第4章
关系数据库标准语言SQL

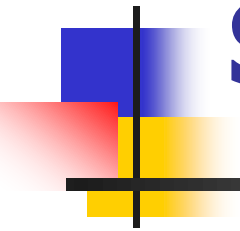
北京理工大学　计算机学院
张文耀
zhwenyao@bit.edu.cn

主要内容

- 4.1SQL简介
- 4.2SQL的系统结构

SQL概述

- 4.3SQL的数据定义 —— SQL数据定义
- 4.4SQL的数据操纵
  - 4.4.1 数据查询 —— SQL数据查询
  - 4.4.2 数据更新 —— SQL数据更新
- 4.5SQL中的视图 —— SQL的视图
- 4.6SQL的数据控制 —— SQL数据控制
- 4.7嵌入式SQL —— 嵌入式SQL
- 4.8小结



SQL概述

主要内容

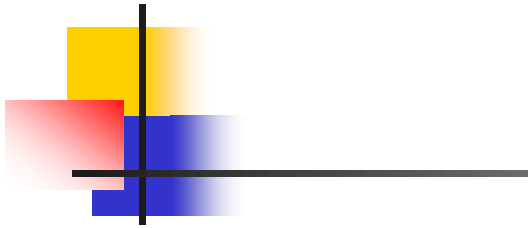
- SQL简介
- SQL的特点
- SQL的系统结构

SQL简介

- **SQL（Structured Query Language）**
  结构化查询语言
  - 是一种介于关系代数与关系演算之间的语言；
  - 是一个通用的、功能极强的关系数据库语言；
  - 目前已成为关系数据库的标准语言，大多数关系数据库产品都支持**SQL**语言；
  - 其前身是**1974**年**Boyce**和**Chamberlin**提出的，并在**System R**上实现的**SQURARE**语言。

- **SQL语言的版本包括：**
  - **SQL-86**
  - **SQL-89**
  - **SQL-92(SQL2)**
  - **SQL:1999(SQL3)** 增加了面向对象的概念, 超**1000**页
  - **SQL:2003(SQL4)**
  - **SQL:2008**
  - **SQL:2011**

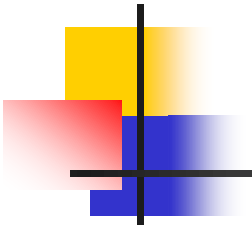
2016年12月14日，ISO/IEC发布了最新版本的数据库语言SQL标准（ISO/IEC 9075:2016）



- **SQL语言按功能划分为四部分：**
  - 数据定义
    定义表、视图和索引
  - 数据操纵
    查询、插入、删除和修改
  - 数据控制
    访问权限管理、事务管理
  - 嵌入式**SQL**
    SQL语句嵌入到程序语言中使用

SQL的特点

- 综合统一
  - 集数据定义语言**DDL**、数据操纵语言**DML**、数据控制语言**DCL**的于一体，可以完成数据库生命周期中的全部活动。
  - 关系模型中实体和实体间的联系都用关系来表示，使得操作符单一，每种操作只使用一个操作符。
- 高度非过程化
  - 使用**SQL**语言，只需要提出“做什么”，而无需指明“怎么做”，无需了解存取路径，提高了数据的独立性。
- 面向集合的操作方式
  - **SQL**语言采用集合操作方式，查询、插入、删除、修改操作的对象都是集合。



