



数据库系统原理

Database System Principle

第三章 关系数据库标准语言

SQL

华中科技大学计算机学院

第三章 关系数据库标准语言SQL

3.1 SQL概述

3.2 学生-课程数据库

3.3 数据定义

3.4 数据查询

3.5 数据更新

3.6 视图

Do You Know SQL?

❖ Explain the difference between:

```
SELECT b  
FROM R  
WHERE a < 10 OR a >= 10;
```

```
SELECT b  
FROM R;
```

a	b
5	20
10	30
20	40
...	...

R

Do You Know SQL?

❖ Explain the difference between:

```
SELECT b  
FROM R  
WHERE a < 10 OR a >= 10;
```

```
SELECT b  
FROM R;
```

a	b
5	20
10	30
20	40
null	50

R

And How About These?

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

a	b
1	2
3	4

R

b	c
2	5
4	6

S

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

IN is a Predicate About R's Tuples

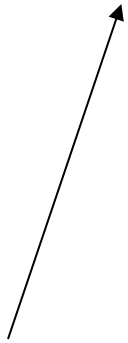
```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

Two 2's



(SELECT b FROM S);

One loop, over
the tuples of R



a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) satisfies
the condition;
1 is output once.

This Query Pairs Tuples from R, S

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

Double loop, over
the tuples of R and S

a	b	b	c
1	2	2	5
3	4	2	6

R S

(1,2) with (2,5)
and (1,2) with
(2,6) both satisfy
the condition;
1 is output twice.

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

❖ SQL是一个通用的、功能极强的关系数据库语言

SQL概述（续）

❖ **3.1.1 SQL 的产生与发展**

❖ **3.1.2 SQL的特点**

❖ **3.1.3 SQL的基本概念**

SQL标准的进展过程

标准	大致页数	发布日期
■ SQL/86		1986.10
■ SQL/89 (FIPS 127-1)	120页	1989年
■ SQL/92	622页	1992年
■ SQL99 (SQL3)	1700页	1999年
■ SQL2003		2003年
■ SQL2008		2008年
■ SQL2011		2010年

两大标准化组织:

ANSI (American National Standard Institute)

ISO (International Organization for Standardization)

SQL的产生

- ❖ 1974年，IBM的Ray Boyce和Don Chamberlin将Codd关系数据库的12条准则的数学定义以简单的关键字语法表现出来，里程碑式地提出了SQL (Structured Query Language)语言。
- ❖ 1986年，ANSI把SQL作为关系数据库语言的美国标准，同年公布了标准SQL文本。目前SQL标准有3个主要的版本。

SQL-89

- ❖ 基本SQL定义是ANSIX3135-89, “Database Language-SQL with Integrity Enhancement”[ANSI89, 一般叫做SQL-89]。
- ❖ SQL-89描述了模式定义、数据操作和事务处理。
- ❖ SQL-89和随后的ANSIX3168-1989, “Database Language-Embedded SQL”构成了第一代SQL标准。

SQL92

- ❖ ANSI X3.135-1992 [ANSI92] 描述了一种增强功能的SQL，现在叫做SQL-92标准。
- ❖ SQL-92 包括模式操作，动态创建和SQL语句动态执行、网络环境支持等增强特性。

SQL3

- ❖ 在完成SQL-92标准后，ANSI和ISO即开始合作开发SQL3标准。
- ❖ SQL3的主要特点在于抽象数据类型的支持，为新一代对象关系数据库提供了标准。

3.1 SQL概述

❖ 3.1.1 SQL 的产生与发展

❖ 3.1.2 SQL的特点

❖ 3.1.3 SQL的基本概念

3.1.2 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

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**”，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.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.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.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 3.1 SQL 语言的动词

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

3.1 SQL概述

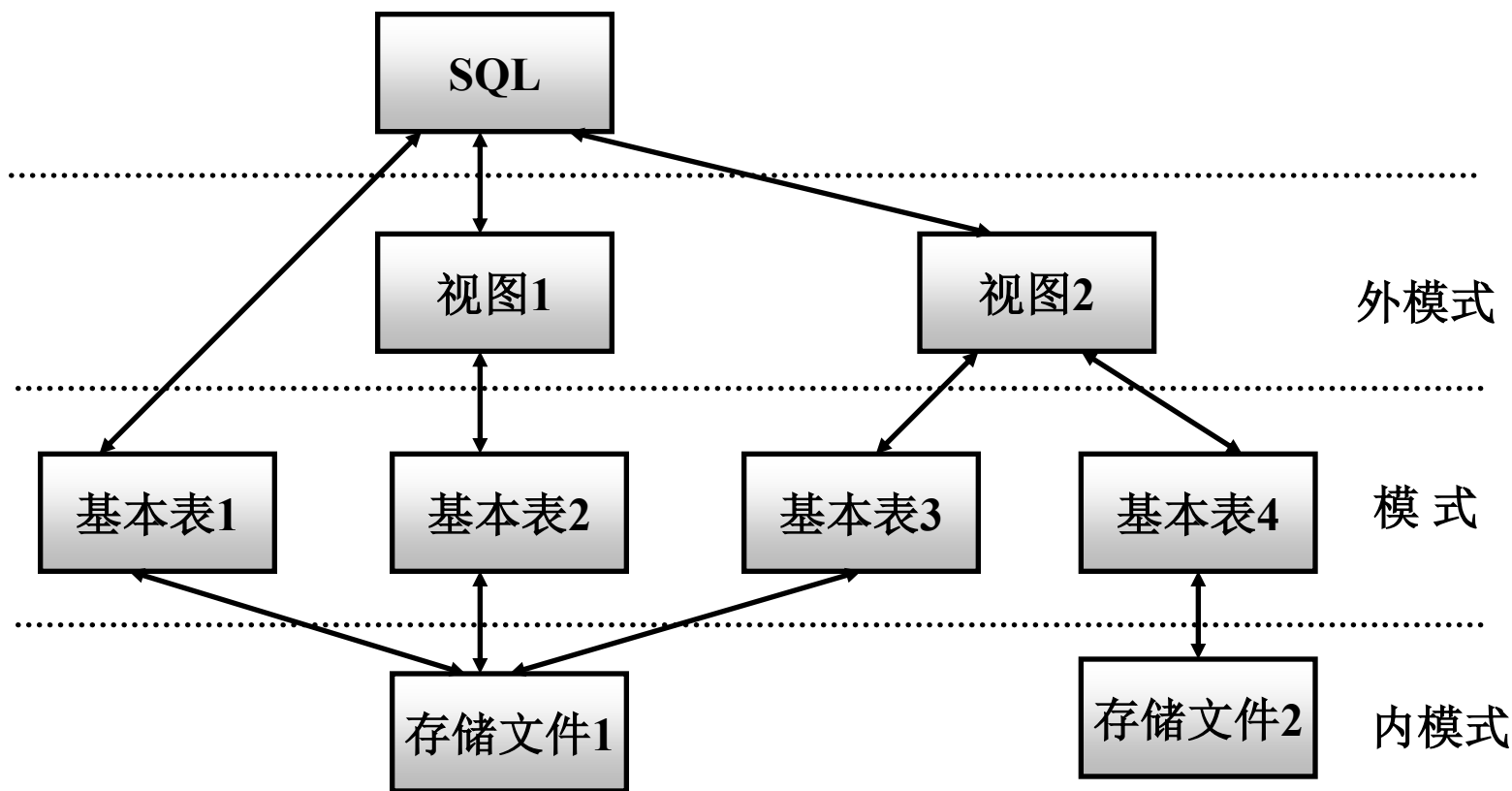
❖ 3.1.1 SQL 的产生与发展

❖ 3.1.2 SQL的特点

❖ 3.1.3 SQL的基本概念

SQL的基本概念（续）

SQL支持关系数据库三级模式结构



SQL的基本概念（续）

❖ 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

第三章 关系数据库标准语言SQL

3.1 SQL概述

3.2 学生-课程数据库

3.3 数据定义

3.4 数据查询

3.5 数据更新

3.6 视图

3.2 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)

Student表

学 号 Sno	姓 名 Sname	性 别 Ssex	年 龄 Sage	所 在 系 Sdept
200215121	李勇	男	20	CS
200215122	刘晨	女	19	CS
200215123	王敏	女	18	MA
200515125	张立	男	19	IS

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

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

第三章 关系数据库标准语言SQL

3.1 SQL概述

3.2 学生-课程数据库

3.3 数据定义

3.4 数据查询

3.5 数据更新

3.6 视图

3.3 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 3.2 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	

3.3 数据定义

❖ 3.3.1 模式的定义与删除

❖ 3.3.2 基本表的定义、删除与修改

❖ 3.3.3 索引的建立与删除

3.3.1 模式的定义与删除

[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

[例2]CREATE SCHEMA AUTHORIZATION WANG;

<模式名>隐含为用户名WANG

- 如果没有指定<模式名>，那么<模式名>隐含为<用户名>

一、定义模式

- ❖ 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]

