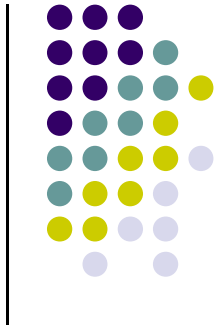
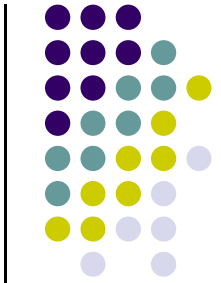


第 三 章

关系数据库标准语言 SQL

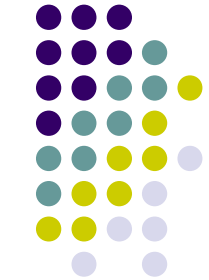
https://dev.mysql.com/doc/refman/8.0/en/sql-statements.html

- SQL语言 (Structured Query Language)
  - 1974年由Boyce和Chamberlin提出
  - 1975年～1979年IBM公司在System R原型系统上实现
  - 是关系数据库的标准语言，是数据库领域中一个主流语言
  - SQL是一个通用的、功能极强的关系数据库语言



- SQL标准发展历程
  - SQL-86
    - 第一个SQL标准
    - 由美国国家标准局（American National Standard Institute，简称ANSI）公布
    - 1987 年 国 际 标 准 化 组 织 （ International Organization for Standardization，简称ISO）通过
  - SQL-89:增加了引用完整性
  - SQL-92:被DBMS生产商广发泛接受
  - SQL-99（SQL3）:Core level跟其他8种相应的level，包括递归查询，程序跟流程控制，基本的对象支持包括oids；
  - SQL-2003:包含了XML相关内容,自动生成列值
  - SQL-2006:定义了SQL与XML(包含XQuery)的关联应用
  - SQL-2008：
  - SQL-2011:

第三章 关系数据库标准语言SQL



3.1 SQL概述

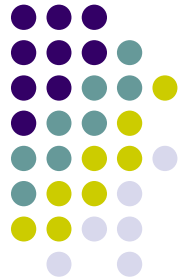
3.2 数据定义

3.3 数据查询

3.4 数据更新

3.5 视图

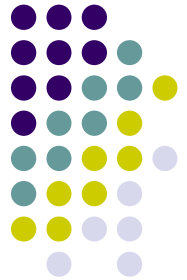
3.1 SQL 概 述



3.1.1 SQL的特点

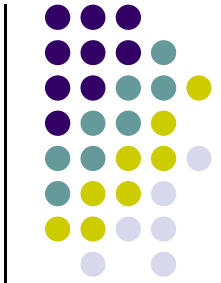
3.1.2 SQL的基本概念

3.1.1 SQL的特点



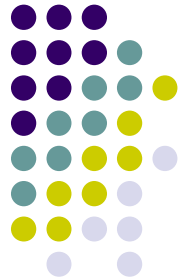
1. 综合统一
2. 高度非过程化
3. 面向集合的操作方式
4. 同一种语法结构提供两种使用方式
5. 语言简洁，易学易用

1 综合统一



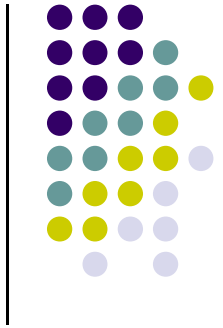
- SQL语言集数据定义语言DDL、数据操纵语言DML、数据控制语言DCL的功能于一体

- 非关系模型的数据语言分为
  - 模式数据定义语言（模式DDL）
  - 外模式数据定义语言（外模式DDL或子模式DDL）
  - 与数据存储有关的描述语言（DSDL）
  - 数据操纵语言（DML）



- 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，不影响数据库的运行。
- 数据结构的统一带来了数据操作符统一

2. 高度非过程化



- 用户只需提出“做什么”，而不必指明“怎么做”
- 存取路径的选择以及SQL语句的操作过程由系统自动完成。大大减轻了用户负担，而且有利于提高数据独立性。

3. 面向集合的操作方式

- SQL语言采用集合操作方式
  - 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

- 非关系数据模型采用的是面向记录的操作方式，操作对象是一条记录。

4. 同一种语法结构提供两种使用方式

- 独立的语言
  - 能够独立地用于联机交互的使用方式

- 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。

两种不同使用方式下，SQL语言的语法结构基本一致

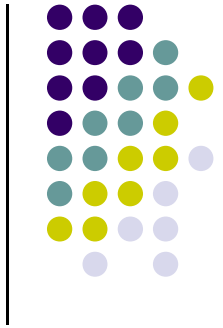
5. 语言简洁，易学易用

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

3.1 SQL 概 述

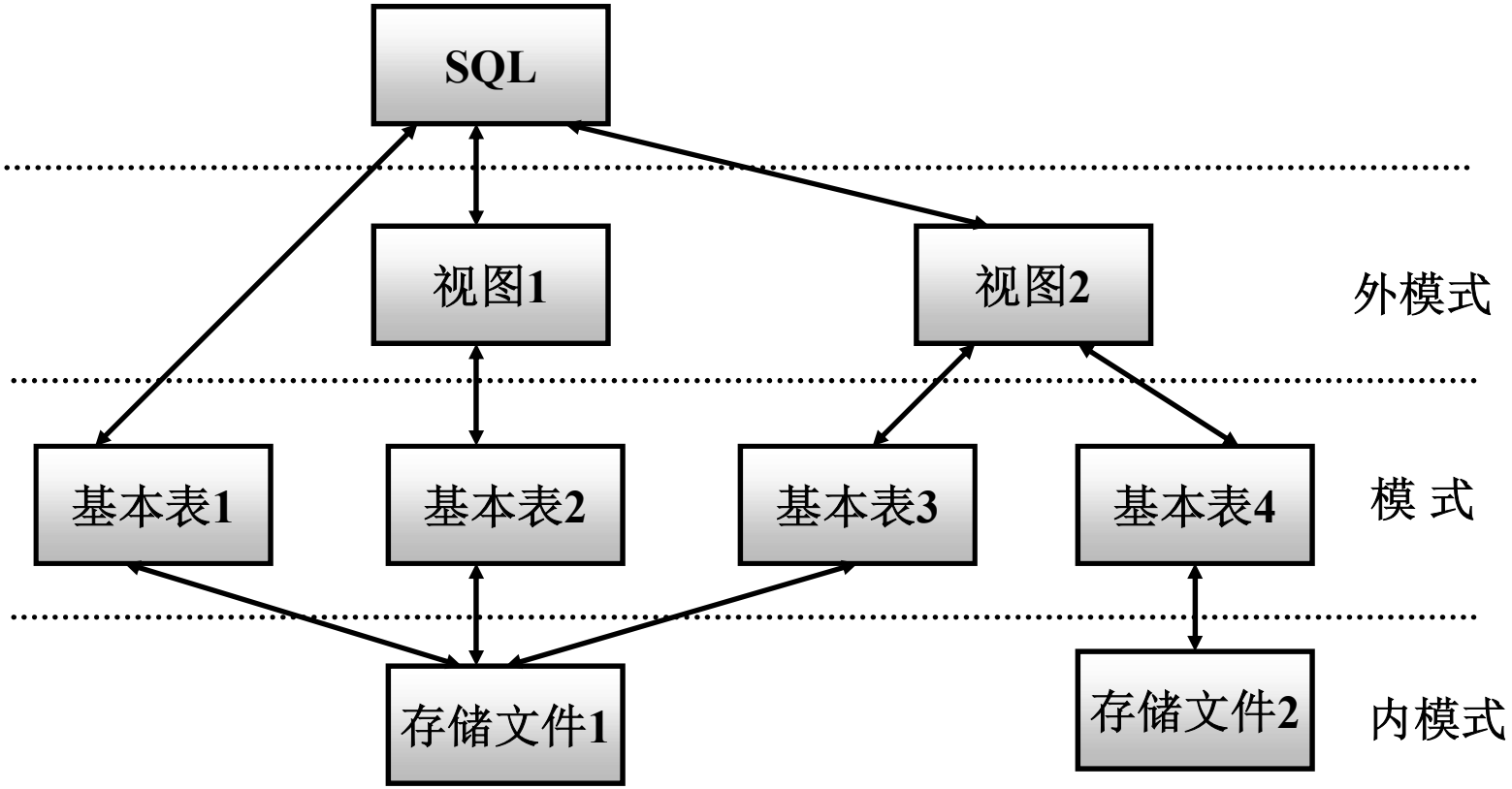


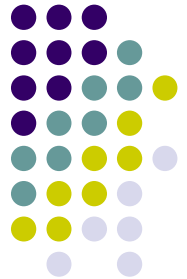
3.1.1 SQL的特点

3.1.2 SQL的基本概念

3.1.2 SQL的基本概念

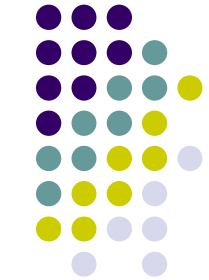
SQL支持关系数据库三级模式结构





- 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 数据定义

3.3 数据查询

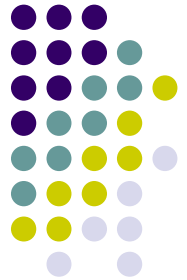
3.4 数据更新

3.5 视图

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**	**ALTER INDEX**

3.2 数据定义

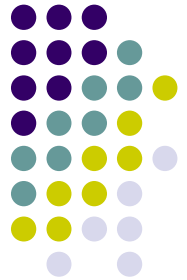


3.2.1 定义与删除模式

3.2.2 定义、删除与修改基本表

3.2.3 建立与删除索引

定义模式



- 语句格式：

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

  - 如果没有指定<模式名>， 那么<模式名> 隐含为<用户名>；
  - 创建模式，必须拥有DBA权限或者是CREATE SCHEMA的权限。

[例1]定义一个学生-课程模式S-T

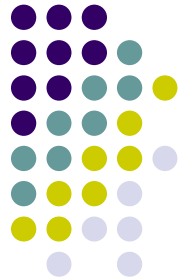
CREATE SCHEMA “S-T” AUTHORIZATION WANG;

为用户WANG定义了一个模式S-T

[例2]CREATE SCHEMA AUTHORIZATION WANG；

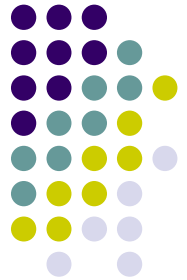
<模式名>隐含为用户名WANG

- 如果没有指定<模式名>，那么<模式名>隐含为<用户名>



- 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。

二、删除模式



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

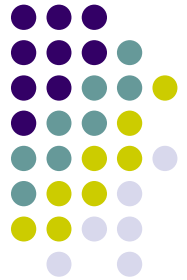
CASCADE(级联)

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。

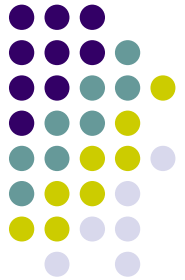


[例4] DROP SCHEMA WANG CASCADE;

删除模式WANG

同时该模式中定义的对象也被删除

3.2 数据定义



3.2.1 定义与删除模式

3.2.2 定义、删除与修改基本表

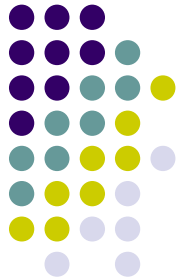
一、定义基本表

二、修改基本表

三、删除基本表

3.2.3 建立与删除索引

一、定义基本表



- 关系名（表名）
- 属性名（列名）
- 完整性约束

- 语句格式

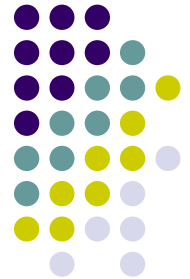
CREATE TABLE <表名>

(<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



- 表级完整性约束与列级完整性约束
  - *列级完整性约束条件*指非空、唯一、主码、缺省值等。

    [***default*** 缺省值] [***not null***] [UNIQUE]
  - *表级完整性约束条件*包括主码、外码、用户自定义商业规则或完整性约束，可以用以下语法来描述：

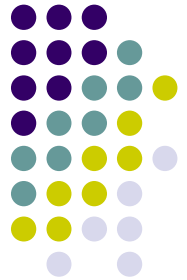
    [，***primary key***（列名 [，列名] ...）]

    [，***foreign key***（列名 [，列名] ...）

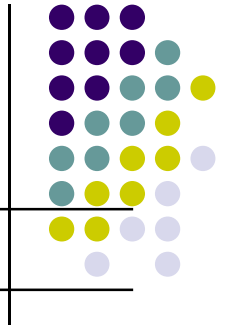
    references 表名（列名 [，列名] ...）]