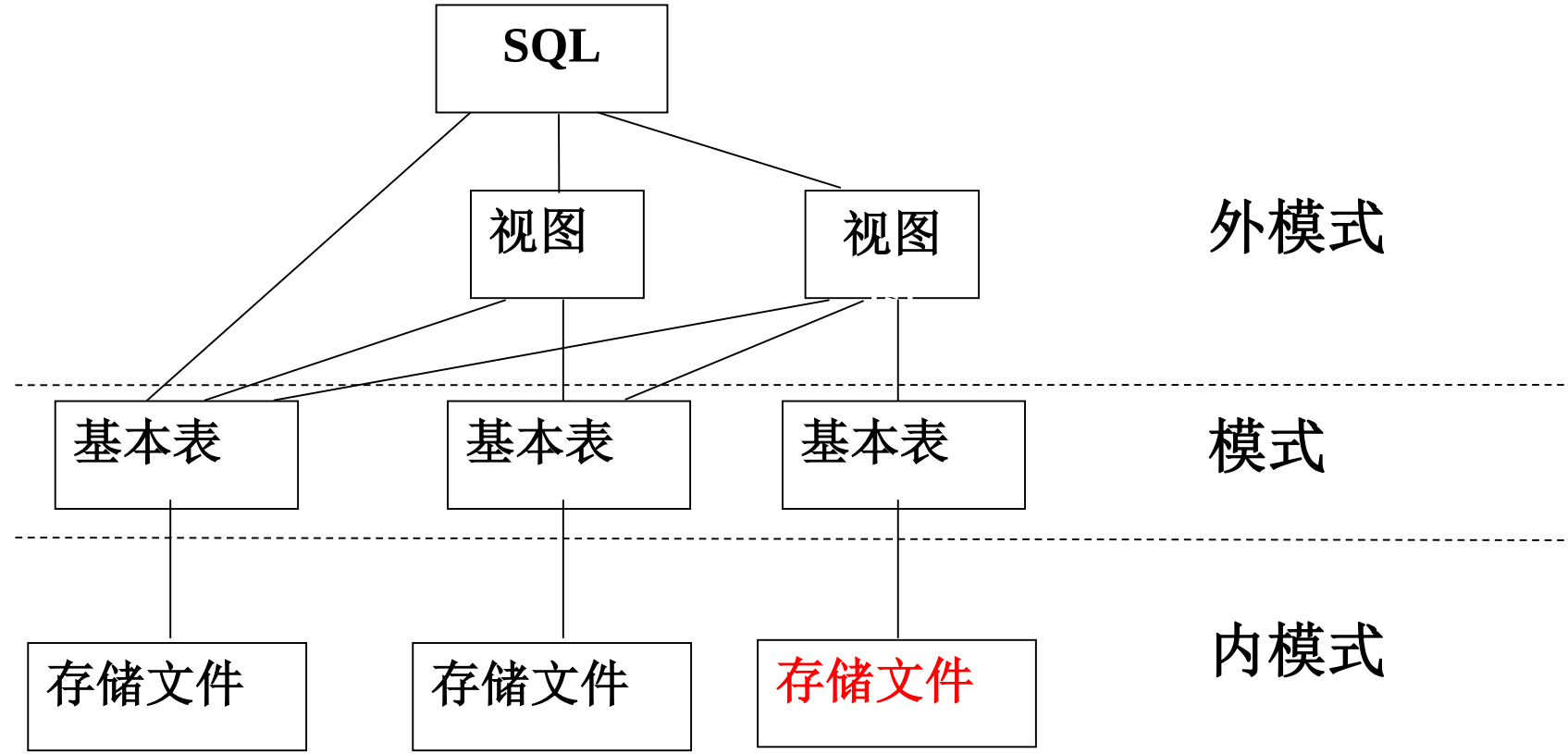
- 以同一种语法结构提供两种使用方式
  - 作为独立的语言（交互式SQL）
    提供联机交互工具，在终端键盘上直接键入SQL命令对数据库进行操作，由DBMS来进行解释；
  - 作为嵌入式语言（嵌入式SQL）
    SQL语句能嵌入到高级语言程序中，使应用程序具备利用SQL访问数据库的能力。
  - 不同方式下，SQL的语法结构基本上是一致的，提供了极大的灵活性和方便性。
- 语言简洁，易学易用
  - 完成核心功能只用了9个动词。

SQL 的动词

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

SQL的系统结构

- **SQL语言支持数据库的三级模式结构**
  - 在**SQL**中，关系模式称为基本表（**Base Table**），基本表的集合形成数据库模式，对应三级模式结构的模式
  - 基本表在物理上与存储文件相对应，所有存储文件的集合为物理数据库
  - 外模式由视图**(View)** 组成



SQL的三级模式结构

- **SQL的表分为两种：基本表和视图**
- 基本表（**Base Table，Table**）
  - 独立存在的表
  - 一个关系模式对应一个基本表
- 视图（**View**）
  - 是从一个或多个基本表中导出的表，仅有逻辑上的定义，不实际存储数据，是一种虚表。
  - 视图的定义存储在数据字典中，在使用的时候，根据定义从基本表中导出数据供用户使用。
  - 视图可以象基本表一样进行查询和某些更新操作。