```
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。

二、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。

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

3.3 数据定义

- ❖ 3.3.1 模式的定义与删除
- ❖ 3.3.2 基本表的定义、删除与修改
- ❖ 3.3.3 索引的建立与删除

3.3.2 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>]) ;

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。

学生表Student

[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*/
```

```
Sname CHAR(20) UNIQUE, /* Sname取唯一值*/
```

```
Ssex CHAR(2),
```

```
Sage SMALLINT,
```

```
Sdept CHAR(20)
```

```
);
```

主码

课程表Course

[例6] 建立一个“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

学生选课表SC

[例7] 建立一个“学生选课”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*/  
);
```

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

二、数据类型

数据类型	含义
CHAR(n)	长度为n的定长字符串
VARCHAR(n)	最大长度为n的变长字符串
INT	长整数（也可以写作 INTEGER ）
SMALLINT	短整数
NUMERIC(p, d)	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
REAL	取决于机器精度的浮点数
Double Precision	取决于机器精度的双精度浮点数
FLOAT(n)	浮点数，精度至少为n位数字
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

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*/

- 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
- ❖ 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`

模式与表（续）

- ❖ 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。

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** <新列名> <数据类型> [完整性约束]]

[**DROP** <完整性约束名>]

[**ALTER COLUMN** <列名> <数据类型>];

修改基本表（续）

[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(缺省)。

- 欲删除的基本表不能被其他表的约束所引用
- 如果存在依赖该表的对象，则此表不能被删除

■ **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

删除基本表(续)

[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、触发器等也将被删除

3.3 数据定义

- ❖ 3.3.1 模式的定义与删除
- ❖ 3.3.2 基本表的定义、删除与修改
- ❖ 3.3.3 索引的建立与删除

3.3.3 索引的建立与删除

❖ 建立索引的目的：加快查询速度

❖ 谁可以建立索引

- DBA 或表的属主（即建立表的人）
- DBMS一般会建立以下列上的索引

PRIMARY KEY

UNIQUE

❖ 谁维护索引

DBMS自动完成

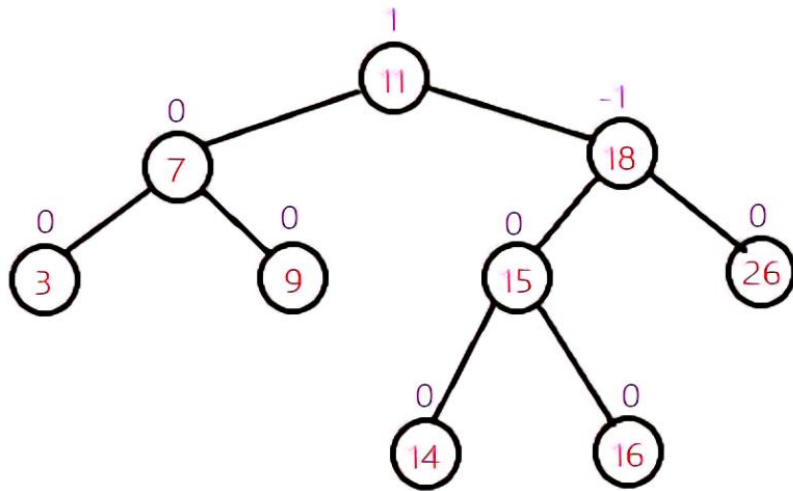
❖ 使用索引

DBMS自动选择是否使用索引以及使用哪些索引

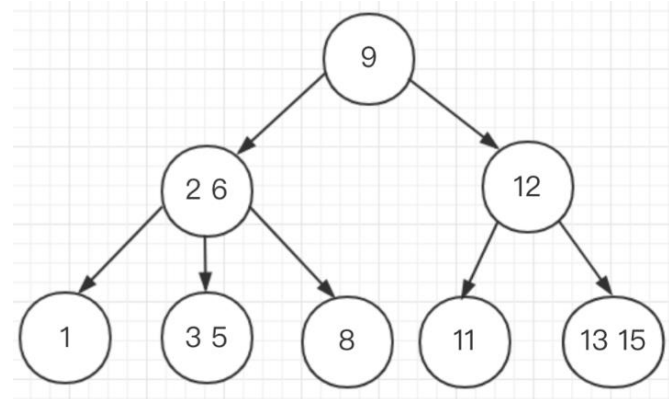
索引

- ❖ 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，则由具体的RDBMS来决定
- ❖ 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

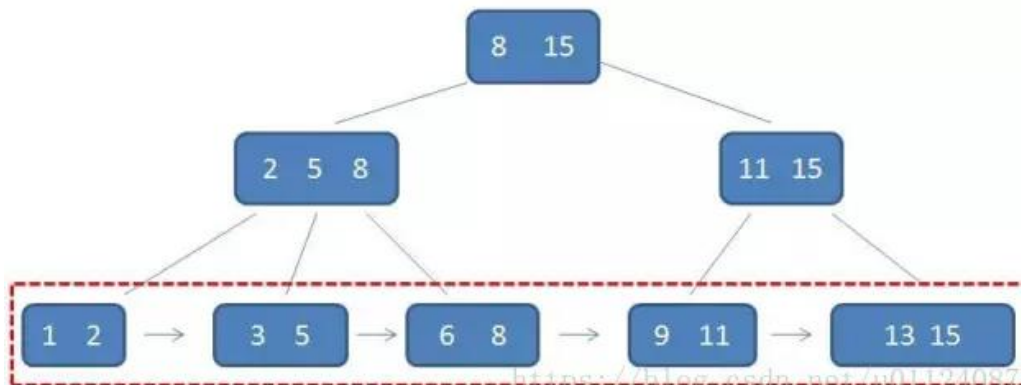
索引——B+树



平衡二叉树



B树



B+树

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>]]...);

注:

UNIQUE: 唯一性索引，不允许表中不同的行在索引列上取相同值。若已有相同值存在，则系统给出相关信息，不建此索引。

CLUSTER: 聚集索引，表中元组按索引项的值排序并物理地聚集在一起。

次序 (ASC/DESC): 索引表中索引值的排序次序，缺省为ASC。

建立索引（续）

[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

建立索引（续）

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

```
CREATE UNIQUE INDEX Stusno ON Student(Sno);  
CREATE UNIQUE INDEX Coucno ON Course(Cno);  
CREATE UNIQUE INDEX SCno ON SC(Sno ASC, Cno  
DESC);
```

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

二、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15] 删除Student表的Stusname索引

```
DROP INDEX Stusname;
```

索引的有关说明

- ❖ 可以动态地定义索引，即可以随时建立和删除索引
- ❖ 不允许用户在数据操作中引用索引，索引如何使用完全由系统决定
- ❖ 应该在使用频率高的、经常用于连接的列上建索引
- ❖ 一个表上可建多个索引
- ❖ 索引可以提高查询效率，但索引过多耗费空间，且降低了插入、删除、更新的效率

3.4 数据库的建立与撤消(*)