    [，***check***（条件）]
- 常用完整性约束
  - 主码约束： PRIMARY KEY；参照完整性约束；唯一性约束：UNIQUE；非空值约束：NOT NULL；取值约束：CHECK

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

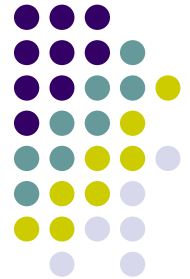


数据类型	含义
CHAR(n)	长度为n的定长字符串
VARCHAR(n)	最大长度为n的变长字符串
INT	长整数（也可以写作INTEGER）
SMALLINT	短整数
NUMERIC(p，d)	定点数，由p位数字（不包括符号、小数点）组成，小数后面有d位数字
REAL	取决于机器精度的单精度浮点数
Double Precision	取决于机器精度的双精度浮点数
FLOAT(n)	浮点数，精度至少为n位数字
DATE	日期，包含年、月、日，格式为YYYY-MM-DD
TIME	时间，包含一日的时、分、秒，格式为HH:MM:SS

SQL 提供的一些主要数据类型

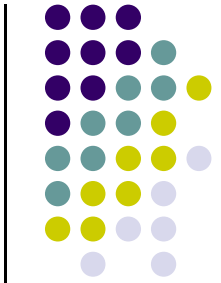
- 不同的数据库系统支持的数据类型不完全相同
- IBM DB2 SQL支持的数据类型
  - SMALLINT　　半字长二进制整数。
  - INTEGER或INT　　全字长二进制整数。
  - DECIMAL(p[, q])　　压缩十进制数，共p位，其中小数点后有q位。0≤q≤p≤15，q=0时可以省略不写。

    或 DEC(p[, q])
  - FLOAT　　双字长浮点数。



IBM DB2 SQL支持的数据类型

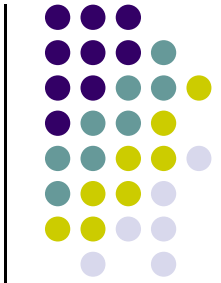
- CHARTER(n)　　长度为n的定长字符串
- 或CHAR(n)
- VARCHAR(n)　最大长度为n的变长字符串
- GRAPHIC(n)　长度为n的定长图形字符串
- VARGRAPHIC(n)　最大长度为n的变长图形字符串
- DATE　　日期型，格式为YYYY-MM-DD
- TIME　　时间型，格式为HH.MM.SS
- TIMESTAMP　　日期加时间



ORACLE SQL支持的数据类型

- VARCHAR2(n)
- CHAR(n)
- NUMBER[(p[,q])]
- DATE
- LONG
- RAW

  或 LONGRAW

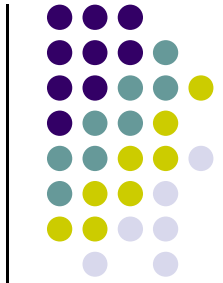


Microsoft SQL Server支持的数据类型

- TINYINT
- SMALLINT
- INTEGER or INT
- REAL
- FLOAT
- CHARACTER(n) or CHAR(n)
- VARCHAR(n)
- DATETIME
- TIMESTAMP

[例1] 建立一个“学生”表Student，它由学号Sno、姓名Sname、性别Ssex、年龄Sage、所在系Sdept五个属性组成。其中学号不能为空，值是唯一的，并且姓名取值也唯一。

Sno	Sname	Ssex	Sage	Sdept
↑ 字符型 长度为5 不能为空值	↑ 字符型 长度为20	↑ 字符型 长度为1	↑ 整数	↑ 字符型 长度为15

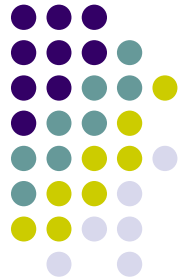


```
CREATE TABLE Student
        (Sno  CHAR(5) NOT NULL UNIQUE，
         Sname  CHAR(20) UNIQUE，
         Ssex  CHAR(1)，
         Sage  INT，
         Sdept  CHAR(15))；
```

[例2] 建立一个“学生选课”表SC，它由学号Sno、课程号Cno，修课成绩Grade组成，其中(Sno, Cno)为主码。

```
CREATE TABLE SC(
        Sno CHAR(5) ,
        Cno CHAR(3) ,
        Grade   int,
        Primary key (Sno, Cno));
```

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”.Cource（.....）；

    Create table “S-T”.SC（.....）；

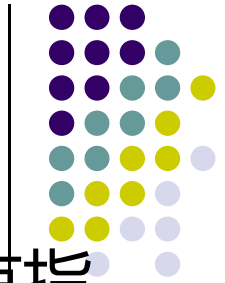
- 方法二：在创建模式语句中同时创建表
  - 在CREATE SCHEMA中可以接受CREATE TABLE, CREATE VIEW和GRANT子句。

    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>]
  - 例子见下页
- 方法三：设置所属的模式

[例]

```
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。



- 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

- DBA用户可以设置搜索路径，然后定义基本表

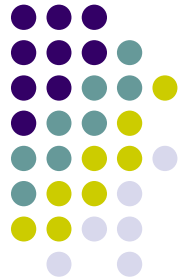
  SET search_path TO “S-T”，PUBLIC；

  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。

二、修改基本表



ALTER TABLE <表名>

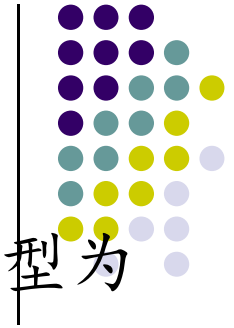
[ADD [COLUMN]<新列名> <数据类型> [完整性约束]]

[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>];



[例2] 向Student表增加“入学时间”列，其数据类型为日期型。

ALTER TABLE Student ADD Scome DATE；

- 不论基本表中原来是否已有数据，新增加的列一律为空值。
- 如果基本表中原来已有数据，新增列不可有NOT NULL约束

[例3] 将年龄的数据类型由字符型改为整型。

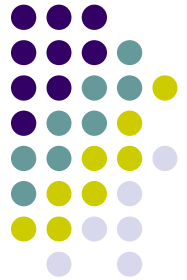
```
ALTER TABLE Student ALTER COLUMN Sage INT;
```

- 注：修改原有的列定义有可能会破坏已有数据。

[例4] 增加课程名必须取唯一值的约束。

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

三、删除基本表



DROP TABLE <表名> [RESTRICT| CASCADE] ；

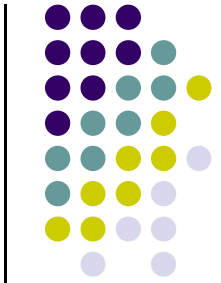
RESTRICT：删除表是有限制的。

- 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；

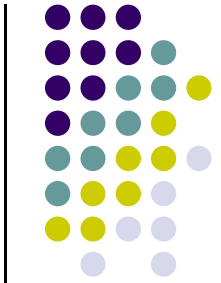
缺省情况是RESTRICT;



[例5] 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



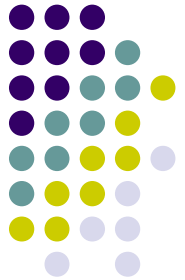
不同数据库产品在遵循SQL标准的基础上具体实现细节和处理策略上会与标准有差别；

- P87页表DROP Table时，SQL2011与3个RDBMS的处理策略比较。

SQL Server 没有RESTRICT 和CASCADE选项。

MySQL8.0中呢?

3.2 数据定义

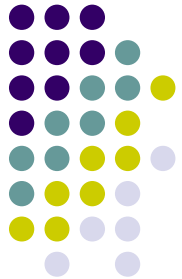


3.2.1 定义与删除模式

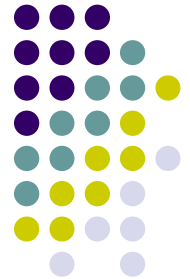
3.2.2 定义、删除与修改基本表

3.2.3 建立与删除索引

3.2.3 建立与删除索引



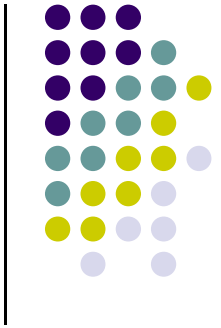
- 建立索引的目的：加快查询速度
- 谁可以建立索引
  - DBMS一般会自动建立以下列上的索引
    - PRIMARY KEY
    - UNIQUE
  - DBA或表的属主（即建立表的人）
- 谁 维护索引
  - DBMS自动完成
- 使用索引
  - DBMS自动选择是否使用索引以及使用哪些索引



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

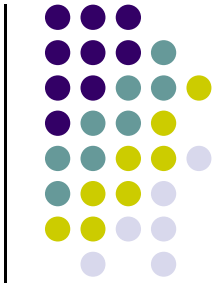
一、建立索引



1、在已有表上创建：

CREATE [UNIQUE] [CLUSTER] INDEX <索引名>
ON <表名>(<列名>[<次序>][,<列名>[<次序>]]…);

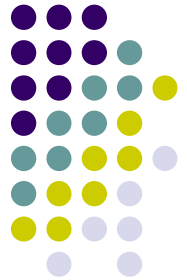
- 用<表名>指定要建索引的基本表名字
- 索引可以建立在该表的一列或多列上，各列名之间用逗号分隔
- 用<次序>指定索引值的排列次序，升序：ASC，降序：DESC。缺省值：ASC
- UNIQUE表明此索引的每一个索引值只对应唯一的数据记录
- CLUSTER表示要建立的索引是聚簇索引(把某个属性或者属性组上具有相同值的元组集中存放在连续的物理块中称为聚簇)



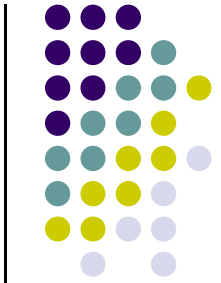
- 2、在已有表上创建：

通过 **Alter table 表名 add** [unique|fulltext(MySQL中)] **index** 索引名 (属性名[(长度)] [asc|desc])

- 3、在**create table**时创建，在表级完整性约束位置添加索引：[unique|fulltext] **index** [索引名](属性名[(长度)][asc|desc])



- 唯一值索引
  - 对于已含重复值的属性列不能建UNIQUE索引
  - 对某个列建立UNIQUE索引后，插入新记录时DBMS会自动检查新记录在该列上是否取了重复值。这相当于增加了一个UNIQUE约束。



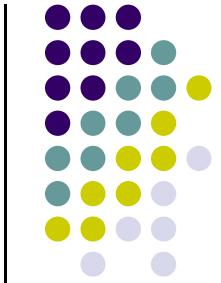
- 聚簇索引
  - 建立聚簇索引后，基表中数据也需要按指定的聚簇属性值的升序或降序存放。也即聚簇索引的索引项顺序与表中记录的物理顺序一致。

例：

CREATE CLUSTERED INDEX Stusname ON Student(Sname);

在Student表的Sname（姓名）列上建立一个聚簇索引，而且Student表中的记录将按照Sname值的升序存放

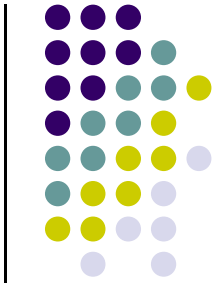
- 在一个基本表上最多只能建立一个聚簇索引
- 聚簇索引的用途：对于某些类型的查询，可以提高查询效率
- 聚簇索引的适用范围
  - 很少对基表进行增删操作
  - 很少对其中的变长列进行修改操作



[例6] 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、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7] 删除Student表的Stusname索引。

  DROP INDEX Stusname;

第三章 关系数据库标准语言SQL

3.1 SQL概述

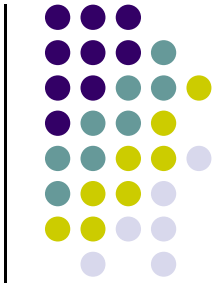
3.2 数据定义

3.3 数据查询

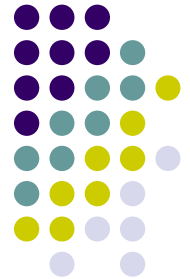
- 3.3.1 单表查询
- 3.3.2 连接查询
- 3.3.3 嵌套查询
- 3.3.4 集合查询
- 3.3.5 小结

3.4 数据更新

3.5 视图

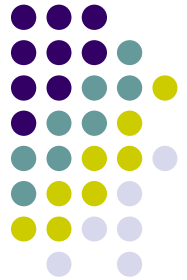