SQL的数据定义

第4章 关系数据库标准语言SQL

- 4.1SQL简介
- 4.2SQL的系统结构
- 4.3SQL的数据定义
- 4.4SQL的数据操纵
  - 4.4.1 数据查询
  - 4.4.2 数据更新
- 4.5SQL中的视图
- 4.6SQL的数据控制
- 4.7嵌入式SQL
- 4.8小结

主要内容

- **SQL**的数据定义功能
- **SQL**模式的定义
- **SQL**模式的删除
- 定义基本表
- 修改基本表
- 删除基本表
- 建立索引
- 删除索引

SQL的数据定义功能

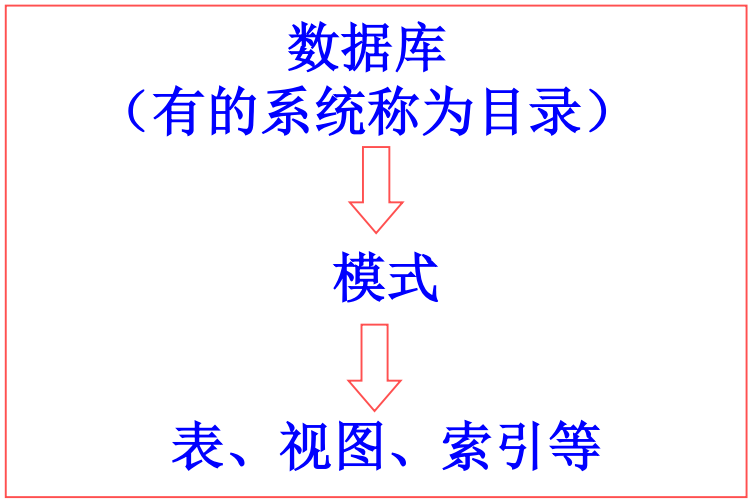
- 定义表、定义视图和定义索引
- 在SQL2中还增加了对SQL数据库模式的定义。

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	

SQL模式

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



SQL模式的定义

- 一个**SQL模式(SQL Schema)**由模式名、权限标识符和模式中元素的描述符组成。
  - 权限标识符指明拥有该模式的用户或帐号
  - 模式元素包含一个数据库应用的表、视图和索引等
- 属于同一应用的表、视图和索引等可以定义在同一模式中。
- 定义模式后，实际上定义了一个命名空间，可以进一步定义该模式包含的数据库对象，如表、视图和索引等。

- 在定义模式时可先给出模式名和权限标识符，以后再定义其中的元素，语法格式：

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

- 可以在创建模式的同时在模式定义中进一步创建基本表、视图、定义授权等

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】定义学生数据库模式SST，用户为SDBA。

```
CREATE SCHEMA SST
        AUTHORIZATION SDBA;
CREATE SCHEMA
        AUTHORIZATION WANG ;
```

没有指定“模式名”，则默认为用户名WANG

【例】创建模式时定义模式元素

```
CREATE SCHEMA AUTHORIZATION ross
  CREATE TABLE t1 (c1 INT PRIMARY KEY,
                   c2 INT REFERENCES t2(c1))
  CREATE TABLE t2 (c1 INT PRIMARY KEY,
                   c2 INT REFERENCES t1(c1));
```

SQL模式的删除

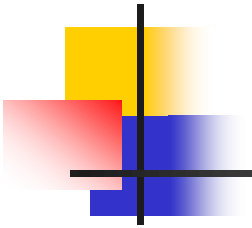
- 删除模式语句：

  DROP SCHEMA〈模式名〉［CASCADE｜RESTRICT］

  - **CASCADE (级联式)方式**
    在删除模式的同时把该模式所属的基本表、视图和索引等元素全部一起删除。
  - **RESTRICT (限制式)方式**
    只有当模式中没有任何元素时，才能删除该模式，否则拒绝该删除操作。

关于模式的补充

- 不同的系统对于**Schema**的定义和使用有所不同
- 在**SQL Server2005**之后的版本中，创建数据库时会包含一些默认的**Schema**：**dbo**，**guest**，**sys**，**INFORMATION_SCHEMA**，另外有一些角色**Schema**等
- 创建数据库对象（如**table**），如果没有指定**Schema**，则：
  （**1**）创建在当前登录用户默认的**Schema**上；
  （**2**）若没有默认的**Schema**，则创建在**dbo Schema**上；
  （**3**）如果指定了**Schema**，则按照指定的做。