- 有的DBMS支持多库
 - 通常，数据库由包含数据的表集合和其它对象（如视图、索引、存储过程和触发器）组成。
- 建立一个新数据库

create database 数据库名
- 撤消一个数据库

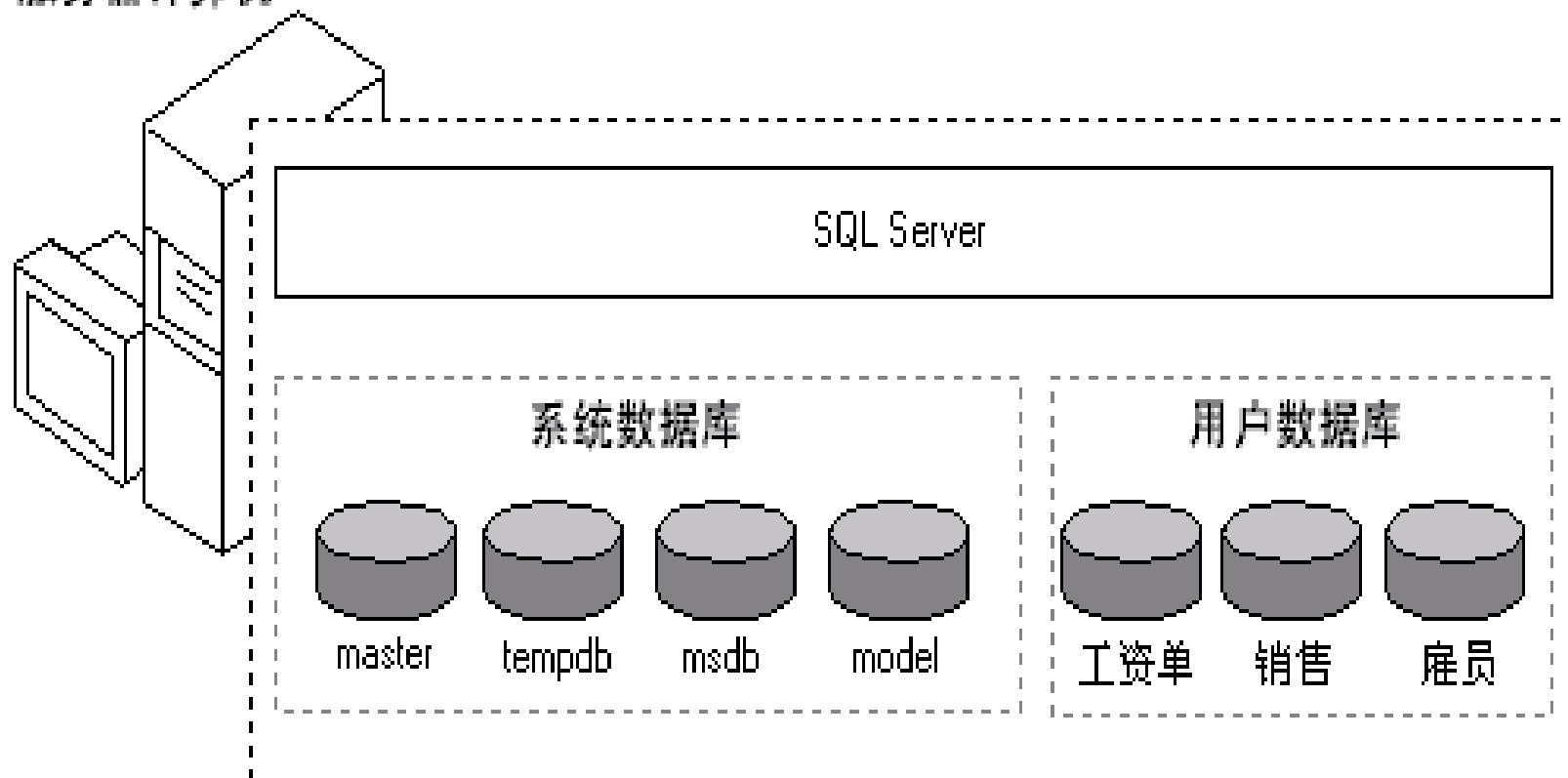
drop database 数据库名
- 指定当前数据库

database 数据库名

use 数据库名

示例 — SQL Server 数据库

服务器计算机



Microsoft SQL Server Management Studio

File Edit View Project Debug Tools Window Community Help

New Query

Execute

Object Explorer

Connect

(local) (SQL Server 10.50.1617 - panppl-THINK\panppl)\Databases\pptest\Tables

Name	Schema	Create Date	Policy Health State
System Tables			
Table_person	dbo		
student	dbo		
sc	dbo		

New Table...
Design
Select Top 1000 Rows
Edit Top 200 Rows
Script Table as
View Dependencies
Full-Text index
Storage
Policies
Facets
Start PowerShell
Reports
Rename
Delete
Refresh
Properties

Table_person

Schema: dbo
Create Date: 2013/4/22 21:23

Used (KB): 8
File Group: PRIMARY
Replicated: False

Properties

Current connection parameter

Aggregate Status

Connection
Elapsed time 00:00:00.436
Finish time 2017/3/29 1:18:00
Name (local)
Rows returned 1
Start time 2017/3/29 1:18:00
State Open

Connection

Connection (local) (panppl-THINK\panppl)

Connection Details

Connection 00:00:00.436
Connection 2017/3/29 1:18:00
Connection 1
Connection 2017/3/29 1:18:00
Connection Open
Display name (local)
Login name panppl-THINK\panppl
Server name (local)
Server version 10.50.1617
Session Trace
SPID 53

Output

Ready

第三章 关系数据库标准语言SQL

3.1 SQL概述

3.2 学生-课程数据库

3.3 数据定义

3.4 数据查询

3.5 数据更新

3.6 视图

3.4 数据查询

- ❖ 3.4.1 概述
- ❖ 3.4.2 简单查询
- ❖ 3.4.3 嵌套查询
- ❖ 3.4.4 集合查询

3.4.1 概述

SELECT

从数据库中检索行，并允许从一个或多个表中选择一个或多个行或列。虽然 SELECT 语句的完整语法较复杂，但是其主要的子句可归纳如下：

```
SELECT select_list
[ INTO new_table ]
FROM table_source
[ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

The diagram illustrates the components of a SELECT statement. Red arrows point from labels in yellow boxes to specific parts of the SQL syntax:

- 行条件子句** (Row condition clause) points to the `WHERE` clause.
- 分组子句** (Grouping clause) points to the `GROUP BY` clause.
- 组条件子句** (Group condition clause) points to the `HAVING` clause.
- 排序子句** (Sorting clause) points to the `ORDER BY` clause.

可以在查询之间使用 UNION 运算符，以将查询的结果组合成单个结果集。

3.4.2 简单查询

- ❖ 1. SQL数据查询基本结构
- ❖ 2. SELECT子句
- ❖ 3. 重复元组的处理
- ❖ 4. FROM子句
- ❖ 5. WHERE子句
- ❖ 6. 更名运算
- ❖ 7. 字符串操作
- ❖ 8. 元组显示顺序
- ❖ 9. 分组与聚集函数
- ❖ 10. 空值

3.4.2 简单查询（续）

❖ 1. SQL数据查询基本结构

SELECT A_1, A_2, \dots, A_n
FROM R_1, R_2, \dots, R_m
WHERE P

\Leftrightarrow

$\Pi_{A_1, A_2, \dots, A_n}(\sigma_P(R_1 \times R_2 \times \dots \times R_m))$

3.4.2 简单查询（续）

❖ 示例

Ex: 查询所有老师的姓名

```
SELECT pname FROM Prof
```


$$\Pi_{pname}(Prof)$$

3.4.2 简单查询（续）

❖ 2. SELECT子句

- 目标列形式
 - 列名, *, 算术表达式, 聚集函数

Ex 1: 查询所有老师的证件号, 姓名, 年龄

```
SELECT p#, pname, age  
FROM Prof
```

Ex 2: 查询所有老师的信息

```
SELECT * FROM Prof
```

Ex 3: 查询所有院系的信息

```
SELECT * FROM Dept
```

3.4.2 简单查询（续）

❖ 2. SELECT子句

- 目标列形式
 - 列名, *, 算术表达式, 聚集函数

Ex 4 : 查询所有老师的姓名及税后工资额