```
SELECT [ALL|DISTINCT]
                <目标列表达式> [<别名>]
                [，<目标列表达式>[<别名>]] …
FROM    <表名或视图名>[<别名>]
                [，  <表名或视图名>[<别名>] ] …
[ WHERE <条件表达式> ]
[ GROUP BY <列名> [, <列名>] …
                                [ HAVING <条件表达式> ] ]
[ ORDER BY <列名> [, <列名>] … [ ASC|DESC ] ];
```



- SELECT子句：指定要显示的属性列
- FROM子句：指定查询对象(基本表或视图)
- WHERE子句：指定查询条件
- GROUP BY子句：对查询结果按指定列的值分组，该属性列值相等的元组为一个组。通常会在每组中作用集函数。
- HAVING短语：筛选出满足指定条件的组
- ORDER BY子句：对查询结果表按指定列值的升序或降序排序

示例数据库



学生-课程数据库

- 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 号 Sno	姓 名 Sname	性 别 Ssex	年 龄 Sage	所在系 Sdept
2014001	李勇	男	20	CS
2014002	刘晨	女	19	IS
2014003	王敏	女	18	MA
2014004	张立	男	19	IS

Student

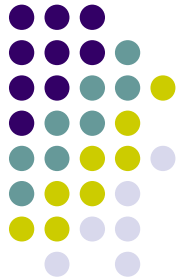
课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL语言	6	4

Course

学 号 Sno	课 程 号 Cno	成 绩 Grade
2014001	1	92
2014001	2	85
2014001	3	88
2014002	2	90
2014002	3	80

SC

3.3 查　询



3.3.1 单表查询

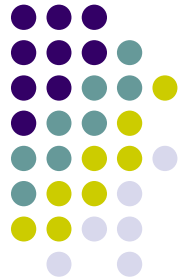
3.3.2 连接查询

3.3.3 嵌套查询

3.3.4 集合查询

3.3.5 小结

3.3.1 单表查询

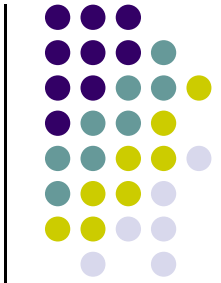


- 单表查询

  查询仅涉及一个表，是一种最简单的查询操作

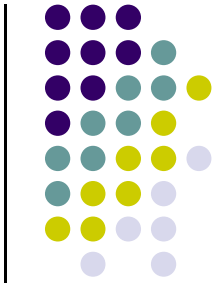
  - 选择表中的若干列
  - 选择表中的若干元组
  - 对查询结果排序
  - 使用集函数
  - 对查询结果分组

一、选择表中的若干列

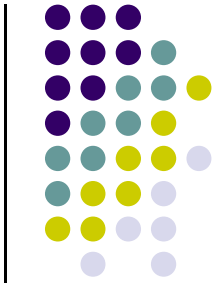


- 属投影运算
  - 不消除重复行
- 变化方式主要表现在SELECT子句的<目标表达式>上
  - 查询指定列
  - 查询全部列
  - 查询经过计算的值

1. 查询指定列



- 方法
  - 在SELECT子句的<目标列表达式>中指定要查询的属性
  - <目标列表达式>中各个列的先后顺序可以与表中的逻辑顺序不一致。即用户可以根据应用的需要改变列的显示顺序



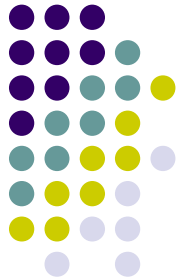
[例1] 查询全体学生的学号与姓名。

```
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

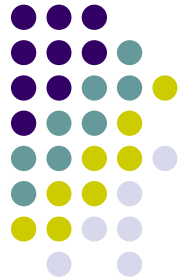


- 方法
  - 在SELECT关键字后面列出所有列名
  - 当列的显示顺序与其在基表中的顺序相同时，也可以简单地将<目标列表达式>指定为 *

[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



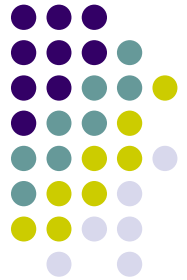
- 方法
  - SELECT子句的<目标列表达式>为表达式
    - 算术表达式
    - 字符串常量
    - 函数
    - 列别名

[例4] 查全体学生的姓名及其出生年份。

```
SELECT Sname，2019-Sage
FROM Student；
```

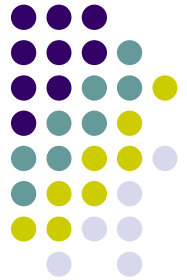
输出结果：

Sname	2019-Sage
李勇	1999
刘晨	2000
王敏	2001
张立	2000



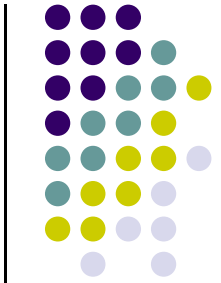
[例5] 查询全体学生的姓名、出生年份和所有系，要求用小写字母表示所有系名。

```
SELECT Sname, 'Year of Birth: ', 2019-Sage,
ISLOWER(Sdept)  FROM Student;
```



输出结果：

Sname	'Year of Birth:'	2019-Sage	ISLOWER(Sdept)
李勇	Year of Birth:	1999	cs
刘晨	Year of Birth:	2000	is
王敏	Year of Birth:	2001	ma
张立	Year of Birth:	2000	is



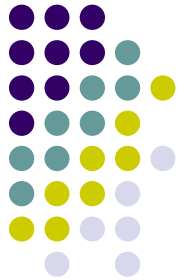
[例5.1] 使用列别名改变查询结果的列标题

```
SELECT Sname NAME，'Year of Birth: ' BIRTH,
              2019-Sage BIRTHDAY,
              LOWER(Sdept) DEPARTMENT
FROM Student;
```

输出结果：

NAME	BIRTH	BIRTHDAY	DEPARTMENT
李勇	Year of Birth:	1999	cs
刘晨	Year of Birth:	2000	is
王名	Year of Birth:	2001	ma
张立	Year of Birth:	2000	is

二、选择表中的若干元组



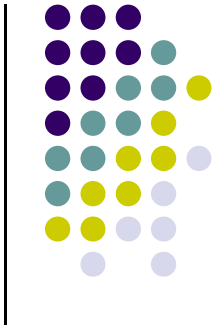
- 消除取值重复的行
- 查询满足条件的元组

1. 消除取值重复的行

- 方法
  - 在SELECT子句中使用DISTINCT短语

示例：假设SC表中有下列数据

Sno	Cno	Grade
2014001	1	92
2014001	2	85
2014001	3	88
2014002	2	90
2014002	3	80



[例6] 查询选修了课程的学生学号。

(1) SELECT Sno
FROM SC;

或

SELECT ALL Sno
FROM SC;

结果：

```
Sno
---------
2014001
2014001
2014001
2014002
2014002
```

(2) SELECT DISTINCT Sno
FROM SC;

结果：

```
Sno
----------
2014001
2014002
```

- 注意

DISTINCT短语的作用范围是所有目标列

例：查询选修课程的各种成绩

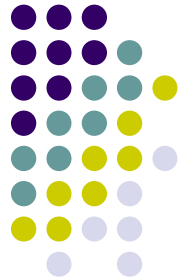
错误的写法

```
SELECT DISTINCT Cno，DISTINCT Grade
FROM SC;
```

正确的写法

```
SELECT DISTINCT Cno，Grade
FROM SC;
```

2. 查询满足条件的元组



- 属选择运算

- 通过WHERE子句实现
  - 比较大小
  - 确定范围
  - 确定集合
  - 字符串匹配
  - 涉及空值的查询
  - 多重条件查询

表3.4 常用的查询条件

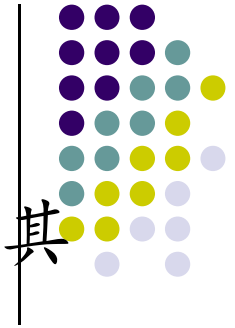
查 询 条 件	谓　词
比　较	=，>，<，>=，<=，!=，<>，!>，!<；NOT+上述比较运算符
确定范围	BETWEEN AND，NOT BETWEEN AND
确定集合	IN，NOT IN
字符匹配	LIKE，NOT LIKE
空　值	IS NULL，IS NOT NULL
多重条件（逻辑运算）	AND，OR，NOT

(1) 比较大小

- 方法
  - 在WHERE子句的<比较条件>中使用比较运算符
    - =，>，<，>=，<=，!=或<>，!>，!<，
    - 逻辑运算符NOT ＋ 含上述比较运算符的表达式

[例7] 查询计算机系全体学生的名单。

```
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；
```

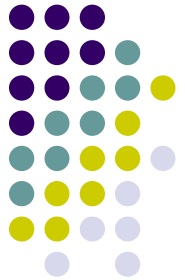
或

```
SELECT Sname，Sage
FROM Student
WHERE NOT 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



- 方法
  - 使用谓词　BETWEEN … AND …

    NOT BETWEEN … AND …
    - BETWEEN后：范围的下限（即低值）
    - AND后：范围的上限（即高值）
  - 用多重条件查询实现

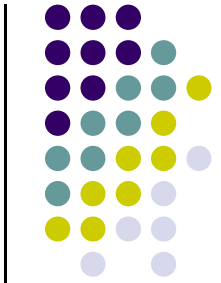
[例10] 查询年龄在20~23岁（包括20岁和23岁）之间的学生的姓名、系别和年龄。

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

[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

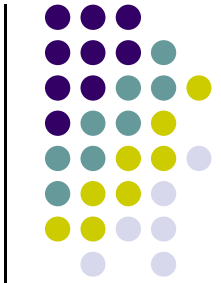


- 方法
  - 使用谓词　　IN <值表>

    NOT IN <值表>
    - <值表>：用逗号分隔的一组取值
  - 用多重条件查询实现

[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' );
```



[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。

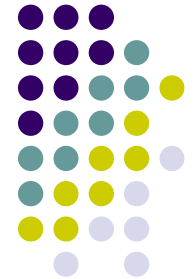
```
SELECT Sname，Ssex
FROM Student
 WHERE Sdept NOT IN ( 'IS'，'MA'，'CS' );
```

(4) 字符串匹配

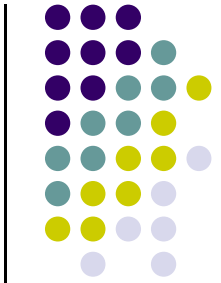
- 使用谓词LIKE或NOT LIKE

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

- <匹配串>：指定匹配模板
  - 匹配模板：固定字符串或含通配符的字符串



- 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。



例题：匹配模板为固定字符串

[例14] 查询学号为2014001的学生的详细情况。

```
SELECT *
FROM  Student
WHERE  Sno LIKE '2014001';
```

等价于：

```
SELECT  *
FROM  Student
WHERE Sno = '2014001';
```

- 当匹配模板为固定字符串时，可以用=运算符取代LIKE谓词，用 != 或 < >运算符取代NOT LIKE谓词

例题： 匹配模板为含通配符的字符串

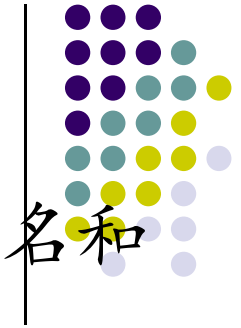
[例15] 查询所有姓刘学生的姓名、学号和性别。

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

[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_ _’;
```

注意：一个汉字要占两个字符的位置，所以匹配串欧阳后面需要跟两个_ 。

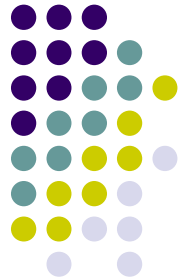


[例17] 查询名字中第2个字为"阳"字的学生的姓名和学号。

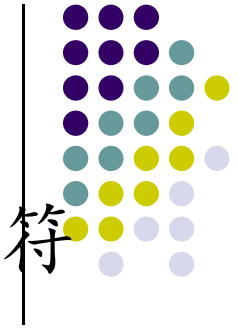
```
SELECT Sname，Sno
FROM Student
WHERE Sname LIKE ‘_ _阳%’;
```

[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%';
```