- **Schema的查找顺序**
  - 假设有个登录用户**Sue**，默认**Schema**为**Sue**，现需查找使用某个表**mytable**，那么系统查找该表的顺序是：
    （1）sys.mytable（sys Schema）
    （2）Sue.mytable (Default Schema)
    （3）dbo.mytalbe (dbo Schema)
- 系统默认的**Schema**不能删除

定义基本表

CREATE TABLE <表名>

(<列名> <数据类型>[<列级完整性约束条件>]
[, <列名> <数据类型>[<列级完整性约束条件>]]
…
[,<表级完整性约束条件>]);

- **<表名>**：所要定义的基本表的名字
- **<列名>**：组成该表的各个属性（列）
- **<列级完整性约束条件>**：涉及相应属性列的完整性约束条件
- **<表级完整性约束条件>**：涉及一个或多个属性列的完整性约束条件

【例】建立学生表**Student**，表中属性有：学号**Sno**，姓名**Sname**，年龄**Sage**，性别**Ssex**，学生所在系**Sdept**。

```
CREATE TABLE Student
  (   Sno CHAR(6) NOT NULL UNIQUE,
      Sname CHAR(8),
      Sage INT,
      Ssex CHAR(2),
      Sdept CHAR(12),
      CONSTRAINT C1 CHECK (Ssex IN('男','女')),
      CONSTRAINT S_PK PRIMARY KEY (Sno));
```

- **CONSTRAINT子句定义列级或表级约束，格式为**
  CONSTRAINT <约束名> <约束>
- **SQL常用的数据类型**

SMALLINT 短整数
INTEGER或INT 长整数
REAL 浮点数
DOUBLE PRECISION 双精度浮点数
FLOAT（n） 浮点数, 精度为n位
NUMBER（p[,q]） 定点数，共p位，其中小数点后有q位
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
BIT（n） 长度为n的二进制位串
BIT VARCHAR（n） 最大长度为n的二进制位串
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

- 在**SQL2**中增加了定义域的语句，可以用域名代替指定列的数据类型。
- 如果有一个或多个表的属性的域是相同的，通过对域的修改可以很容易地改变属性的数据类型。
- 域定义语句的格式为：

  CREATE DOMAIN <域名> <数据类型>;

  - 例

    CREATE DOMAIN Sdept_TYPE CHAR(12);

    域**Sdept_TYPE**创建后，定义学生表时，对列**Sdept**的类型定义可以用域名代替：**Sdept Sdept_TYPE**。

【例】P64

```
CREATE TABLE Course
    ( Cno CHAR(6) NOT NULL,
      Cname CHAR(20),
      Ccredit INT,
      CONSTRAINT C_PK PRIMARY KEY (Cno));
CREATE TABLE SC
    ( Sno CHAR(6) NOT NULL,
      Cno CHAR(6) NOT NULL,
      Grade INT CHECK (Grade BETWEEN 0 AND 100),
      CONSTRAINT SC_PK PRIMARY KEY(Sno, Cno),
      CONSTRAINT SC_FK1 FOREIGN KEY (Sno) REFERENCES
          Student(Sno),
      CONSTRAINT SC_FK2 FOREIGN KEY (Cno) REFERENCES
          Course(Cno));
```

```
CREATE TABLE Course
    ( Cno CHAR(6) NOT NULL,
      Cname CHAR(20),
      Ccredit INT,
      PRIMARY KEY (Cno));

CREATE TABLE SC
    ( Sno CHAR(6) NOT NULL,
      Cno CHAR(6) NOT NULL,
      Grade INT CHECK (Grade BETWEEN 0 AND 100),
      PRIMARY KEY(Sno,Cno),
      FOREIGN KEY (Sno) REFERENCES  Student(Sno),
      FOREIGN KEY (Cno) REFERENCES  Course(Cno));
```

- 在定义基本表时，表所属的数据库模式一般被隐式指定，也可以显式地在定义表时指定表所属的数据库模式名。
  - 如下语句在定义学生表时，同时指出学生表所在的模式为学生数据库模式**SST**。

