- 用连接运算

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

# 四、带有EXISTS谓词的子查询

[例45] 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');
```

# 四、带有EXISTS谓词的子查询

3. 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。

# 四、带有EXISTS谓词的子查询

例：[例39]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’);
```

# 四、带有EXISTS谓词的子查询

4.用EXISTS/NOT EXISTS实现全称量词(难点)

- 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：

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

# 全称量词

［例46］ 查询选修了全部课程的学生姓名。

(∀课程X)该学生选修X ≡ ¬ （ ∃ 课程X (¬ 该学生选修X)）

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

# 四、带有EXISTS谓词的子查询

## 5. 用EXISTS/NOT EXISTS实现逻辑蕴函(难点)

- 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：

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

| p | q | $p \rightarrow q$ |
|---|---|---|
| True | True | True |
| True | False | False |
| False | True | True |
| False | False | True |

| p | q | $\neg p$ | $\neg p \vee q$ |
|---|---|---|---|
| True | True | False | True |
| True | False | False | False |
| False | True | True | True |
| False | False | True | True |

# 逻辑蕴含

［例47］ 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：**

- 用逻辑蕴函表达：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：$(\forall y)\ p \rightarrow q$

# 逻辑蕴含

- 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201215122选修了y，而学生x没有选。

# 逻辑蕴含

- 用NOT EXISTS谓词表示：

不存在这样的课程y，学生201215122选修了y，而学生x没有选

```
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));
```


# 04

# 集合查询

SJTU

# 集合查询

- 集合操作种类：
  - 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。

# 并操作

[例45] 查询计算机科学系的学生或年龄不大于19岁的学生。

方法一：

```
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。

# 并操作

方法二：

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

# 交操作

[例51] 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 ';
```

# 交操作

[例51]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’);
```

# 差操作

[例52] 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;
```

# 差操作

[例52] 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;
```


# 05

# 派生查询

# 派生查询

- 子查询不仅可以出现在WHERE子句中，还可以出现在FROM子句中；

- 子查询生成的临时派生表（derived table）成为主查询的查询对象。

# 基于派生表的查询

[例] 找出每个学生超过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)    派生表
WHERE SC.Sno=Avg_sc.avg_sno and SC.Grade>=Avg_sc.avg_grade
```

# 基于派生表的查询

［例］查询所有选修了1号课程的学生姓名。

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

- 如果子查询没有聚集函数，派生表可以不指定属性列，子查询SELECT子句后面的列名为其默认属性；

# 第四章 结构查询语言SQL

上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

## 目 录

# 一、插入数据

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

# 1. 插入单个元组

- 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。

# 1. 插入单个元组

[例1] 将一个新学生记录（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);
```

- 如果不指定任何属性列，则新插入的元组必须在每个属性列上均有值；
- 属性列的顺序可与表定义中的顺序不一致；
- 如果仅指定部分属性列，则新元组在没有出现的属性列上取空值。
- VALUES子句提供的值必须与INTO子句匹配（值的个数，值的类型）。

# 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;

- 功能

  将子查询结果插入指定表中

# 2. 插入子查询结果

[例2] 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  Deptage
      (Sdept   CHAR(15)                  /* 系名*/
       Avgage SMALLINT);    /*学生平均年龄*/
```

# 2. 插入子查询结果

第二步：插入数据

```
INSERT
INTO   Deptage(Sdept，Avgage)
       SELECT   Sdept，AVG(Sage)
       FROM   Student
       GROUP BY Sdept;
```

# 2. 插入子查询结果

- INTO子句(与插入单条元组类似)
  - 指定要插入数据的表名及属性列
  - 属性列的顺序可与表定义中的顺序不一致
  - 没有指定属性列：表示要插入的是一条完整的元组
  - 指定部分属性列：插入的元组在其余属性列上取空值
- 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

# 2. 插入子查询结果

- DBMS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
    - 对于有NOT NULL约束的属性列是否提供了非空值
    - 对于有UNIQUE约束的属性列是否提供了非重复值
    - 对于有值域约束的属性列所提供的属性值是否在值域范围内

# 二、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

# 二、修改数据

- SET子句
  - 指定修改方式
  - 要修改的列
  - 修改后取值
- WHERE子句
  - 指定要修改的元组
  - 缺省表示要修改表中的所有元组

# 1. 修改元组的值

[例3] 将学生201215121的年龄改为22岁。

```
UPDATE  Student
SET Sage=22
WHERE  Sno=’201215121’;
```