```
SELECT  pname, sal * 0.9  
FROM Prof
```

Ex 5: 查询所有老师的平均工资

```
SELECT  AVG(sal) FROM Prof
```

3.4.2 简单查询（续）

❖ 3. 重复元组的处理

■ 语法约束

- 缺省为保留重复元组，也可用关键字**ALL**显式指明。
- 若要去掉重复元组，可用关键字**DISTINCT**指明

Ex: 查询所有选修课程的学生

```
SELECT DISTINCT  sno  
FROM    SC
```

3.4.2 简单查询（续）

❖ 4. FROM 子句

■ 说明

- FROM 子句列出查询的对象表
- 当目标列取自多个表时，在不混淆的情况下可以不用显式指明来自哪个关系

–Ex: 查询职工的姓名、工资、系别

```
SELECT ALL pname , sal , dname  
FROM Prof , Dept  
WHERE Prof.DNO = Dept.DNO
```


3.4.2 简单查询（续）

- 连接查询说明

- 示例

- Agents(aid, aname,city,percent)
 - Customers(cid, cname,discount)
 - Prouducts(pid, pname,city,quantity,price)
 - Orders(ordno, cid,aid,pid,qty)

- Ex: 查询所有订货顾客及其供应商的姓名

$$\Pi_{\text{cname, aname}}((\Pi_{\text{cid, cname, aid}}(\text{Customers}) \\ | \times | \text{Orders}) | \times | \text{Agents})$$
$$\Pi_{\text{cname, aname}}(\sigma_{\text{Customers.cid} = \text{orders.cid} \wedge \text{Orders.aid} = \\ \text{Agents.aid}}((\text{Customers} \times \text{Orders}) \times \text{Agents}))$$

3.4.2 简单查询（续）

– 连接查询说明

- SELECT语句的执行步骤

$$\Pi_{\text{cname,aname}} (\sigma_{\text{Customers.cid} = \text{orders.cid} \wedge \text{Orders.aid} = \text{Agents.aid}} ((\text{Customers} \times \text{Orders}) \times \text{Agents}))$$

SELECT DISTINCT Customers.cname, Agents.aname

FROM Customers, Orders, Agents

WHERE Customers.cid = Orders.cid

AND Orders.aid = Agents.aid

3.4.2 简单查询（续）

❖ FROM 子句（续）

- Ex: 列出教授“哲学”课程的老师的教工号及姓名

SELECT Prof.pno , Prof.pname

FROM Prof , PC , Course

WHERE Prof.pno = PC.pno

AND PC.Cno = Course.cno

AND Course.cname = “哲学”

3.4.2 简单查询（续）

❖ 5. WHERE 子句

■ 语法成分

- 比较运算符

<、<=、>、>=、=、<>

- 逻辑运算符

AND, OR, NOT

- BETWEEN 条件

– 判断表达式的值是否在某范围内

- IN, LIKE, IS NULL, ...

3.4.2 简单查询（续）

❖ 5. WHERE 子句

Ex 1: 查询工资低于2000的老师的姓名、工资、系别

```
SELECT      pname , sal , dname  
FROM        Prof , Dept  
WHERE        sal < 2000  
              AND Prof.dno = Dept.dno
```

Ex 2: 查询工资在1000-2000之间的老师姓名

```
SELECT      pname  
FROM        Prof  
WHERE      sal BETWEEN 1000 AND 2000
```

3.4.2 简单查询（续）

❖ 6. 更名运算

- 格式

old_name AS new_name

为关系和属性重新命名，可出现在SELECT和FROM子句中

注：AS可选

- 属性更名

Ex1 :给出所有老师的姓名、所纳税额及税后工资额

```
SELECT pname, sal*0.05 AS tax,  
          sal * 0.95 AS incoming
```

```
FROM Prof
```

Ex 2: 查询工资比所在系主任工资高的老师姓名及工资

DEPT(D# , DNAME , DEAN)

PROF(P# , PNAME, AGE, D# , SAL)

S(S# , SNAME , SEX , AGE , D#)

COURSE(C# , CN , PC# , CREDIT)

SC(S# , C# , SCORE)

TEACH(P# , C#)

SQL Server Enterprise Manager

控制台(C) 窗口(W) 帮助(H)

1:控制台根目录\Microsoft SQL Servers\SQL Server 组\HUST-26M1ZYR73T\WEBKITTEN (Windows NT)\数据库\test1\表

操作(A) 查看(V) 工具(T)

树

- 控制台根目录
 - Microsoft SQL Servers
 - SQL Server 组
 - HUST-26M1ZYR73T\WEBKITTEN (W
 - 数据库
 - ems
 - master
 - model
 - msdb
 - Northwind
 - pubs
 - school
 - school2
 - school3
 - shop
 - tempdb
 - test1
 - 关系图
 - 表
 - 视图
 - 存储过程
 - 用户
 - 角色
 - 规则
 - 默认
 - 用户定义的数
 - 用户定义的函
 - 全文目录

表 22 个项目

名称	所有者	类型	创建日期
dept	dbo	用户	2003-11-10 21:41:48
dtproperties	dbo	系统	2003-11-10 21:40:53
prof	dbo	用户	2003-11-10 21:43:31
syscolumns	dbo	系统	2000-8-6 1:29:12

2:表"dept"中的数据, 位置是"test1"中、"HUST-26M1ZYR73T\WEBKITTEN"上

d#	dname	dean
01	computer	卢炎生
02	automatic	王红卫
03	english	张力
04	sigal	王炎坤
05	chinese	胡铁花

SQL Server Enterprise Manager

控制台(C) 窗口(W) 帮助(H)

1:控制台根目录\Microsoft SQL Servers\SQL Server 组\HUST-26M1ZYR73T\WEBKITTEN (Windows NT)\数据库\test1\表

操作(A) 查看(V) 工具(T)

树

- 控制台根目录
 - Microsoft SQL Servers
 - SQL Server 组
 - HUST-26M1ZYR73T\WEBKITTEN (W
 - 数据库
 - ems
 - master
 - model
 - msdb
 - Northwind
 - pubs
 - school
 - school2
 - school3
 - shop
 - tempdb
 - test1
 - 关系图
 - 表
 - 视图
 - 存储过程
 - 用户
 - 角色
 - 规则
 - 默认
 - 用户定义的
 - 用户定义的
 - 全文目录

表 22 个项目

名称	所有者	类型	创建日期
dept	dbo	用户	2003-11-10 21:41:48
dtproperties	dbo	系统	2003-11-10 21:40:53
prof	dbo	用户	2003-11-10 21:43:31
syscolumns	dbo	系统	2000-8-6 1:29:12
syscomments	dbo	系统	2000-8-6 1:29:12
sysdepends	dbo	系统	2000-8-6 1:29:12
sysfilegroups	dbo	系统	2000-8-6 1:29:12

2:表"prof"中的数据, 位置是"test1"中、"HUST-26M1ZYR73T\WEBKITTEN"上

p#	pname	age	d#	sal
p0001	卢炎生	53	01	3000
p0002	徐丽萍	38	01	2500
p0003	殷贤亮	52	01	4000
p0004	赵栋	25	01	2000
p0005	胡侃	36	01	5000
p0006	盛理智	58	02	3000
p0007	王红卫	38	02	4000
p0008	代额动	42	02	4500
p0009	张力	45	03	2600
p0010	罗杰	34	03	2300

SQL 查询分析器

文件(F) 编辑(E) 查询(Q) 工具(T) 窗口(W) 帮助(H)



查询 — HUST-26M1ZYR73T\WEBKITTEN.test1.HUST-26M1ZYR73T\Administrator — 无标题1*