    CREATE TABLE SST.Student

    (Sno CHAR(6) NOT NULL UNIQUE,…)；
- 在创建模式语句中同时创建表

- 常用完整性约束
  - 主键约束：　**PRIMARY KEY**
  - 唯一性约束：**UNIQUE**
  - 非空值约束：**NOT NULL**
  - 参照完整性约束

修改基本表

ALTER TABLE <表名>
[ADD <列名> <数据类型> [<完整性约束>]]
[DROP <列名> [CASCADE | RESTRICT]]
[ALTER <列名> <数据类型>];

- **ADD子句**用于增加新列，包括列名、数据类型和列级完整性约束
- **DROP子句**用于删除指定的列名，
  - **CASCADE**表示删除列时自动删除引用该列的视图和约束
  - **RECTRICT**表示没有视图和约束引用时才能删除该列，否则将拒绝删除操作
- **ALTER子句**用于修改列的定义，如修改列的数据类型或修改列的宽度等

【例】在学生表**Student**中增加一列，列名为班级。

```
ALTER TABLE Student
    ADD Class CHAR(8);
```

- 不论基本表中原来是否已有数据，新增加的列一律为空值；不能在其上指定**NOT NULL**

【例】修改学生表**Student**中姓名列的长度为**20**。

```
ALTER TABLE Student
    ALTER Sname CHAR(20) ;
```