- ESCAPE 短语：
  - 当用户要查询的字符串本身就含有 % 或 _ 时，要使用ESCAPE '<换码字符>' 短语对通配符进行转义。

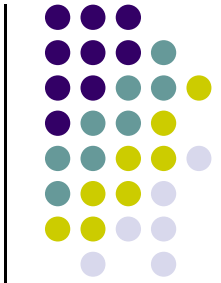


例题：使用换码字符将通配符转义为普通字符

[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'
```

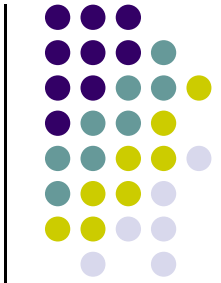
```
SELECT Cno，Ccredit
FROM Course
WHERE Cname LIKE 'DB\_Design'
                ESCAPE '\'
```



[例20] 查询以"DB_"开头，且倒数第3个字符为 i的课程的详细情况。

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

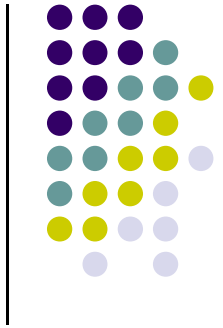
(5) 涉及空值的查询



- 方法
  - 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的学号和相应的课程号。

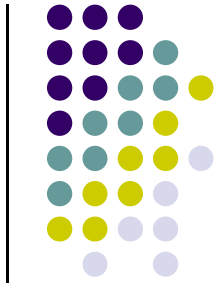
```
SELECT Sno，Cno
FROM SC
WHERE Grade IS NULL；
```



[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；
```

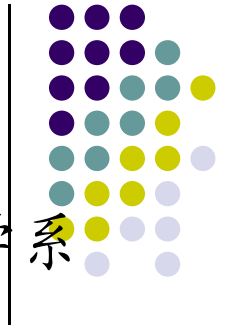
(6) 多重条件查询



- 方法
  - 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 …

[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;
```

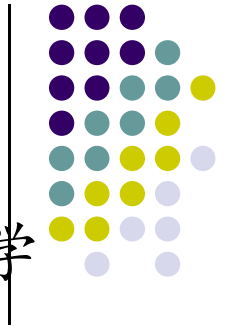


[例12] 查询信息系（IS）、数学系（MA）和计算机科学系（CS）学生的姓名和性别。

```
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 ';
```



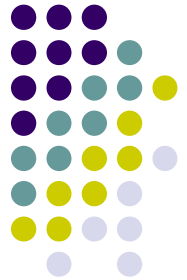
[例10] 查询年龄在20~23岁（包括20岁和23岁）之间的学生的姓名、系别和年龄。

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

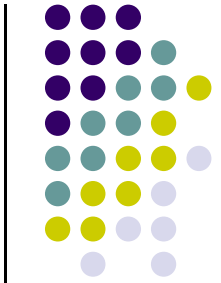
可改写为：

```
    SELECT Sname，Sdept，Sage
   FROM Student
    WHERE  Sage>=20 AND Sage<=23；
```

三、对查询结果排序



- 方法
  - 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（具体系统决定）
    - 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



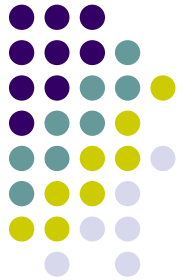
[例24] 查询选修了3号课程的学生的学号及其成绩，查询结果按分数降序排列。

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

[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、使用集函数



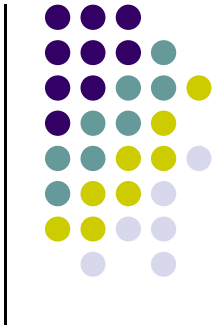
- 方法
  - 5类主要集函数
    - 计数

      COUNT（*）

      COUNT（[DISTINCT|ALL] <列名>）
    - 计算总和

      SUM（[DISTINCT|ALL] <列名>）
    - 计算平均值

      AVG（[DISTINCT|ALL] <列名>）



- 5类主要集函数
  - 求最大值

    MAX（[DISTINCT|ALL] <列名>）
  - 求最小值

    MIN（[DISTINCT|ALL] <列名>）
- DISTINCT短语：在计算时要取消指定列中的重复值
- ALL短语：不取消重复值
- ALL为缺省值

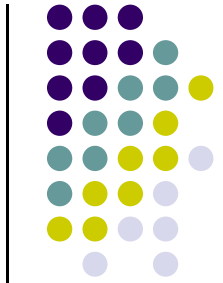
[例26] 查询学生总人数。

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

[例27] 查询选修了课程的学生人数。

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

注：用DISTINCT以避免重复计算学生人数



[例28] 计算1号课程的学生平均成绩。

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

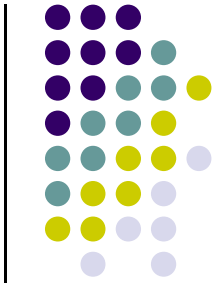
[例29] 查询选修1号课程的学生最高分数。

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

五、对查询结果分组

- 用途
  - 细化集函数的作用对象
    - 未对查询结果分组，集函数将作用于整个查询结果
    - 对查询结果分组后，集函数将分别作用于每个组
- 方法（参照后面例题）
  - 使用GROUP BY子句分组
    - 分组方法：按指定的一列或多列值分组，值相等的为一组
    - 使用GROUP BY子句后，SELECT子句的列名列表中只能出现分组属性和集函数
    - GROUP BY子句的作用对象是查询的中间结果表

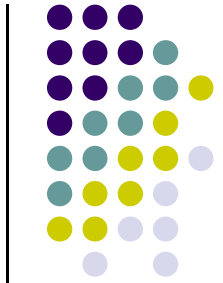
插 播



- MySQL中的group by 语句可以select 没有被分组的字段，如：

select id, name, age from A group by age是允许的，取出的id,name所在的行是每个分组中的第一行数据。

- GROUP_CONCAT()：把来自同一个组的某一列（或者多列）的数据连接起来成为一个字符串。

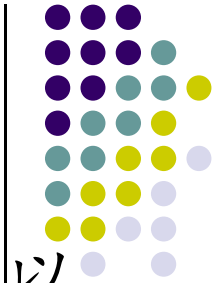


[例30] 求各个课程号及相应的选课人数。

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

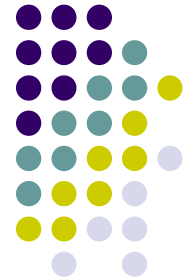
结果形式为：

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

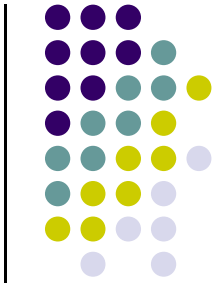


[例31] 求各个课程号及相应的课程成绩在90分以上的学生人数。

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



- 方法
  - 使用HAVING短语筛选最终输出结果
    - 只有满足HAVING短语指定条件的组才输出
    - HAVING短语与WHERE子句的区别：作用对象不同
      - WHERE子句作用于基表或视图，从中选择满足条件的元组。
      - HAVING短语作用于组，从中选择满足条件的组。



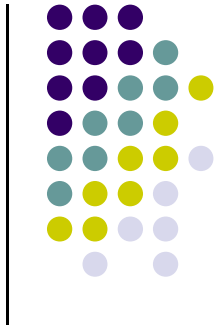
[例32] 查询选修了3门以上课程的学生学号。

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

[例33] 查询有3门以上课程在90分以上的学生的学号及90分以上的课程数。

```
SELECT Sno, COUNT(*)
FROM SC
WHERE Grade>=90
GROUP BY Sno
HAVING COUNT(*)>=3;
```

3.3 查　询



3.3.1 单表查询

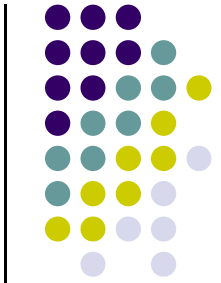
3.3.2 连接查询

3.3.3 嵌套查询

3.3.4 集合查询

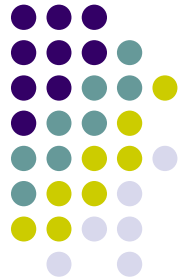
3.3.5 小结

3.3.2 连接查询



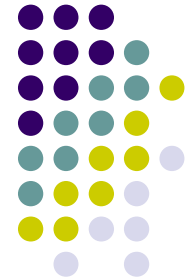
- 同时涉及多个表的查询称为连接查询
- 连接条件
  - 用来连接两个表的条件称为连接条件或连接谓词
  - 常用格式
    - [<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>

      主要比较运算符：=、>、<、>=、<=、!=
    - [<表名1>.]<列名1> BETWEEN [<表名2>.]<列名2> AND [<表名2>.]<列名3>



- 连接字段
  - 连接谓词中的列名称为连接字段
  - 连接条件中的各连接字段类型必须是可比的，但不必是相同的

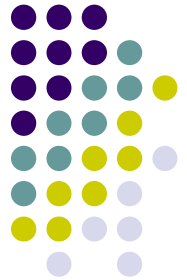
- 连接操作的执行过程
  - 嵌套循环法(NESTED-LOOP)
    - 首先在表1中找到第一个元组，然后从头开始扫描表2，逐一查找满足连接件的元组，找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组。
    - 表2全部查找完后，再找表1中第二个元组，然后再从头开始扫描表2，逐一查找满足连接条件的元组，找到后就将表1中的第二个元组与该元组拼接起来，形成结果表中一个元组。
    - 重复上述操作，直到表1中的全部元组都处理完毕为止。



- 排序合并法(SORT-MERGE)：常用于=连接
  - 首先按连接属性对表1和表2排序
  - 对表1的第一个元组，从头开始扫描表2，顺序查找满足连接条件的元组，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。当遇到表2中第一条大于表1连接字段值的元组时，对表2的查询不再继续
  - 找到表1的第二条元组，然后从刚才的中断点处继续顺序扫描表2，查找满足连接条件的元组，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。直接遇到表2中大于表1连接字段值的元组时，对表2的查询不再继续
  - 重复上述操作，直到表1或表2中的全部元组都处理完毕为止

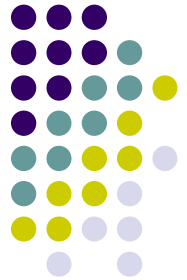
- 索引连接(INDEX-JOIN)
  - 对表2按连接字段建立索引
  - 对表1中的每个元组，依次根据其连接字段值查询表2的索引，从中找到满足条件的元组，找到后就将表1中的第一个元组与该元组拼接起来，形成结果表中一个元组

SQL中连接查询的主要类型



- 广义笛卡尔积
- 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（自学）
- 复合条件连接查询

一、广义笛卡尔积

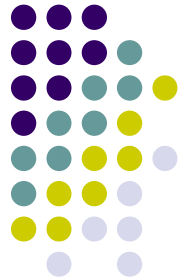


- 不带连接谓词的连接
- 很少使用

例:

```
SELECT Student.*, SC.*
FROM Student, SC
```

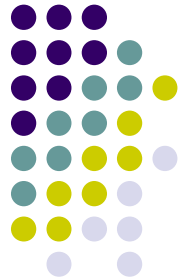
二、等值与非等值连接查询



- 等值连接
- 自然连接
- 非等值连接

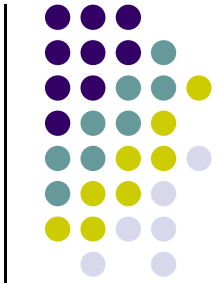
二、等值与非等值连接查询

- 等值连接
  - 连接运算符为 = 的连接操作
    - [<表名1>.]<列名1> = [<表名2>.]<列名2>
  - 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。



[例32] 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；
```



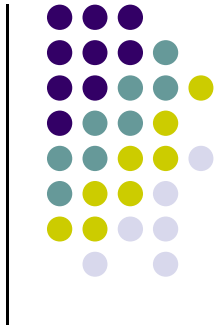
假设Student表、SC表分别有下列数据：

Student表

Sno	Sname	Ssex	Sage	Sdept
2014001	李勇	男	20	CS
2014002	刘晨	女	19	IS
2014003	王敏	女	18	MA
2014004	张立	男	19	IS

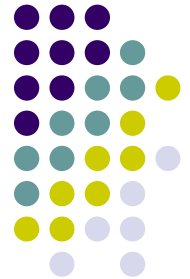
SC表

Sno	Cno	Grade
2014001	1	92
2014001	2	85
2014001	3	88
2014002	2	90
2014002	3	80



结果表

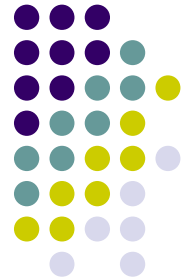
Student.Sno	Sname	Ssex	Sage	Sdept	SC.Sno	Cno	Grade
2014001	李勇	男	20	CS	2014001	1	92
2014001	李勇	男	20	CS	2014001	2	85
2014001	李勇	男	20	CS	2014001	3	88
2014002	刘晨	女	19	IS	2014002	2	90
2014002	刘晨	女	19	IS	2014002	3	80