```
SELECT      P1.pname,P1.sal
FROM        Prof AS P1,Prof AS P2,Dept
WHERE       P1.d# = Dept.d#
            AND      Dept.dean = P2.pname
            AND      P1.sal > P2.sal
```

	pname	sal	
1	殷贤亮	4000.0000	
2	胡侃	5000.0000	
3	代额动	4500.0000	

网格 消息

批查询完成。

HUST-26M1ZYR73T\WEBKITTEN

HUST-26M1ZYR73T\Admi

3.4.2 简单查询（续）

❖ 7. 字符串操作

■ 语法格式

- 列名 **[NOT] LIKE** ‘<匹配串>’ [**ESCAPE** ‘<换码字符>’]
- 找出满足给定匹配条件的字符串

■ 匹配规则

- “**%**”：匹配零个或多个字符
- “**_**”：匹配任意单个字符
- **ESCAPE**：定义转义字符，以去掉特殊字符的特定含义，使其被作为普通字符看待

如**ESCAPE** “\”，定义 \ 作为转义字符，则可用 \% 去匹配 %，用 _ 去匹配 _

3.4.2 简单查询（续）

❖ 7. 字符串操作

- Ex 1: 查询姓名以“张”打头的教师的所有信息

```
SELECT *  
FROM Prof  
WHERE pname LIKE '张%'
```

- Ex 2: 查询姓‘司马’且全名为四个汉字的学生的姓名

```
SELECT sname  
FROM S  
WHERE sname LIKE '司马_ _ _ _'
```

注意：一个汉字占两个字符

3.4.2 简单查询（续）

❖ 7. 字符串操作

- Ex 3: 列出名称中含有4个字符以上，且倒数第3个字符是d，倒数第2个字符是_的所有信息

```
SELECT  *  
FROM    Prof  
WHERE   pname LIKE '%_d\__' ESCAPE '\'
```

3.4.2 简单查询（续）

❖ 8. 元组显示顺序

■ 命令

- **ORDER BY** 列名 [ASC | **DESC**]

Ex 1: 按系名升序列出老师姓名, 所在系名, 同一系中老师按姓名降序排列

SELECT *dname, pname*

FROM *Prof, Dept*

WHERE *Prof.d# = Dept.d#*

ORDER BY *dname ASC, pname DESC*

SQL 查询分析器

文件(F) 编辑(E) 查询(Q) 工具(T) 窗口(W) 帮助(H)

test1

查询 — HUST-26M1ZYR73T\WEBKITTEN.test1.HUST-26M1ZYR73T\Administrator — 无标题

```
SELECT    dname, pname
FROM      Prof, Dept
WHERE     Prof.d# = Dept.d#
ORDER BY  dname ASC, pname DESC
```

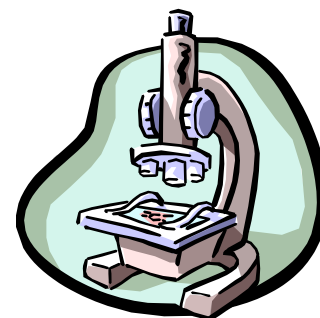
	dname	pname
1	automatic	王红卫
2	automatic	盛理智
3	automatic	代额动
4	computer	赵栋
5	computer	殷贤亮
6	computer	徐丽萍
7	computer	卢炎生
8	computer	胡侃
9	english	张力
10	english	罗杰

3.4.2 简单查询（续）

❖ 9. 分组与聚集函数

■ 聚集函数

- **COUNT** ([DISTINCT] A) / **COUNT**([DISTINCT]*)
- **SUM** ([DISTINCT] A)
- **AVG** ([DISTINCT] A)
- **MAX** (A)
- **MIN** (A)



3.4.2 简单查询（续）

Ex 1: 查询教师的平均工资，最低工资，最高工资

```
SELECT AVG(sal), MIN(sal), MAX(sal)
FROM Prof
```

查询 — HUST-26M1ZYR73T\WEBKITTEN.test1.

```
SELECT AVG(sal), MIN(sal), MAX(sal)
FROM Prof
```

	(无列名)	(无列名)	(无列名)
1	3290.0000	2000.0000	5000.0000

查询 — HUST-26M1ZYR73T\WEBKITTEN.test1.HUST-26M1ZYR73T\Administrator — 无标题1*

```
SELECT AVG(sal) as 平均工资, MIN(sal) as 最低工资, MAX(sal) as 最高工资
FROM Prof
```

	平均工资	最低工资	最高工资
1	3290.0000	2000.0000	5000.0000

3.4.2 简单查询（续）

❖ 9. 分组与聚集函数

■ 分组命令

GROUP BY 列名 [**HAVING** 条件表达式]

- **GROUP BY** + *Group-list*

- 将表中的元组按指定列上值相等的原则分组，然后在每一分组上使用聚集函数，得到单一值

- **HAVING** + *Group-qualification*

- 对分组进行选择，只将聚集函数作用到满足条件的分组上

❖ 9.分组与聚集函数

SC(S# , C# , SCORE)

Ex 3: 列出每个学生的平均成绩

SELECT S#, **AVG**(score)

FROM SC

GROUP BY S#

查询 — HUST-26M1ZYR73T\WEB		
<pre>SELECT s#, AVG(score) FROM SC GROUP BY s#</pre>		
	s#	(无列名)
1	s1	90
2	s2	85
3	s3	92

S#	C#	score
s1	c1	84
s1	c2	90
s1	c3	96
s2	c1	80
s2	c2	90
s3	c2	96
s3	c3	88

❖ 9.分组与聚集函数

SC(S# , C# , SCORE)

– Ex 4: 列出每门课程的平均成绩

SELECT C#, **AVG**(score)

FROM SC

GROUP BY C#

查询 — HUST-26M1ZYR73T\WEBKITT

```
SELECT c#, AVG(score)
FROM SC
GROUP BY c#
```

	c#	(无列名)
1	c1	82
2	c2	92
3	c3	92

S#	C#	score
s1	c1	84
s1	c2	90
s1	c3	96
s2	c1	80
s2	c2	90
s3	c2	96
s3	c3	88

82

92

92

❖ 9.分组与聚集函数

■ Having

- 可以针对聚集函数的结果值进行筛选，它是作用于分组计算的结果集
- 跟在**Group By**子句的后面

例：列出具有两门（含）以上课程在90分以上的学生的学号、90分以上的课目数。

```
Select s#, count(c#)
From SC
Where score >=90
Group By s#
Having count(c#) >= 2
```

查询 — HUST-26M1ZYR73T\WEBK			
Select s#, count (s#) From SC Where score >=90 Group By s# Having count (s#) >= 2			
◀			
	s#	(无列名)	
1	s1	2	

3.4.2 简单查询（续）

❖ 9. 分组与聚集函数

- Ex 5: 查询Prof表中各系的老师的最高、最低、平均工资

```
SELECT d#, MAX(sal), MIN(sal), AVG(sal)  
FROM Prof  
GROUP BY d#
```

3.4.2 简单查询（续）

❖ 10. 空值

- 格式

IS [NOT] NULL

- 测试指定列的值是否为空值
- = NULL 

- Ex 1: 找出年龄值为空的老师姓名