[例4] 将所有学生的年龄增加1岁。

```
UPDATE Student
SET Sage= Sage+1;
```

# 2. 带子查询的修改语句

[例5] 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');
```

# 三、删除数据

```
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所有元组（表的定义仍在数据字典中）

# 1. 删除元组的值

[例6] 删除学号为201215128的学生记录。

```
DELETE
FROM Student
WHERE Sno= '201215128';
```

[例7] 删除2号课程的所有选课记录。

```
DELETE
FROM SC;
WHERE Cno='2';
```

[例8] 删除所有的学生选课记录。

```
DELETE
FROM SC;
```

# 2. 带子查询的删除语句

［例9］ 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');
```

# 三、删除数据

- DBMS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？
  - 参照完整性
    - 不允许删除
    - 级联删除

# 第四章 结构查询语言SQL

上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

## 目 录

# 空值的概念

- 空值就是“不知道”或“不存在”或“无意义”的值。

- SQL语言允许某些元组的某些属性在一定情况下取空值：
  - 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；
  - 某种原因不便于填写；

# 空值的产生

[例3.79] 向SC表中插入一个元组，学号是“201215126”，课程号是“1”，成绩为空。

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

或

```
INSERT INTO SC(Sno, Cno)
    VALUES('201215126', '1');
   /*在插入语句中没有赋值的属性，取空值*/
```

# 空值的产生

[例3.80] 将Student表中学号是“201215200”的学生所属的系改为空值。

```
UPDATE Student

SET Sdept = NULL

WHERE Sno = ' 201215200' ;
```

注：外连接会产生空值；空值的关系运算也会产生空值

# 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

[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;
```

# 空值的约束条件

- 属性定义（或域定义）中有NOT NULL约束条件的不能取空值；
- 加了UNIQUE限制的属性不能取空值；
- 码属性不能取空值；

# 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）的算术运算结果为空值；
- 空值与另一个值（包括另一个空值）的比较运算结果为UNKNOWN；
- 三值逻辑的概念：T、F、U；

# 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

［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'  ;
```

选出了参加考试不及格的学生，不包括缺考的学生！

# 空值的算术运算、比较运算和逻辑运算

［例3.83］找出选修1号课程的不及格的学生及缺考的学生。

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

# 空值的算术运算、比较运算和逻辑运算

[例3.83] 找出选修1号课程的不及格的学生及缺考的学生。

```
SELECT   Sno

FROM SC

WHERE Cno=′ 1′ AND (Grade<60 OR Grade IS NULL);
```

# 第四章 结构查询语言SQL

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

## 目 录

# 视图

- 视图的特点：
  - 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，不会出现数据冗余；
  - 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；

# 视图

基于视图的操作：

- 查询
- 删除（只影响视图本身）
- 受限更新（直接影响基本表）
- 定义基于该视图的新视图

# 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，是否含有ORDER BY子句和DISTINCT短语，取决于具体系统的实现；
- WITH CHECK OPTION表示对视图进行更新、插入或删除操作时，要保证满足视图定义中的谓词条件（子查询条件表达式）。

# 1. 建立视图

- 组成视图的属性列名：全部省略或全部指定。但在下面三种情况必须明确指定组成视图的所有列名：

  1）某个目标列不是单纯的属性名，而是聚集函数或列表达式；

  2）多表连接时选出了几个同名列作为视图的字段；

  3）需要在视图中为某个列启用新的更合适的名字。

- DBMS执行CREATE VIEW语句时只是把视图的定义存入数据字典，并不执行其中的SELECT语句。

# 行列子集视图

- 若一个视图由单个基本表导出，只是去掉某些行列，但保留主码，这类视图称为行列子集视图。

[例1] 建立信息系学生的视图，并要求透过该视图进行的更新操作只涉及信息系学生。

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

# 基于多个基表的视图

[例2] 建立信息系统选修了1号课程的学生视图。

```
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';
```

# 基于视图的视图

［例3］ 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；
```

# 带表达式的视图

[例4] 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
```

- 设置一些派生属性列，也称为虚拟列，例如Sbirth ；
- 带表达式的视图必须明确定义组成视图的各个属性列名；

# 建立分组视图

[例5] 将学生的学号及他的平均成绩定义为一个视图，假设SC表中“成绩”列Grade为数字型。

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

# 不指定属性列

[例6]将Student表中所有女生记录定义为一个视图