- 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。
    - <表名1>.<列名1> = <表名2>.<列名2>
  - **SELECT语句不能直接实现自然连接**

[例33] 对[例32]用自然连接完成。

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



- 非等值连接
  - 连接运算符不为 = 的连接操作

[<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>

比较运算符：>、<、>=、<=、!=

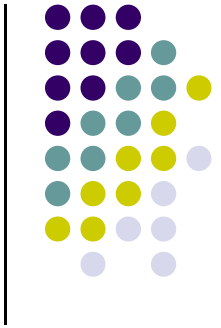
[<表名1>.]<列名1> BETWEEN [<表名2>.]<列名2> AND [<表名2>.]<列名3>

三、自身连接

- 一个表与其自己进行连接，称为表的自身连接
- 表示方法
  - 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34] 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；
```



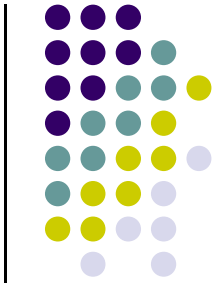
结果

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

SECOND表（Course表）

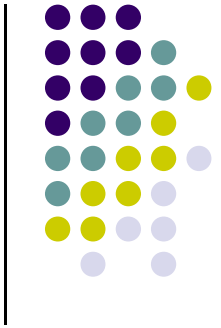
Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL语言	6	4



查询结果

Cno	Cpno
1	7
3	5
5	6

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

- 复合条件连接的类型
  - 两表按多个属性连接
  - 自身按多个属性连接
  - 多表连接

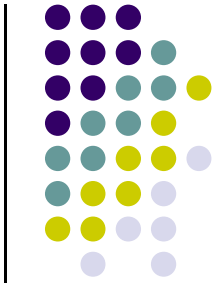
[例35] 假设学校中性别相同的学生不会重名。现如下设计学生表和选修表：

Std(Sname, Ssex, Sage, Sdept)

StdC(Sname, Ssex, Cno, Grade)

查询选修2号课程且成绩在90分以上的所有学生的姓名，性别及所在系。

```
SELECT Std.Sname, Std.Ssex, Sdept
FROM   Std，StdC
WHERE  Std.Sname = StdC.Sname   /* 连接谓词 */
    AND    Std.Ssex = StdC.Ssex /* 连接谓词 */
    AND    StdC.Cno= ‘2’         /* 其他限定条件 */
    AND    StdC.Grade>90；       /* 其他限定条件 */
```



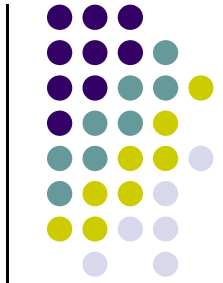
[例36] 查询每个学生的学号、姓名、选修的课程名及成绩。

```
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
2014001	李勇	数据库	92
2014001	李勇	数学	85
2014001	李勇	信息系统	88
2014002	刘晨	数学	90
2014002	刘晨	信息系统	80

3.3 查　询



3.3.1 单表查询

3.3.2 连接查询

3.3.3 嵌套查询

- 嵌套查询概述
- 嵌套查询分类
- 嵌套查询求解方法
- 引出子查询的谓词

3.3.4 集合查询

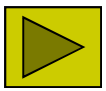
3.3.5 小结

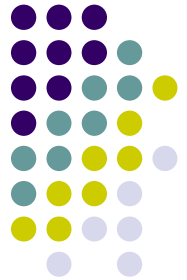
嵌套查询

- 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**

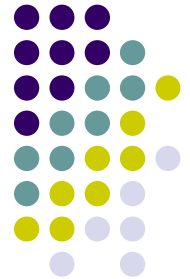
例

```
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 ');
```

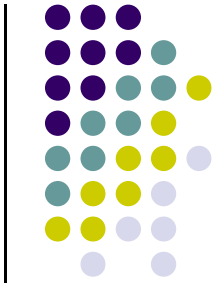





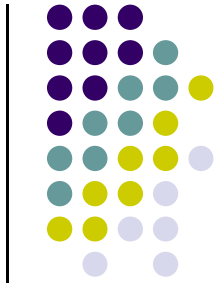
- 子查询的限制
  - 不能使用ORDER BY子句
- 层层嵌套方式反映了 SQL语言的结构化
- 有些嵌套查询可以用连接运算替代



- 嵌套查询分类
  - 不相关子查询（例如）
    - 子查询的查询条件不依赖于父查询
  - 相关子查询（例如）
    - 子查询的查询条件依赖于父查询
- 嵌套查询求解方法
  - 不相关子查询
    - 是由里向外逐层处理。即每个子查询在上一级查询处理之前求解，子查询的结果用于建立其父查询的查找条件。



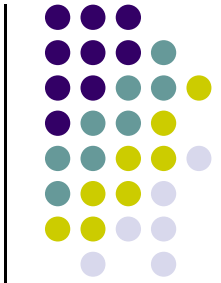
- 嵌套查询求解方法
  - 相关子查询
    - 首先取外层查询中表的第一个元组，根据它与内层查询相关的属性值处理内层查询，若WHERE子句返回值为真，则取此元组放入结果表；
    - 然后再取外层表的下一个元组；
    - 重复这一过程，直至外层表全部检查完为止。



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37] 查询与“刘晨”在同一个系学习的学生。

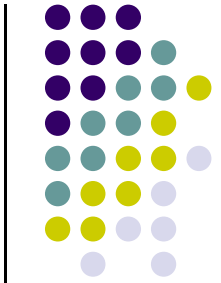
- 此查询要求可以分步来完成

① 确定“刘晨”所在系名

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

结果为：

Sdept
IS

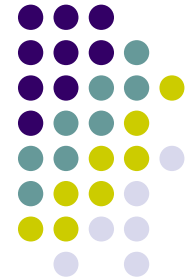


② 查找所有在IS系学习的学生。

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

结果为：

Sno	Sname	Sdept
2014001	刘晨	IS
2014004	张立	IS

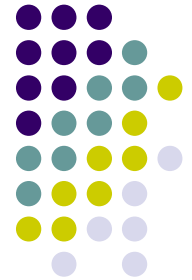


- 构造嵌套查询

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 ’ );
```

此查询为不相关子查询。DBMS求解该查询时也是分步去做的。



- 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’ ；
```

- 父查询和子查询中的表均可以定义别名

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

[例38]查询选修了课程名为“信息系统”的学生学号和姓名

– 嵌套查询

```
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

结果：

Sno	Sname
2014001	李勇
2014002	刘晨

- 本查询同样可以用连接查询实现：

```
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

- 使用范围
  - 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

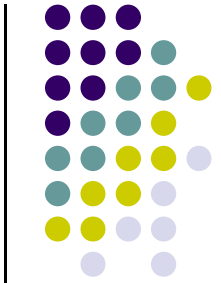
例：假设一个学生只可能在一个系学习，并且必须属于一个系，则在[例37]可以用 = 代替IN：

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

- 子查询一定要跟在比较符之后

  错误的例子：

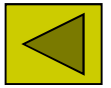
```
SELECT Sno，Sname，Sdept
FROM Student
WHERE ( SELECT Sdept
        FROM Student
        WHERE Sname= ' 刘晨 ' ) = Sdept;
```

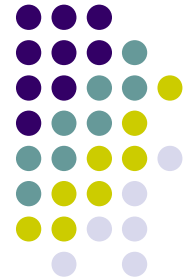


[例] 找出每个学生超过他选修课程平均成绩的课程号。

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

相关子查询





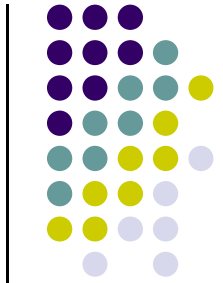
- 可能的执行过程：

1. 从外层查询中取出SC的一个元组x，将元组x的Sno值（2014001）传送给内层查询。

```
SELECT AVG(Grade)
FROM SC y
WHERE y.Sno='2014001';
```

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;
```



3. 执行这个查询，得到

（2014001，1）

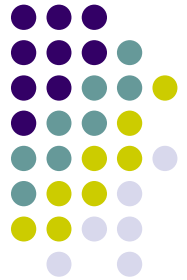
4.外层查询取出下一个元组重复做上述1至3步骤，直到外层的SC元组全部处理完毕。结果为:

（2014001，1）

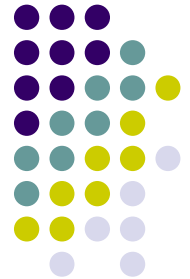
（2014001，3）

（2014002，2）

三、带有ANY或ALL谓词的子查询

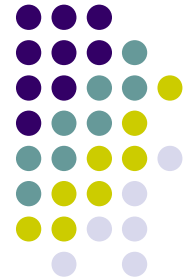


- 谓词语义
  - ANY：某个值
  - ALL：所有值



- 必须配合比较运算符使用

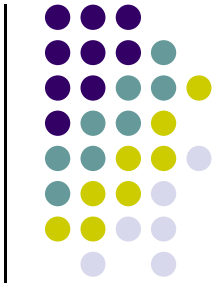
> ANY　　大于子查询结果中的某个值
>
> ALL　　大于子查询结果中的所有值
>
< ANY　　小于子查询结果中的某个值
>
< ALL　　小于子查询结果中的所有值
>
>= ANY　　大于等于子查询结果中的某个值
>
>= ALL　　大于等于子查询结果中的所有值
>
<= ANY　　小于等于子查询结果中的某个值
>
<= ALL　　小于等于子查询结果中的所有值
>
= ANY　　等于子查询结果中的某个值
>
=ALL 等于子查询结果中的所有值（通常没有实际意义）
>
!=（或<>）ANY　　不等于子查询结果中的某个值
>
!=（或<>）ALL　　不等于子查询结果中的任何一个值



[例39] 查询其他系中比信息系某一学生年龄小的学生姓名和年龄

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

/* 注意这是父查询块中的条件 */



结果

Sname	Sage
王敏	18

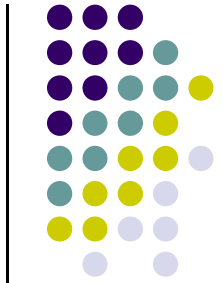
执行过程

DBMS执行此查询时，首先处理子查询，找出IS系中所有学生的年龄，构成一个集合(19，18)。然后处理父查询，找所有不是IS系且年龄小于19或18的学生。

- ANY和ALL谓词有时可以用集函数实现
  - ANY与ALL与集函数的对应关系

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

  - **用集函数实现子查询**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



[例39']：用集函数实现[例39]

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

[例40] 查询其他系中比信息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= ' IS ')
      AND Sdept <> ' IS ';
```

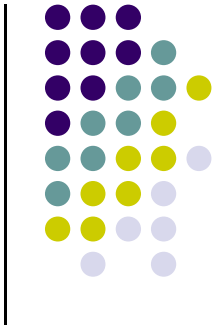
查询结果为空表。

方法二：用集函数

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

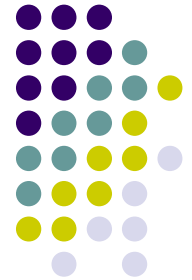
四、带有EXISTS谓词的子查询

1. EXISTS谓词
2. NOT EXISTS谓词
3. 不同形式的查询间的替换
4. 相关子查询的效率
5. 用EXISTS/NOT EXISTS实现全称量词（自学）
6. 用EXISTS/NOT EXISTS实现逻辑蕴函（自学）



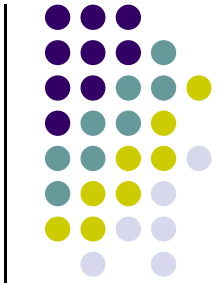
1. EXISTS谓词
   - 存在量词∃
   - 带有EXISTS谓词的子查询不返回任何数据，只产生逻辑真值“true”或逻辑假值“false”。
     - 若内层查询结果非空，则返回真值
     - 若内层查询结果为空，则返回假值
   - 由EXISTS引出的子查询，其目标列表达式通常都用*，因为带EXISTS的子查询只返回真值或假值，给出列名无实际意义
2. NOT EXISTS谓词

[例41] 查询所有选修了1号课程的学生姓名。



思路分析：

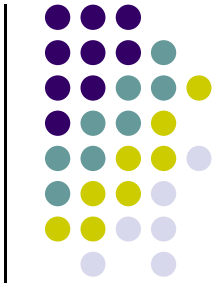
- 本查询涉及Student和SC关系。
- 在Student中依次取每个元组的Sno值，用此值去检查SC关系。
- 若SC中存在这样的元组，其Sno值等于此Student.Sno值，并且其Cno= '1'，则取此Student.Sname送入结果关系。