```
SELECT  pname  
FROM    Prof  
WHERE age IS NULL
```




■ 注意事项

- 除is [not] null之外，空值不满足任何查找条件
- 如果null参与算术运算，则该算术表达式的值为null
- 如果null参与比较运算，则结果可视为false，在SQL-92中可看成unknown
- 如果null参与聚集运算，则除count(*)之外其它聚集函数都忽略null

3.4.2 简单查询（续）

SC

S#	C#	score
s1	c1	80
s1	c2	90
s1	c3	95
s2	c1	85
s2	c2	
s3	c2	

❖ 10. 空值

■ 说明

SELECT SUM (score)
FROM SC

350

SELECT COUNT(*)
FROM SC

6

SELECT COUNT(score)
FROM SC

4

SC

S#	C#	score
s1	c1	80
s1	c2	90
s1	c3	95
s2	c1	85
s2	c2	null
s3	c2	null

NULL Values

- ❖ Tuples in SQL relations can have NULL as a value for one or more components.
- ❖ Meaning depends on context. Two common cases:
 - *Missing value* : e.g., we know Joe's Bar has some address, but we don't know what it is.
 - *Inapplicable* : e.g., the value of attribute *spouse* for an unmarried person.

Comparing NULL's to Values

- ❖ The logic of conditions in SQL is really 3-valued logic: TRUE, FALSE, UNKNOWN.
- ❖ Comparing any value (including NULL itself) with NULL yields UNKNOWN.
- ❖ A tuple is in a query answer iff the WHERE clause is TRUE (not FALSE or UNKNOWN).

Three-Valued Logic

❖ To understand how AND, OR, and NOT work in 3-valued logic, think of TRUE = 1, FALSE = 0, and UNKNOWN = $\frac{1}{2}$.

❖ AND = MIN; OR = MAX, NOT(x) = $1-x$.

❖ Example:

TRUE AND (FALSE OR NOT(UNKNOWN)) =
MIN(1, MAX(0, (1 - $\frac{1}{2}$))) = MIN(1, MAX(0, $\frac{1}{2}$))
= MIN(1, $\frac{1}{2}$) = $\frac{1}{2}$.

Surprising Example

❖ From the following **SC** relation:

SC

S#	C#	Score
S1	C2	NULL

SELECT S#

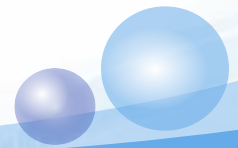
FROM SC

WHERE Score < 80 OR Score >= 80;

← UNKNOWN →

← UNKNOWN →

← UNKNOWN →



Reason: 2-Valued Laws \neq 3-Valued Laws

- ❖ Some common laws, like commutativity of AND, hold in 3-valued logic.
- ❖ But not others, e.g., the *law of the excluded middle* (排中律): $p \text{ OR NOT } p = \text{TRUE}$.
 - When $p = \text{UNKNOWN}$, the left side is $\text{MAX}(\frac{1}{2}, (1 - \frac{1}{2})) = \frac{1}{2} \neq 1$.

3.4.3 嵌套查询

- ❖ 1. 概念
- ❖ 2. 集合成员资格 (IN)
- ❖ 3. 集合之间的比较 (比较运算符)
- ❖ 4. 相关嵌套查询 (EXISTS)

3.4.3 嵌套查询(续)

❖ 1. 嵌套查询概念

- 子查询是嵌套在另一查询中的 **Select-From-Where** 表达式
- 可以用多个简单查询构成复杂查询，以增强SQL的查询能力
- 子查询中不能使用 **Order By** 子句，Order By子句只能对最终查询结果进行排序
- 相关概念：
 - 查询块
 - 嵌套查询
 - 父查询
 - 子查询

3.4.3 嵌套查询(续)

❖ 2. 集合成员资格 (IN)

■ 格式

表达式 [**NOT**] **IN** (子查询)

– 判断表达式的值是否在子查询的结果中

■ 示例

– 列出张军和王红同学的所有信息

```
select   *  
from     S  
where    SNAME in  
          ( “张军”, “王红” )
```

3.4.3 嵌套查询(续)

- 选修了001号课程的学生们的学号及姓名

```
select  SNO, SNAME  
from    S  
where   SNO in  
          (select  SNO  
             from    SC  
             where  CNO = 001 )
```

3.4.3 嵌套查询(续)

- 列出选修了001号和002号课程的学生们的学号

select SNO

from SC

where SC.CNO = 001

and SNO ***in***

(***select*** SNO

from SC

where CNO = 002)

3.4.3 嵌套查询(续)

Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

例：查询与“刘晨”在同一个系学习的学生的学号、姓名

3.4.3 嵌套查询(续)

Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

Sc(sno, cno, grade)

①确定“刘晨”所在系名。

```
SELECT Sdept
FROM Student
WHERE Sname='刘晨'
```

结果为:

Sdept

IS

②查找所有在IS系学习的学生学号、姓名。

```
SELECT Sno, Sname, Sdept
FROM Student
WHERE Sdept='IS'
```

结果为:

Sno	Sname
95001	刘晨
95004	张立



Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

③查询与“刘晨”在同一个系学习的学生的学号、姓名。

SELECT Sno, Sname

FROM Student

WHERE Sdept IN

(SELECT Sdept

FROM Student

WHERE Sname=' 刘晨 ')

Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

例：查询选修了课程名为‘信息系统’的学生学号和姓名。

完成此查询的基本思路是：

- ①首先在**Course**关系中找到‘信息系统’课程的课程号**Cno**。
- ②然后在**SC**关系中找到**Cno**等于第一步给出的**Cno**集合中某个元素的**Sno**。
- ③最后在**Student**关系选出**Sno**等于第二步中求出**Sno**集合中某个元素的元组，取出**Sno**和**Sname**送入结果表列。

Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

相应的SQL语句:

```
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' )) Database System Principle 114
```



Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

本查询同样可以用连接查询实现：

```
SELECT Student.Sno, Sname
```

```
FROM Student, SC, Course
```

```
WHERE Student.Sno=SC.Sno AND
```

```
SC.Cno=Course.Cno AND
```

```
Course.Cname = '信息系统'
```

Remember These Two Queries?

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

IN is a Predicate About R's Tuples

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

Two 2's

One loop, over
the tuples of R

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) satisfies
the condition;
1 is output once.

This Query Pairs Tuples from R, S

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

Double loop, over
the tuples of R and S

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) with (2,5)
and (1,2) with
(2,6) both satisfy
the condition;
1 is output twice.

3.4.3 嵌套查询(续)

● IN的用法

- 1、IN（值1, 值2, 。 。 。 ）
- 2、IN（单列多行子查询）
- 3、IN（多列子查询）特殊用法

```
1  -- 查询和SCOTT同部门同职位的员工信息
2  SELECT * FROM EMP WHERE (DEPTNO, JOB) IN(SELECT DEPTNO, JOB FROM EMP WHERE ENAME='SCOTT');
3  -- 查询每个部门工资最低的员工信息
4  SELECT * FROM EMP WHERE (DEPTNO, SAL) IN(SELECT DEPTNO, MIN(SAL) FROM EMP GROUP BY DEPTNO);
```

EMP

3.4.3 嵌套查询(续)

❖ 3. 集合之间的比较

- 带有比较运算符的子查询
 - 指父查询与子查询之间用比较运算符进行连接。
 - 当用户能确切知道内层查询返回的是单值时，可以用>、<、=、>=、<=、!=或<>等比较运算符。

3.4.3 嵌套查询(续)

3. 集合之间的比较

Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

Ex 2: 查询与王红在同一个系学习的同学的姓名

```
SELECT sname  
FROM Student  
WHERE Sdept = ( SELECT sdept  
                  FROM Student  
                  WHERE sname = '王红' )
```


3.4.3 嵌套查询(续)

❖ 3. 集合之间的比较

- 带ANY或ALL谓词的子查询
 - 表达式 比较运算符 θ **ANY** (子查询)
 - 表达式的值至少与子查询结果中的一个值相比满足比较运算符 θ
 - 表达式 比较运算符 θ **ALL** (子查询)
 - 表达式的值与子查询结果中的所有值相比都满足比较运算符 θ

3.4.3 嵌套查询(续)

❖ 多行比较运算符：ANY（任意一个）、ALL(所有)、IN 存在。ANY、ALL要配合单行比较符使用。

> ANY 大于最小的

> ALL 大于最大的

< ANY 小于最大的

< ALL 小于所有的

= ANY IN

= ALL ~~错~~ (没有这个写法)

3.4.3 嵌套查询(续)

❖ 3. 集合之间的比较

Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

例：查询其他系中比IS系任一学生年龄小的学生名单。

```
SELECT Sname, Sage
FROM Student
WHERE Sage<ANY
    (SELECT Sage
     FROM Student
     WHERE Sdept=' IS')
AND Sdept<>'IS'
ORDER BY Sage DESC;
```

3.4.3 嵌套查询(续)

❖ 3. 集合之间的比较

Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

例：查询其他系中比IS系所有学生年龄都小的学生名单。

SELECT Sname, Sage

FROM Student

WHERE Sage < ALL

(SELECT Sage

FROM Student

WHERE Sdept='IS')

AND Sdept <> 'IS'

ORDER BY Sage DESC;

3.4.3 嵌套查询(续)



❖ 3. 集合之间的比较

Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

Ex 5: 查询平均成绩最高的学生的学号