```
CREATE VIEW
        F_Student1(F_sno，name，sex，age，dept)
  AS
  SELECT *
  FROM   Student
  WHERE Ssex=’女’；
```

缺点：修改基表Student的结构后，Student表与F_Student1视图的映象关系被破坏，导致该视图不能正确工作。

# 2. 删除视图

DROP VIEW <视图名>[CASCADE];

- 该语句从数据字典中删除指定的视图定义；
- 由该视图导出的其他视图定义仍在数据字典中，但已不能使用，必须显式删除；
- 删除基表时，由该基表导出的所有视图定义都必须显式删除；

# 2. 删除视图

[例7]　删除视图BT_S：DROP VIEW BT_S;

删除视图IS_S1：DROP VIEW IS_S1;

拒绝执行；

级联删除：

DROP VIEW IS_S1 CASCADE;

# 3. 查询视图

- 从用户角度：查询视图与查询基本表相同
- DBMS实现视图查询的方法
  - 实体化视图（View Materialization）
    - 有效性检查：检查所查询的视图是否存在
    - 执行视图定义，将视图临时实体化，生成临时表
    - 查询视图转换为查询临时表
    - 查询完毕删除被实体化的视图(临时表)

# 3. 查询视图

- 视图消解法（View Resolution）
  - 进行有效性检查，检查查询的表、视图等是否存在。如果存在，则从数据字典中取出视图的定义；
  - 把视图定义中的子查询与用户的查询结合起来，转换成等价的对基本表的查询；
  - 执行修正后的查询；

# 3. 查询视图

[例8] 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;
```

IS_Student视图的定义（视图定义例1）：

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

# 3. 查询视图

- 视图消解法

转换后的查询语句为：

```
SELECT   Sno，Sage

FROM   Student

WHERE   Sdept= 'IS'   AND   Sage<20;
```

# 多表查询

[例9] 查询信息系统选修了1号课程的学生

```
SELECT    IS_Student.Sno, Sname

FROM      IS_Student, SC

WHERE     IS_Student.Sno = SC.Sno AND SC.Cno= '1';
```

# 查询视图（续）

[例10]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视图定义：

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

# 查询视图（续）

结合：

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

查询出现语法错误

正确：

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

- 视图消解法的局限
  - 有些情况下，视图消解法不能生成正确查询，DBMS会限制这类查询

# 4. 更新视图

- 用户角度：更新视图与更新基本表相同
- DBMS实现视图更新的方法
  - 视图实体化法（View Materialization）
  - 视图消解法（View Resolution）
- 指定WITH CHECK OPTION子句后
  - DBMS在更新视图时会进行检查，防止用户通过视图对不属于视图范围内的基本表数据进行更新。

# 4. 更新视图

[例11] 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

[例12] 向信息系学生视图IS_S中插入一个新的学生记录：201215129，赵新，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(’201215129’,’赵新’，20,’IS’);
```

# 更新视图

[例13] 删除视图CS_S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(对两类方法均如此)

例：视图S_G为不可更新视图。

对于如下更新语句：

```
UPDATE   S_G
SET      Gavg=90
WHERE    Sno= '95001';
```

注意：无论实体化法还是消解法都无法将其转换成对基本表SC的更新

# 实际系统对视图更新的限制

- 允许对行列子集视图进行更新；
- 对其他类型视图的更新不同系统有不同限制

  例如 DB2对视图更新的限制（P127）

# 1. 视图能够简化用户的操作

- 当视图中数据不是直接来自基本表时，定义视图能够简化用户的操作。
  - 基于多张表连接形成的视图
  - 基于复杂嵌套查询的视图
  - 含导出属性的视图

# 2. 视图使用户能以多种角度看待同一数据

- 视图机制能使不同用户以不同方式看待同一数据，适应不同种类的用户

  数据库共享的需要

# 3.视图对重构数据库提供了一定程度的逻辑独立性

例：数据库逻辑结构发生改变

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)

# 3.视图对重构数据库提供了一定程度的逻辑独立性

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;
```

# 4. 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；
- 通过WITH CHECK OPTION对关键数据定义操作时间限制；

# 5. 适当的视图能够更清晰表达查询

- 复杂过程简单化；

例如：对每个同学找出他获得最高成绩的课程号；

```
CREATE VIEW VMGRADE
AS
SELECT Sno, MAX(Grade) Mgrade
FROM SC
GROUP BY Sno;
```

再用如下查询语句完成查询：

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

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

# THANK YOU !

饮水思源 爱国荣校