– 用嵌套查询

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

是相关子查询。



- 可用连接运算实现

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

[例42] 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');
```

此例用连接运算难于实现

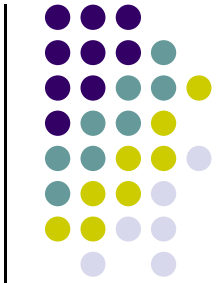
3. 不同形式的查询间的替换

- 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。

例[43]：[例37]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’);
```



4. 相关子查询的效率

- 由于带EXISTS量词的相关子查询只关心内层查询是否有返回值，并不需要查具体值，因此其效率并不一定低于其他形式的查询。

相关子查询的效率可能高于连接查询

例[44]：查询选修了课程的学生姓名

法一：

```
SELECT Sname
FROM Student
WHERE EXISTS
  (SELECT *
   FROM SC
   WHERE Sno=Student.Sno);
```

法二：

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

法三：

```
SELECT Sname
FROM Student
WHERE Sno in
  (SELECT distinct sno
   FROM SC);
```

- 用EXISTS/NOT EXISTS实现全称量词(难点)

  SQL语言中没有全称量词∀ （For all）

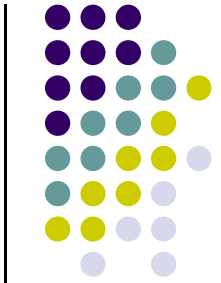
  可以把带有全称量词的谓词转换为等价的带有存在量词的谓词：

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

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：

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



[例]查询至少选修了学生2014002选修的全部课程的学生号码。

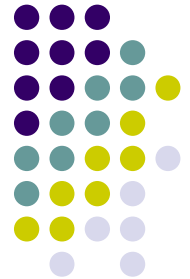
解题思路：

- 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2014002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2014002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: $(\forall y)\, p \rightarrow q$



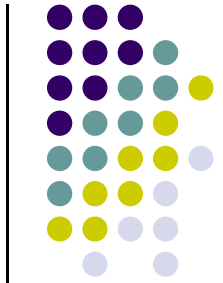
- 等价变换：

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

- 变换后语义：不存在这样的课程y，学生2014002选修了y，而学生x没有选。

- 用NOT EXISTS谓词表示：

```
SELECT DISTINCT Sno
  FROM SC SCX
  WHERE NOT EXISTS
            (SELECT *
             FROM SC SCY
             WHERE SCY.Sno = ' 2014002 '  AND
                       NOT EXISTS
                       (SELECT *
                        FROM SC SCZ
                        WHERE SCZ.Sno=SCX.Sno AND
                                  SCZ.Cno=SCY.Cno));
```



3.3 查　询

3.3.1 单表查询

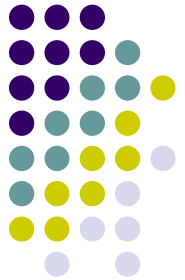
3.3.2 连接查询

3.3.3 嵌套查询

3.3.4 集合查询

3.3.5 小结

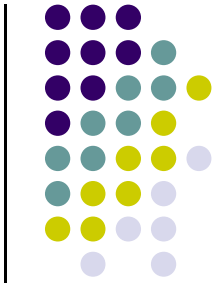
3.3.4 集合查询



- 标准SQL直接支持的集合操作种类
  - 并操作(UNION)

- 一般商用数据库支持的集合操作种类
  - 并操作(UNION)
  - 交操作(INTERSECT)
  - 差操作(EXCEPT)

1． 并操作



- 形式

  <查询块>

  UNION[ALL]

  <查询块>

  - 参加UNION操作的各结果表的列数必须相同；对应项的数据类型也必须相同
  - UNION结果自动消除重复行；
  - UNION ALL 结果包括所有重复行

[例45] 查询计算机科学系的学生及年龄不大于19岁的学生。

方法一：

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

方法二：

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

使用UNION将多个查询结果合并起来，系统会自动去掉重复元组。

[例46] 查询选修了课程1或者选修了课程2的学生。

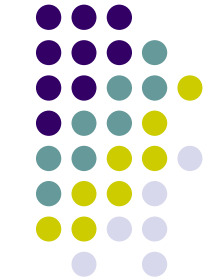
方法一：

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

方法二：

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

2． 交操作



- 标准SQL中没有提供集合交操作，但可用其他方法间接实现。

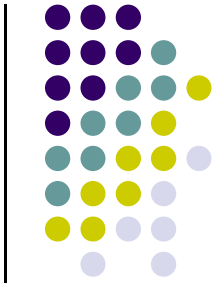
[例48] 查询计算机科学系的学生与年龄不大于19岁的学生的交集

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

本例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;
```

3． 差操作



- 标准SQL中没有提供集合差操作，但可用其他方法间接实现。

[例51] 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；
```

本例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;
```

4. 对集合操作结果的排序

- **ORDER BY子句只能用于对最终查询结果排序，不能对中间结果排序**
- 任何情况下，ORDER BY子句只能出现在最后
- 对集合操作结果排序时，ORDER BY子句中用数字指定排序属性，也可用属性名

[例53]

错误写法

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

正确写法

```
SELECT *
FROM Student
WHERE Sdept=
'CS'
  UNION
  SELECT *
  FROM Student
  WHERE Sage<=19
  ORDER BY 1;
```

3.3 查　询

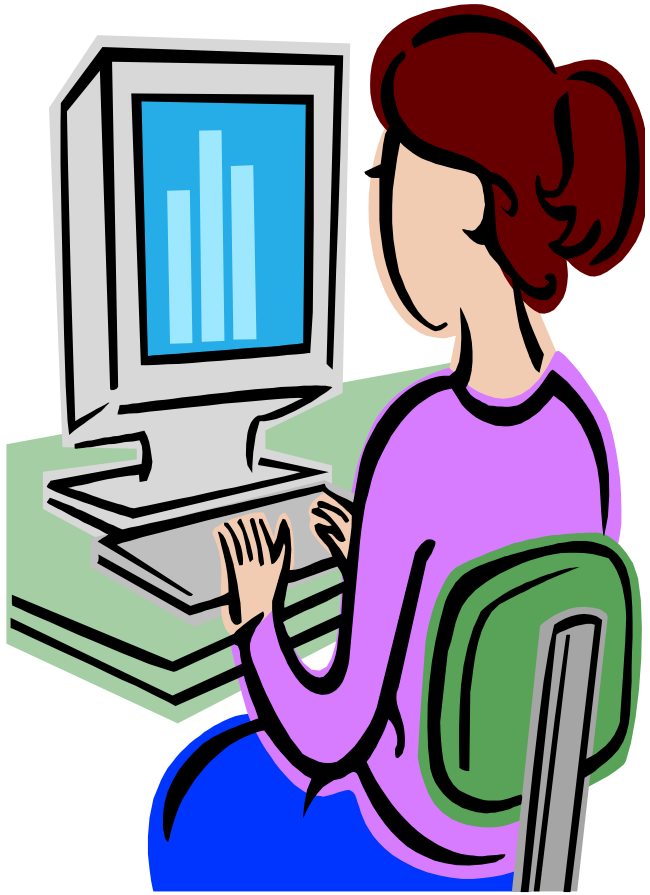
3.3.1 单表查询

3.3.2 连接查询

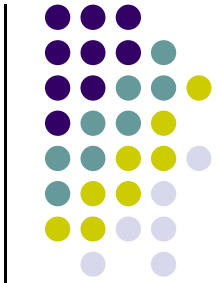
3.3.3 嵌套查询

3.3.4 集合查询

3.3.5 小结

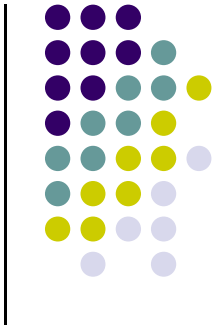


3.3.5 小结



- SELECT语句的一般格式

SELECT [ALL|DISTINCT] <目标列表达式>
[别名] [，<目标列表达式> [别名]] …
FROM <表名或视图名> [别名]
[，<表名或视图名> [别名]] …
[WHERE <条件表达式>]
[GROUP BY <列名1>[，<列名1’>] ...
[HAVING <条件表达式>]]
[ORDER BY <列名2> [ASC|DESC]
[，<列名2’> [ASC|DESC]] …];

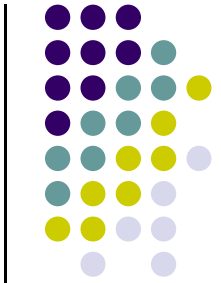


- 目标列表达式
  - 目标列表达式格式

(1) [<表名>.]*

(2) [<表名>.]<属性列名表达式>[，[<表名>.]<属性列名表达式>] …

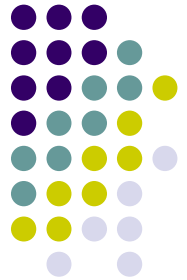
<属性列名表达式>：由属性列、作用于属性列的集函数和常量的任意算术运算（+，-，*，/）组成的运算公式。



- 集函数格式

{ COUNT / SUM / AVG / MAX / MIN } ([DISTINCT|ALL] <列名>)

COUNT ([DISTINCT|ALL] *)



- 条件表达式格式

（1）

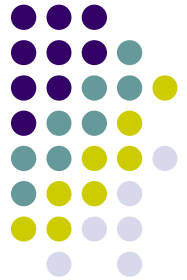
<属性列名> θ { <属性列名> | <常量> | [ANY|ALL] (SELECT语句) }

（2）

<属性列名> [NOT] BETWEEN { <属性列名> | <常量> | (SELECT 语句) } AND { <属性列名> | <常量> | (SELECT 语句) }

（3）

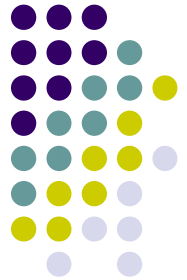
<属性列名> [NOT] IN { (<值1>[，<值2>] …) | (SELECT语句) }



(4) <属性列名> [NOT] LIKE <匹配串>

(5) <属性列名> IS [NOT] NULL

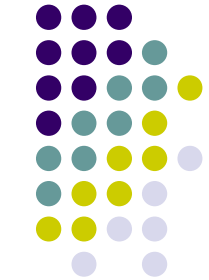
(6) [NOT] EXISTS (SELECT语句)



（7）

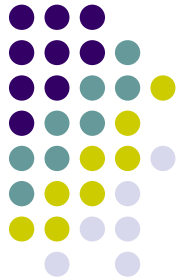
<条件表达式> { AND / OR } <条件表达式> ({ AND / OR } <条件表达>) …

第三章 关系数据库标准语言SQL



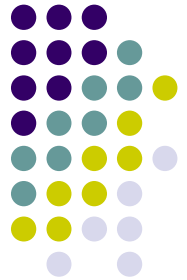
3.1 SQL概述
3.2 数据定义
3.3 数据查询
3.4 数据更新
3.4.1 插入数据
3.4.2 修改数据
3.4.3 删除数据
3.5 视图

3.4.1 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

1. 插入单个元组



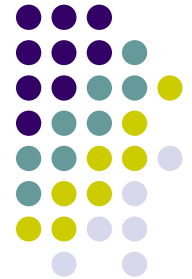
- 语句格式

  INSERT

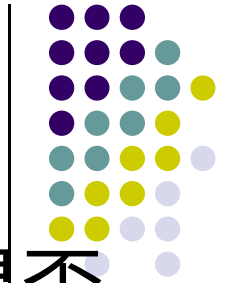
  INTO <表名> [(<属性列1>[，<属性列2 >…)]

  VALUES (<常量1> [，<常量2>]…)

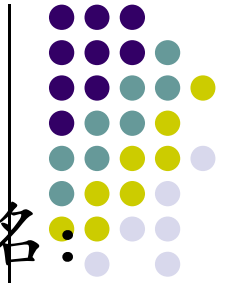
  - 功能
    - 将新元组插入指定表中。



- INTO子句
  - 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，且属性列属性与表定义中的顺序一致
  - 指定部分属性列：插入的元组在其余属性列上取空值
- VALUES子句
  - 提供的值必须与INTO子句匹配
    - 值的个数
    - 值的类型



- 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



[例1] 将一个新学生记录（学号：2014020；姓名：陈冬；性别：男；所在系：IS；年龄：18岁）插入到Student表中。

```
INSERT
  INTO Student
  VALUES ('2014020', '陈冬', '男', 'IS', 18);
```

[例2] 插入一条选课记录('2014020', '1')。