```
SELECT sno
FROM SC
GROUP BY sno
HAVING AVG(grade) >= ALL (SELECT AVG(grade)
FROM SC
GROUP BY sno)
```

3.4.3 嵌套查询(续)

❖ 4. 相关嵌套查询

■ 概念

- 不相关子查询

- 相关子查询

- 内层子查询的条件引用外层主查询的某些属性
- 在**Exists**运算中，**更多地**涉及到相关子查询的使用

3.4.3 嵌套查询(续)

❖ 4. 相关嵌套查询(EXISTS)

- [NOT] EXISTS （子查询）

- 说明

- 判断子查询的结果集合中是否有任何元组存在

3.4.3 嵌套查询(续)

- **Exists + 子查询**可用以测试该子查询的结果是否有元组
- 帶有**Exists**的子查询不返回任何数据，只产生 True/False
- 当子查询的结果集含有元组时**Exists**为True
- 当子查询的结果集不含有任何元组时**Exists**为False
- 由于不关心子查询的具体内容，因此用 Select *

3.4.3 嵌套查询(续)

❖ 4. 相关嵌套查询(EXISTS)


Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

例：查询所有选修了1号课程的学生姓名。

查询所有选修了1号课程的学生姓名涉及Student关系和SC关系，可以在Student关系中依次取每个元组的Sno值，用此Student.Sno值去检查SC关系，若SC中存在这样的元组，其SC.Sno值等于用来检查的Student.Sno值，并且其SC.Cno= '1'，则取此Student.Sname送入结果关系。



Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

SELECT Sname

FROM Student

WHERE EXISTS (SELECT *

FROM SC

WHERE Sno=Student.Sno AND Cno='1');

例：查询没有选修课程代号为 ‘03’ 的学生姓名

```
select   sname  
  
from    student  
  
where not exists  
  
    (select *  
     from sc  
     where sno = student.sno and  
           cno = '03')
```

3.4.3 嵌套查询(续)

❖ 4. 相关嵌套查询(EXISTS)

■ **IN**和**EXISTS**的比较

*IN*后的子查询与外层查询无关，每个子查询执行一次，而 *EXISTS*后的子查询与外层查询有关，需要执行多次。

3.4.4 集合查询

■ 命令

- 集合并: **union**
- 集合交: **intersect**
- 集合差: **except**

— 提示

集合操作自动去除重复元组，如果要保留重复元组的话，必须用**all**关键词指明

3.4.4 集合查询(续)

■ 示例

– 求选修了001或（且）002号课程的学生号

```
(select   SNO  
from     SC  
where    CNO = 001)
```

union (intersect)

```
(select   SNO  
from     SC  
where    CNO = 002)
```

– 求选修了001和002号而没有选003号课程的学生号

```
(select  SNO  
  from   SC  
 where  CNO = 001 or CNO = 002 )
```

except

```
(select  SNO  
  from   SC  
 where  CNO = 003)
```

Relational Algebra on Bags

- ❖ A *bag* (or *multiset*) is like a set, but an element may appear more than once.
- ❖ **Example:** $\{1,2,1,3\}$ is a bag.
- ❖ **Example:** $\{1,2,3\}$ is also a bag that happens to be a set.

Why Bags?

- ❖ SQL, the most important query language for relational databases, is actually a bag language.
- ❖ Some operations, like projection, are more efficient on bags than sets.

Operations on Bags

- ❖ **Selection** applies to each tuple, so its effect on bags is like its effect on sets.
- ❖ **Projection** also applies to each tuple, but as a bag operator, we do not eliminate duplicates.
- ❖ **Products** and **joins** are done on each pair of tuples, so duplicates in bags have no effect on how we operate.

Example: Bag Selection

R(

A,	B)
1	2
5	6
1	2

$\sigma_{A+B < 5} (R) =$

A	B
1	2
1	2

Example: Bag Projection

R(

A,	B
1	2
5	6
1	2

)

$\pi_A(R) =$

A
1
5
1

Example: Bag Product

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

R X S =

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	3	4
5	6	7	8
1	2	3	4
1	2	7	8

Example: Bag Theta-Join

R(

A,	B)
1	2
5	6
1	2

S(

B,	C)
3	4
7	8

$R \bowtie_{R.B < S.B} S =$

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	7	8
1	2	3	4
1	2	7	8

Bag Union

❖ An element appears in the union of two bags the sum of the number of times it appears in each bag.

❖ Example:

$$\{1,2,1\} \cup \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\}$$

Bag Intersection

❖ An element appears in the intersection of two bags the minimum of the number of times it appears in either.

❖ **Example:**

$$\{1,2,1,1\} \cap \{1,2,1,3\} = \{1,1,2\}.$$

Bag Difference

❖ An element appears in the difference $A - B$ of bags as many times as it appears in A , minus the number of times it appears in B .

- But never less than 0 times.

❖ Example:

$$\{1, 2, 1, 1\} - \{1, 2, 3\} = \{1, 1\}.$$

Beware: Bag Laws \neq Set Laws

- ❖ Some, but *not all* algebraic laws that hold for sets also hold for bags.
- ❖ **Example:** the commutative law for union ($R \cup S = S \cup R$) *does* hold for bags.
 - Since addition is commutative, adding the number of times x appears in R and S doesn't depend on the order of R and S .

Example: A Law That Fails

- ❖ Set union is *idempotent*, meaning that $S \cup S = S$.
- ❖ However, for bags, if x appears n times in S , then it appears $2n$ times in $S \cup S$.
- ❖ Thus $S \cup S \neq S$ in general.
 - e.g., $\{1\} \cup \{1\} = \{1,1\} \neq \{1\}$.

Bag Semantics

- ❖ Although the SELECT-FROM-WHERE statement uses **bag semantics**, the default for union, intersection, and difference is **set semantics**.
 - That is, duplicates are eliminated as the set operation is applied.

Motivation: Efficiency

- ❖ When doing projection, it is easier to avoid eliminating duplicates.
 - Just work tuple-at-a-time.
- ❖ For intersection or difference, it is most efficient to sort the relations first.
 - At that point you may as well eliminate the duplicates anyway.

Controlling Duplicate Elimination

- ❖ Force the result to be a set by
`SELECT DISTINCT . . .`
- ❖ Force the result to be a bag (i.e., don't eliminate duplicates) by `ALL`, as in
`. . . UNION ALL . . .`

第三章 关系数据库标准语言SQL

3.1 SQL概述

3.2 学生-课程数据库

3.3 数据定义

3.4 数据查询

3.5 数据更新

3.6 视图

3.5 数据更新

3.5.1 插入数据

3.5.2 删除数据

3.5.3 修改数据

3.5 数据更新(续)

❖ 3.5.1 插入数据

■ 格式

- 插入单个元组

INSERT INTO 表名 [(列名[, 列名]...]
VALUES (值 [, 值]...)

- 插入子查询结果

INSERT INTO 表名 [(列名[, 列名]...]
(子查询)

3.5 数据更新(续)

❖ 3.5.1 插入数据

- 示例

- **INSERT INTO** *Prof*
VALUES ('P0011', '王明' , 35, 08, 4980)
- **INSERT INTO** *Prof* (*P#*, *PNAME*, *D#*)
VALUES ('P0012', '李明' , 03)

3.5 数据更新(续)

❖ 3.5.1 插入数据

- Ex 10: 将平均成绩大于90的学生加入到表Excellent中

Excellent(s#,avg_score)

SC(s#, c#, score)