```
INSERT
 INTO SC(Sno, Cno)
 VALUES ('2014020', '1');
```

新插入的记录在Grade列上取空值

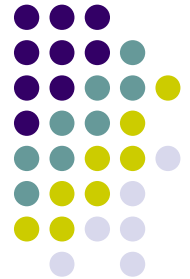
2. 插入子查询结果

INSERT INTO <表名>

[(<属性列1> [，<属性列2>…)]

子查询;

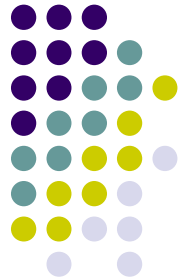
- 功能
  - 将子查询结果插入指定表中



- 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

- 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

[例3] 对每一个系，求学生的平均年龄，并把结果存入数据库。



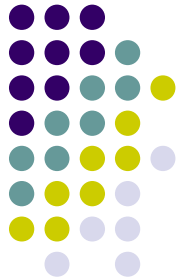
第一步：建表

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

第二步：插入数据

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

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

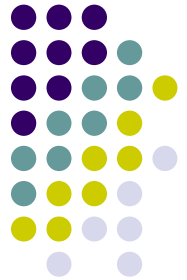
3.4.2 修改数据

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
  - 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

- 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

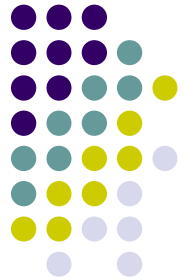
1. 修改某一个元组的值



[例4] 将学生2014001的年龄改为22岁。

```
UPDATE  Student
SET Sage=22
WHERE  Sno=' 2014001 ';
```

2. 修改多个元组的值



[例5] 将所有学生的年龄增加1岁。

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

[例6] 将信息系所有学生的年龄增加1岁。

```
UPDATE Student
SET Sage= Sage+1
WHERE Sdept=' IS ';
```

3. 带子查询的修改语句

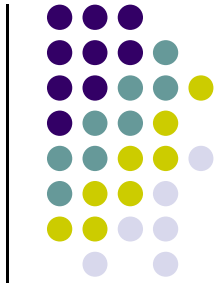
[例7] 将计算机科学系全体学生的成绩置零。

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

[例7] 解法2

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

3.4 数据更新

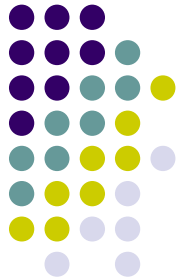


3.4.1 插入数据

3.4.2 修改数据

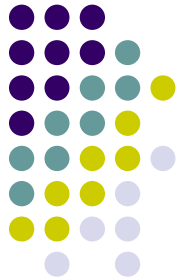
3.4.3 删除数据

3.4.3 删除数据

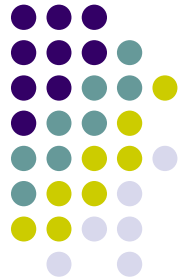


DELETE
FROM <表名>
[WHERE <条件>];

- 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
  - 缺省表示要修改表中的所有元组



- DBMS在执行删除语句时会检查所删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



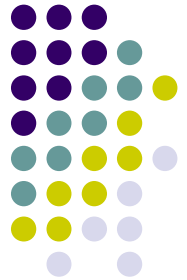
- 三种删除方式
  - 删除某一个元组的值
  - 删除多个元组的值
  - 带子查询的删除语句

1. 删除某一个元组的值

[例8] 删除学号为2014019的学生记录。

```
DELETE
FROM Student
WHERE Sno='2014019';
```

2. 删除多个元组的值



[例9] 删除2号课程的所有选课记录。

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

[例10] 删除所有的学生选课记录。

```
DELETE
FROM SC;
```

3. 带子查询的删除语句

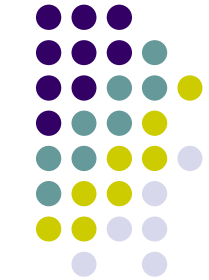
[例11] 删除计算机科学系所有学生的选课记录。

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

解法2

```
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

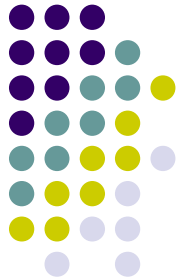
3.3 数据查询

3.4 数据更新

3.5 视图

3.5 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
  - 基于视图的操作
    - 定义视图(DDL)
      - 建立
      - 定义基于该视图的新视图
      - 删除
    - 查询(DML)
    - 受限更新(DML)



3.5 视　图

3.5.1 定义视图

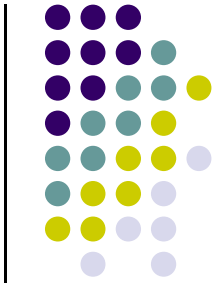
1. 建立视图
2. 删除视图

3.5.2 查询视图

3.5.3 更新视图

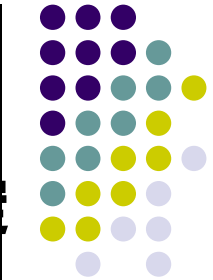
3.5.4 视图的作用

1. 建立视图

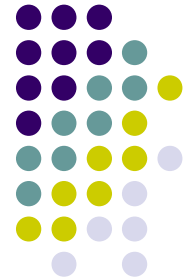


- CREATE VIEW <视图名>
  [(<列名> [，<列名>]…)]
  AS <子查询>
  [WITH CHECK OPTION];
  - 子查询
    - **通常**不含ORDER BY子句和DISTINCT短语的SELECT语句
  - WITH CHECK OPTION
    - 透过视图进行增删改操作时，不得破坏视图定义中的谓词条件（即子查询中的条件表达式）

- 组成视图的属性列名**或全部省略或全部指定**
  - 省略视图的各个属性列名，则隐含该视图由子查询中SELECT子句目标列中的诸字段组成。
  - 必须明确指定组成视图的所有列名的情形
    (1) 某个目标列不是单纯的属性名，而是集函数或列表达式
    (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语句的结果**只是把视图的定义存入数据字典**，并不执行其中的SELECT语句。只是在对视图查询时，才按视图的定义从基本表中将数据查出。
- 常见的视图形式
  - 行列子集视图
  - WITH CHECK OPTION的视图
  - 基于多个基表的视图
  - 基于视图的视图
  - 带表达式的视图
  - 分组视图



- 行列子集视图
  - 从单个基本表导出
  - 只是去掉了基本表的某些行和某些列，但保留了主码

[例1] 建立信息系学生的视图。

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

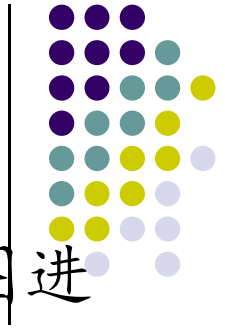
行列子集视图视图IS_Student由Sno，Sname，Sage三列组成

WITH CHECK OPTION的视图

[例2] 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。

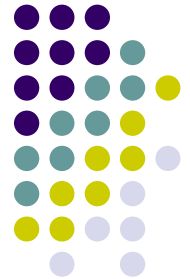
```
CREATE VIEW IS_Student
    AS
    SELECT Sno，Sname，Sage
    FROM  Student
    WHERE  Sdept= 'IS'
    WITH CHECK OPTION；
```

- 对IS_Student视图的更新操作
  - 修改操作：DBMS自动加上Sdept= 'IS'的条件
  - 删除操作：DBMS自动加上Sdept= 'IS'的条件
  - 插入操作：DBMS自动检查Sdept属性值是否为'IS'
    - 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'IS'



[例3] 建立1号课程的选课视图，并要求透过该视图进行的更新操作只涉及1号课程，同时对该视图的任何操作只能在工作时间进行。

```
CREATE VIEW IS_SC
        AS
        SELECT Sno,Cno,Grade
        FROM  SC
        WHERE  Cno= '1'
        AND cast(substring(Convert(varchar(20),GETDATE()),12,2) As
int) BETWEEN 9 AND 17
        AND datename(weekday, getdate()) in('星期一','星期二','星期三','
星期四','星期五')
        WITH CHECK OPTION;
```



- 基于多个基表的视图

[例4] 建立信息系选修了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';
```

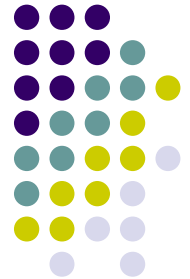
- 由于视图IS_S1的属性列中包含了Student表与SC表的同名列Sno，所以必须在视图名后面明确说明视图的各个属性列名。

基于视图的视图

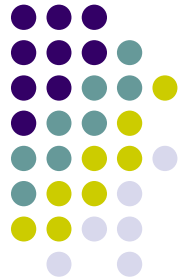
[例5] 建立信息系选修了1号课程且成绩在90分以上的学生的视图。

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

- 视图IS_S2建立在视图IS_S1之上



- 带表达式的视图
  - 在设计数据库时，为了减少数据冗余，基本表中只存放基本数据，由基本数据经过各种计算派生出的数据一般是不存储的。
  - 视图中的数据并不实际存储，所以定义视图时可以根据应用的需要，设置一些派生属性列，以方便应用程序的编制。
  - 派生属性称为虚拟列。带虚拟列的视图称为带表达式的视图。
  - 带表达式的视图必须明确定义组成视图的各个属性列名。



[例6] 定义一个反映学生出生年份的视图。

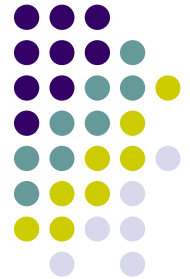
```
CREATE  VIEW BT_S(Sno，Sname，Sbirth)
    AS  SELECT Sno，Sname，2015-Sage
        FROM  Student；
```

- 分组视图
  - 用带集函数和GROUP BY子句的查询来定义的视图称为分组视图
  - 分组视图必须明确定义组成视图的各个属性列名

[例7] 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

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



- 一类不易扩充的视图
  - **以 SELECT * 方式创建的视图可扩充性差，应尽可能避免**

[例8] 将Student表中所有女生记录定义为一个视图

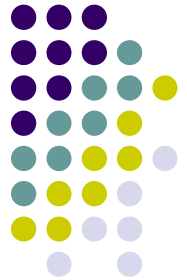
```
CREATE VIEW F_Student1
                    (stdnum，name，sex，age，dept)
  AS  SELECT *
      FROM  Student
      WHERE Ssex='女';
```

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。

```
CREATE VIEW F_Student2
                    (stdnum，name，sex，age，dept)
  AS  SELECT Sno，Sname，Ssex，Sage，Sdept
      FROM  Student
      WHERE Ssex='女';
```

为基本表Student增加属性列不会破坏Student表与F_Student2视图的映象关系。

2. 删除视图

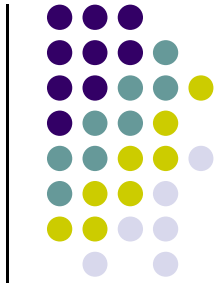


- DROP VIEW <视图名>;
  - 该语句从数据字典中删除指定的视图定义
  - 由该视图导出的其他视图定义仍在数据字典中，但已不能使用，必须显式删除
  - 删除基本表时，由该基表导出的所有视图定义都必须显式删除

[例9] 删除视图IS_S1

```
DROP VIEW IS_S1;
```

执行此语句后，IS_S1视图的定义将从数据字典中删除。由IS_S1视图导出IS_S2视图已无法使用，但其定义虽然仍在数据字典中。



3.5 视　图

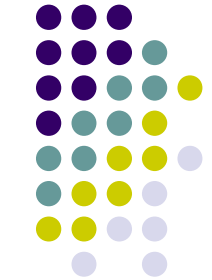
3.5.1 定义视图

3.5.2 查询视图

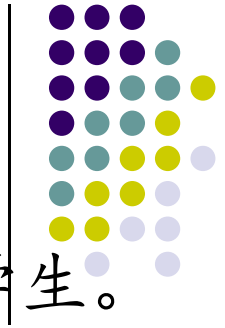
3.5.3 更新视图

3.5.4 视图的作用

3.5.2 查询视图



- 从用户角度而言，查询视图与查询基本表的方法相同
- DBMS实现视图查询的方法
  - 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

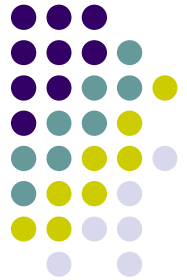


[例1] 在信息系学生的视图中找出年龄小于20岁的学生。

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

IS_Student视图的定义(视图定义例1)：

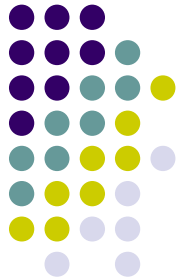
```
CREATE VIEW IS_Student
     AS
     SELECT Sno，Sname，Sage
     FROM  Student
     WHERE  Sdept= 'IS‘；
```



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；
```



- 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。采用视图消解法的DBMS会限制这类查询。（此例在SQLServer2008及以上中可以正确执行）

[例3] 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;
```

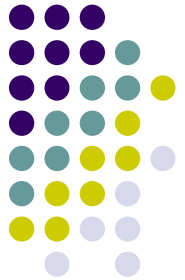
转换后的查询：

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

WHERE子句中是不能用集函数作为条件表达式的

正确转换：

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



3.5 视　图

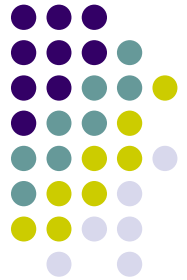
3.5.1 定义视图

3.5.2 查询视图

3.5.3 更新视图

3.5.4 视图的作用

3.5.3 更新视图



- 从用户角度而言，更新视图与更新基本表的方法相同
- 定义视图时指定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数据进行增加、删除、修改时，操作不属于视图范围内的基本表数据

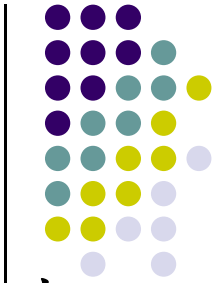
[例1] 将信息系学生视图IS_Student中学号为2014002的学生姓名改为“刘辰”。

```
UPDATE  IS_Student
SET  Sname= '刘辰'
WHERE  Sno= ‘2014002’;
```

- 视图消解法

转换后的查询语句为：

```
UPDATE Student
SET Sname= '刘辰'
WHERE Sno= ‘2014002' AND Sdept= 'IS';
```



[例2] 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2014029，姓名为赵新，年龄为20岁。

```
INSERT
INTO IS_Student
VALUES('2014029'，'赵新'，20);
```

转换为对基本表的更新：

```
INSERT
INTO  Student(Sno，Sname，Sage，Sdept)
VALUES('2014029'，'赵新'，20，'IS' );
```

[例3] 删除计算机系学生视图CS_S中学号为2014029的记录

```
DELETE
FROM IS_Student
WHERE Sno= ‘2014029';
```

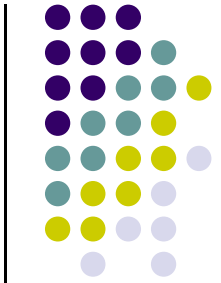
转换为对基本表的更新：

```
DELETE
FROM Student
WHERE Sno= ‘2014029' AND Sdept= 'IS';
```

- DBMS对视图更新的限制
  - 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：视图S_G为不可更新视图。

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



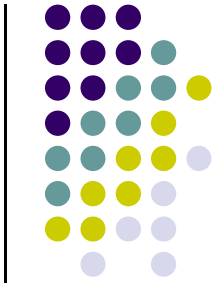
S_G “平均成绩” Gavg属性列为导出列

对于如下更新语句：

```
UPDATE S_G
SET Gavg=90
WHERE Sno= ‘2014001';
```

无法将其转换成对基本表SC的更新

- 视图的可更新性
  - 行列子集视图是可更新的。
  - 除行列子集视图外，还有些视图理论上是可更新的，但它们的确切特征还是尚待研究的课题。
  - 还有些视图从理论上是不可更新的。
- 不可更新的视图与不允许更新的视图是两个不同的概念
- 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



3.5 视　图

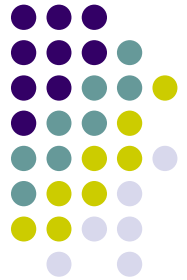
3.5.1 定义视图

3.5.2 查询视图

3.5.3 更新视图

3.5.4 视图的作用（自学）

3.5.4 视图的作用



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

1. 视图能够简化用户的操作

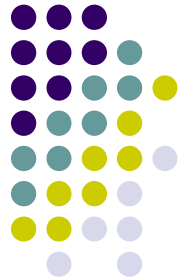
- 当视图中数据不是**直接**来自基本表时，定义视图能够简化用户的操作
  - 基于多张表连接形成的视图
  - 基于复杂嵌套查询的视图
  - 含导出属性的视图

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

- 物理独立性与逻辑独立性的概念
- 视图在一定程度上保证了数据的逻辑独立性



例：数据库逻辑结构发生改变

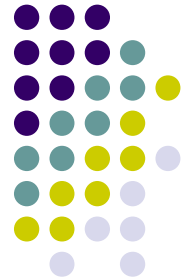
将学生关系

Student(Sno，Sname，Ssex，Sage，Sdept)

“垂直”地分成两个基本表：

SX(Sno，Sname，Sage)

SY(Sno，Ssex，Sdept)



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；
```

使用户的外模式保持不变，从而对原Student表的查询程序不必修改

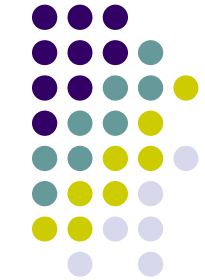
- 视图只能在一定程度上提供数据的逻辑独立性
  - 由于对视图的更新是有条件的，因此应用程序中修改数据的语句可能仍会因基本表结构的改变而改变。

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

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 视图

课上习题

设有一个SPJ数据库，包括S，P，J，SPJ四个关系模式：

S（SNO，SNAME，STATUS，CITY)；　P（PNO，PNAME，COLOR，WEIGHT）；

J（JNO，JNAME，CITY）；　SPJ（SNO，PNO，JNO，QTY）；

供应商表S由供应商代码（SNO）、供应商姓名（SNAME）、供应商状态（STATUS）、供应商所在城市（CITY）组成；**零件表P**由零件代码（PNO）、零件名（PNAME）、颜色（COLOR）、重量（WEIGHT）组成；**工程项目表J**由工程项目代码（JNO）、工程项目名（JNAME）、工程项目所在城市（CITY）组成；**供应情况表SPJ**由供应商代码（SNO）、零件代码（PNO）、工程项目代码（JNO）、供应数量（QTY）组成，表示某供应商供应某种零件给某个工程项目的数量为QTY。

针对该数据库，完成如下题目要求：

1. 用SQL语句创建SPJ表，要求定义主码，外码。
2. 写出“查询使用供应商S1所供应零件的工程号码”的SQL语句。
3. 写出“查询工程项目J2使用的各种零件的名称及其数量”的SQL语句。
4. 写出“查询上海厂商供应的所有零件的名称”的SQL语句。
5. **UPDATE** SPJ **SET** SNO='S2'
   WHERE SNO='S1' **AND** PNO='P6' **AND** JNO='J4'; 说明该SQL语句功能
6. **CREATE VIEW** V_SPJ **AS**
   SELECT SNO,PNO,CITY **FROM** J,SPJ
   WHERE J.JNO=SPJ.JNO **AND** JNAME='三建'；说明该SQL语句的功能

1．用SQL语句创建SPJ表，要求定义主码，外码。

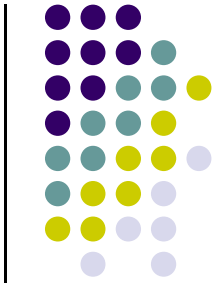
参考答案：

```
CREATE TABLE SPJ(SNO CHAR(8),
                 PNO CHAR(10),
        JNO CHAR(10),
        QTY int,
        PRIMARY KEY(SNO,PNO,JNO),
        FOREIGN KEY(SNO) REFERENCES S(SNO),
        FOREIGN KEY(PNO) REFERENCES P(PNO),
        FOREIGN KEY(JNO) REFERENCES J(JNO));
```

2．写出“查询使用供应商S1所供应零件的工程号码”的SQL语句。

参考答案：

```
SELECT JNO
FROM SPJ
WHERE SNO='S1';
```



3．写出“查询工程项目**J2**使用的各种零件的名称及其数量”的**SQL**语句。

参考答案：

```
SELECT PNAME,QTY
FROM P,SPJ
WHERE P.PNO=SPJ.PNO AND JNO='J2';
```

4．写出“查询上海厂商供应的所有零件的名称”的**SQL**语句。

参考答案：

```
SELECT PNAME
FROM S,P,SPJ
WHERE S.SNO=SPJ.SNO AND P.PNO=SPJ.PNO AND CITY='上海';
```

5. UPDATE SPJ SET SNO='S2'
WHERE SNO='S1' AND PNO='P6' AND JNO='J4'; 说明该SQL语句功能

参考答案：由S1供给J4的零件P6改为由S2供应。

6. CREATE VIEW V_SPJ AS
SELECT SNO,PNO,CITY FROM J,SPJ
WHERE J.JNO=SPJ.JNO AND JNAME='三建'；说明该SQL语句的功能

参考答案：为三建工程项目建立一个供应情况的视图，包括供应商代码、零件代码、供应数量。

课下复习

- “学生—课程”数据库中包括三个表：

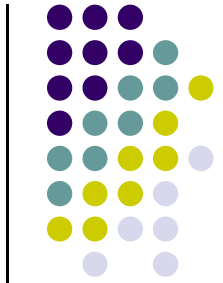
(1) “学生”表：Student由学号（Sno）、姓名（Sname）、性别（Ssex）、年龄（Sage）、所在系（Sdept）五个属性组成，可记为：
Student(Sno,Sname,Ssex,Sage,Sdept) **Sno**

(2) “课程”表：Course由课程号（Cno）、课程名（Cname）、先修课号（Cpno）、学分（Ccredit）四个属性组成，可记为： Course(Cno,Cname,Cpno,Ccredit) **Cno**

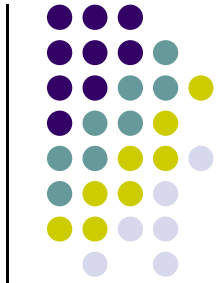
(3) “学生选课”表：SC由学号（Sno）、课程号（Cno）、成绩（Grade）三个属性组成，可记为：
SC(Sno,Cno,Grade) **(SNO, CNO)**

1、创建学生—课程数据库中的学生表（Student），课程表（Course）和选课表（SC），要求定义相应的主码和外码，以及基本的约束条件。

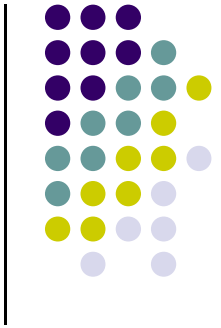
```
CREATE  TABLE  Student
        ( Sno    CHAR（8）,
          Sname  CHAR（8）   NOT NULL,
          Sage   INT,
          Ssex   CHAR(1),
          Sdept  CHAR(2),
          PRIMARY KEY (Sno),
          CHECK (Ssex='0' OR  Ssex='1')
        );
```



```
CREATE  TABLE  Course
        ( Cno   CHAR（4）,
          Cname  CHAR（10）   NOT NULL,
          Cpno   CHAR(4),
           Ccredit  FLOAT,
          PRIMARY KEY (Cno)
        );
```



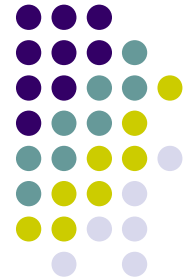
```
CREATE  TABLE  SC
        (Sno   CHAR（8）,
         Cno   CHAR（4）,
         Grade  INT,
          PRIMARY KEY (Sno，  Cno),
          FOREIGN KEY   (Sno)
                REFERENCES   Student(Sno),
          FOREIGN KEY (Cno)
                REFERENCES   Course(Cno),
          CHECK((Grade IS NULL) OR
                Grade BETWEEN 0 AND 100))；
```



2、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 Couse(Cno);
CREATE UNIQUE INDEX SCno ON SC(Sno ASC,Cno DESC);

课堂测试



- **教师关系T(T# ,TNAME,TTITLE)**
- **课程关系C(C#, CNAME,T#)**
- **学生关系S (S#, SNAME, AGE,SEX)**
- **选课关系SC(S#,C#,SCORE)**

试用SQL的查询语句表示下列查询：

1. 查询年龄小于17岁的女学生的学号和姓名。
2. 查询男同学所学课程的课程号和课程名。
3. 查询男同学所学课程的任课老师的工号和姓名。
4. 查询至少选修两门课程的学生学号。
5. 查询王同学不学的课程的课程号。
6. 查询至少有学号为S2和S4学生选修的课程的课程号。
7. ~~查询全部学生都选修的课程的课程号与课程名。~~
8. ~~查询选修课程包含LIU老师所授全部课程的学生学号。~~