```
INSERT INTO Excellent ( s#, avg_score)  
    SELECT  s# , AVG(score)  
    FROM SC  
    GROUP BY s#  
    HAVING AVG(score) > 90
```

3.5 数据更新(续)

❖ 3.5.2 删除数据

- 格式

DELETE FROM <表名>

[WHERE <条件表达式>]

从表中删除符合条件的元组，如果没有where语句，
则删除所有元组

- Ex11: 删除所有选课记录。

DELETE FROM SC

- Ex12:删除学号为95019的学生记录。

DELETE

FROM Student

WHERE Sno = '95019'

3.5 数据更新(续)

❖ 3.5.3 修改数据

- 格式

UPDATE 表名

SET 列名 = 表达式 | 子查询

列名 = [, 表达式 | 子查询]...

[**WHERE** 条件表达式]

Ex 13: 将教师的工资上调5%

UPDATE Prof

SET sal = sal * 1.05

– 工资超过2000的缴纳10%所得税，其余的缴纳5%所得税

① *update* *PROF*

set $SAL = SAL * 0.9$

where $SAL > 2000$

② *update* *PROF*

set $SAL = SAL * 0.95$

where $SAL \leq 2000$

3.5 数据更新(续)

❖ 3.5.3 修改数据

Ex 14: 将01系系主任的工资改为该系的平均工资

DEPT(D# , DNAME , DEAN)

PROF(P# , PNAME, AGE, D# , SAL)

UPDATE Prof

SET sal = (**SELECT AVG**(sal)

FROM Prof

WHERE d# = '01')

WHERE pname = (**SELECT** dean

FROM Dept

WHERE d# = '01')

Ex15: 当C1课程的成绩小于该课程的平均成绩时，将该成绩提高5%

SC(S#, C#, score)

update SC

set score = score * 1.05

where C# = 'C1'

and score < (**select** avg(score)

from SC

where C# = 'C1')

第三章 关系数据库标准语言SQL

3.1 SQL概述

3.2 学生-课程数据库

3.3 数据定义

3.4 数据查询

3.5 数据更新

3.6 视图

3.6 视图

3.6.1 视图的基本概念

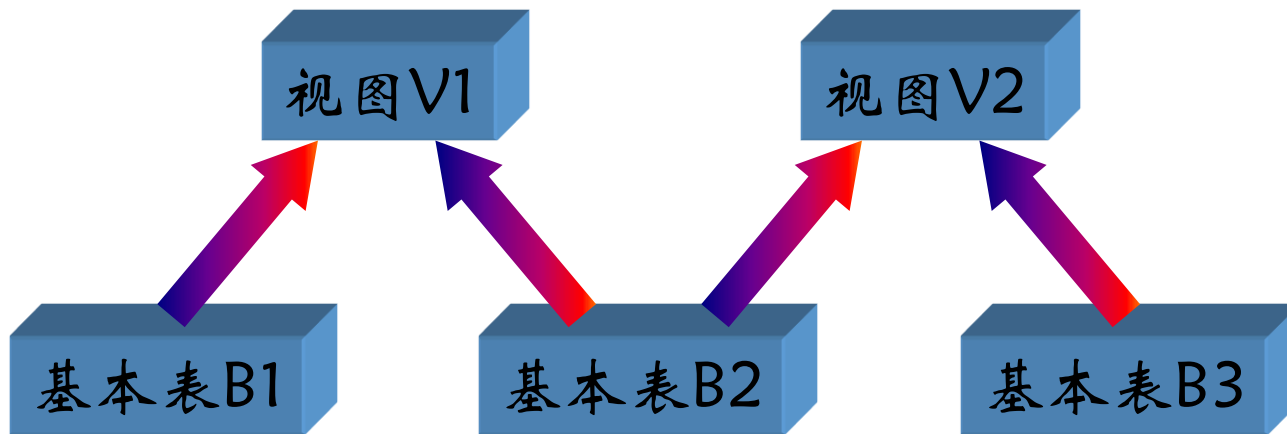
3.6.2 定义视图

3.6.3 查询视图

3.6.4 更新视图

3.6.5 视图的作用

3.6.1 视图的基本概念



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
- ❖ 视图一经定义就可以对其进行查询，但对视图的更新操作有一定的限制

3.6.1 视图的基本概念

基于视图的操作：

- ❖ 查询
- ❖ 删除
- ❖ 受限更新
- ❖ 定义基于该视图的新视图

3.6.2 定义视图

- 创建视图

CREATE [MATERIALIZED] VIEW <视图名> **AS** <查询表达式>

[WITH CHECK OPTION]

- **MATERIALIZED**是指物化视图，不仅生成视图定义，还将执行视图定义语句，将结果存储在物理磁盘上。
- 视图的属性名缺省为子查询结果中的属性名，也可以显式指明。
- **WITH CHECK OPTION**指明当对视图进行insert, update时，要检查是否满足视图定义中的条件。
- 查询表达式不允许含有**ORDER BY**子句和**DISTINCT**短语。

3.6.2 定义视图

Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

[例] 计算机系的花名册

Create View CS_Student As

Select sno, sname, ssex

From Student

Where sdept = 'CS'

[例] 列出计算机系的男生

Select sno, sname

From CS_Student

Where ssex = 'M'

3.6.2 定义视图

[例] 建立计算机系学生的视图，并要求进行修改和插入操作时仍需保证该视图只有计算机系的学生。

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

3.6.2 定义视图

对CS_Student视图的更新操作：

- ❖ 修改操作：自动加上Sdept= 'CS'的条件
- ❖ 删除操作：自动加上Sdept= 'CS'的条件
- ❖ 插入操作：自动检查Sdept属性值是否为 'CS'
 - 如果不是，则拒绝该插入操作
 - 如果没有提供Sdept属性值，则自动定义Sdept为 'CS'

3.6.2 定义视图

❖ 定义基于多个基表的视图

Student(sno, sname, ssex, sage, sdept)

Course(cno, cname, cpno, ccredit)

SC(sno, cno, grade)

- 建立一个含学号，姓名，课程名和成绩的视图

create view S_detail *as*

select sno, sname, cname, grade

from student, sc, course

where student.sno = sc.sno and

course.cno = sc.cno

3.6.2 定义视图

❖ 删除视图

DROP VIEW <视图名>;

- 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

3.6.3 查询视图

- ❖ 用户角度：查询视图与查询基本表相同
- ❖ RDBMS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

3.6.3 查询视图

[例] 在计算机系学生的视图中找出年龄小于20岁的学生。

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

视图消解转换后的查询语句为：

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

3.6.3 查询视图

[例] 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;
```

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;
```

3.6.3 查询视图

转换为:

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

3.6.3 查询视图

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;
```

3.6.4 更新视图

❖ 相关说明

- 对视图的删除操作
- 对视图的更新操作
 - INSERT, UPDATE, DELETE
 - 操作约束
- 视图的优点
 - 个性化服务
 - 安全性
 - 逻辑独立性

3.6.4 更新视图

❖ 操作分类

- INSERT, DELETE, UPDATE

❖ 特点

- 对视图的更新，最终要转换为对基本表的更新

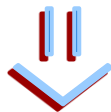
❖ View Ex 1:

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

3.6.4 更新视图

- ❖ Ex 1: 向信息系学生视图IS_Student中插入一个新的学生记录，其中学号为95029，姓名为赵新，年龄为20岁

```
INSERT  
INTO IS_Student  
VALUES('95029', '赵新', 20)
```



```
INSERT  
INTO Student(sno,sname,sage,sdept)  
VALUES('95029', '赵新', 20, 'IS')
```

3.6.4 更新视图

- ❖ Ex 2:删除计算机系学生视图CS_Student中学号为95029的记录

```
DELETE  
FROM CS_Student  
WHERE sno='95029'
```



```
DELETE  
FROM Student  
WHERE Sno='95029' AND Sdept='CS'
```

3.6.4 更新视图

❖ 视图更新的限制

- **SELECT**子句中的目标列不能包含聚集函数
- **SELECT**子句中不能使用**UNIQUE**或**DISTINCT**关键字
- 不能包括**GROUP BY**子句
- 不能包括经算术表达式计算出来的列
- 对于**行列子集视图**可以更新（视图是从单个基本表使用选择、投影操作导出的，并且**包含了基本表的主码**）
- 对其他类型视图的更新不同系统有不同限制

视图更新的限制

❖ 示例:

```
CREATE VIEW P_SAL  
AS SELECT   pno, pname , sal  
FROM       Prof
```

```
INSERT INTO P_SAL  
VALUES ( 'P0018' , '金海心' , 7500 )
```

转换
为

```
INSERT INTO PROF  
VALUES ( 'P0018' , '金海心' , null , null ,  
7500 )
```

视图更新的限制

PROF(P# , PNAME, AGE, D# , SAL)

CREATE VIEW prof_1

AS

SELECT pname, sal

FROM prof

insert into prof_1
values('王涛',3400)

不能插入，因为没有主码P#

3.6.5 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

下课了。。。



休息一会儿。。。