- 修改原有的列定义有可能会破坏已有数据

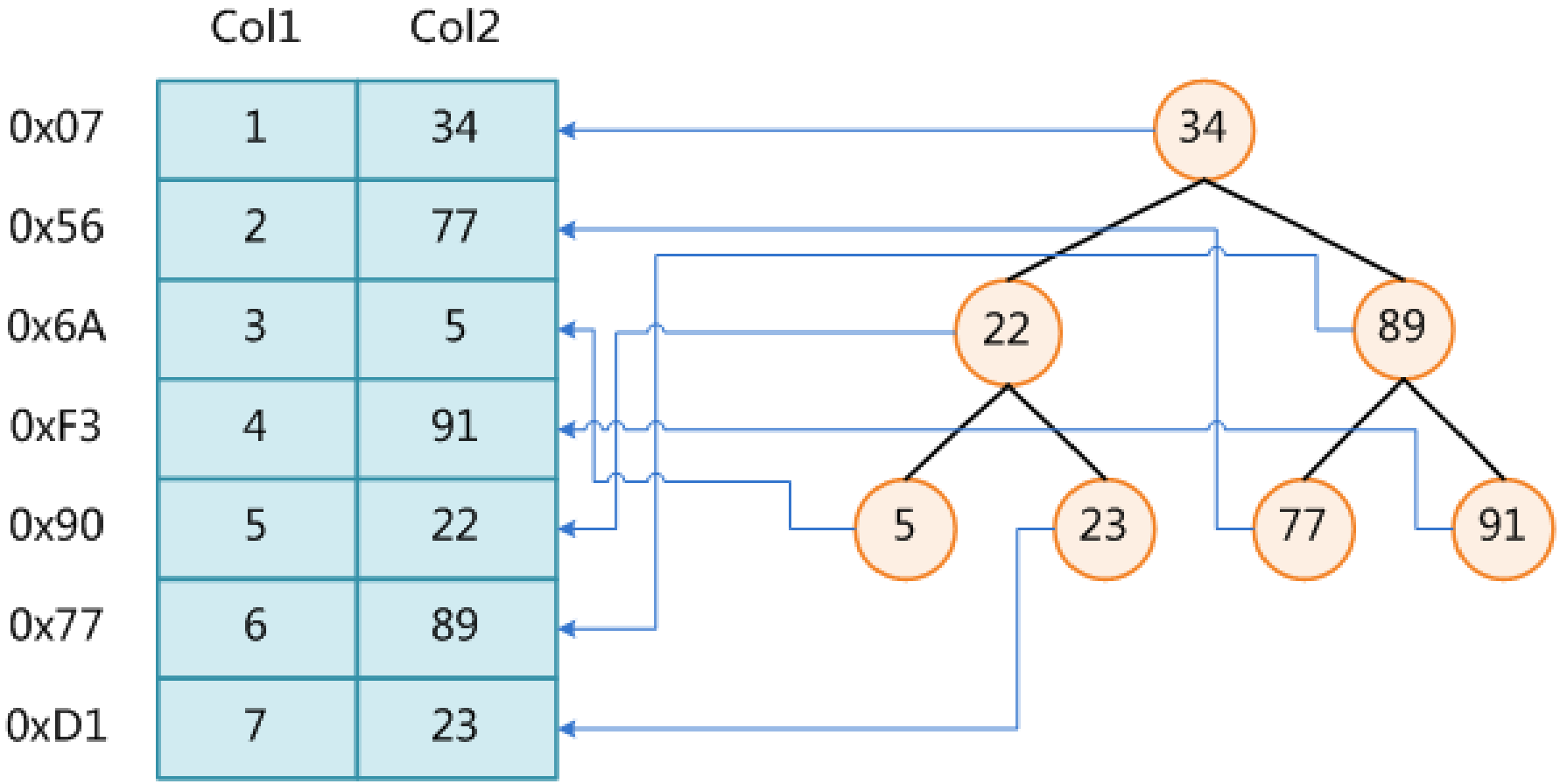
删除基本表

DROP TABLE <表名> [RESTRICT|CASCADE];

- 若选择**RESTRICT**，则删除的基本表不能被其他表的约束所引用（如**CHECK**，**FOREIGN KEY**等约束），不能有视图，不能有触发器，不能有存储过程或函数等。如果存在这些依赖该表的对象，则此表不能被删除。
- 若选择**CASCADE**，则该表的删除没有限制条件。在删除基本表的同时，相关的依赖对象，例如视图等都将被一起删除。
- 一般在缺省情况下默认为**RESTRICT**，与具体实现有关。

建立索引

- 索引是一种数据结构
  - 索引技术是数据库管理系统的核心问题之一
  - 在表上建立索引，可以提供不同的存取路径，可以加快查询速度
  - 可根据需要在一个表上建立一个或多个索引
  - **DBA**或表的创建者有权建立和删除索引
  - 索引的更新和维护是由**DBMS**自动完成的
  - 系统在存取数据时会自动选择是否使用索引，或者是以合适的索引作为存取路径，用户不必也不能选择索引
  - 有些**DBMS**能自动在某些特殊属性列上建立索引
    - **PRIMARY KEY**
    - **UNIQUE**


Col1
Col2
0x07
1
34
0x56
2
77
0x6A
3
5
0xF3
4
91
0x90
5
22
0x77
6
89
0xD1
7
23
34
22
89
5
23
77
91

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

- **<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
- **<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集索引（**Cluster Index**）

- 【例】在学生表**Student**的学号列上按升序建立惟一索引。

```
CREATE UNIQUE INDEX S_SNO
        ON Student (Sno);
```

- 【例】在表**Student**上按班级降序、年龄升序建立索引。

```
CREATE INDEX SCLASS_AGE
        ON Student (Class DESC, Sage ASC);
```

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

```
CREATE UNIQUE INDEX  SnoIdx ON Student(Sno);

CREATE UNIQUE INDEX CnoIdx ON Course(Cno);

CREATE UNIQUE INDEX SCno ON
                        SC(Sno ASC, Cno DESC);
```

- 聚集索引（Cluster Index）
  - 索引次序与基本表中元组的物理次序一致的索引；建立聚集索引后，基表中数据也需要按指定的聚集属性值的升序或降序存放，即：聚集索引的索引项顺序与表中记录的物理顺序一致
  - 例：

    CREATE CLUSTER INDEX Stusname
    ON Student(Sname)；

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

- 对某些类型的查询，聚集索引可以提高查询效率
  - 聚集索引对于那些经常要搜索范围值的列特别有效
    使用聚集索引找到包含第一个值的行后，便可以确保包含后续索引值的行是物理相邻的。
  - 如果对从表中检索的数据进行排序时经常要用到某一列，则可以将该表在该列上聚集(物理排序)，避免每次查询该列时都进行排序。
- 在一个基本表上最多只能建立一个聚集索引
- 聚集索引可以包含多个列（组合索引）
- 聚集索引的适用范围
  - 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

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：
    删除学生表上建立的**SSNO**索引。

    DROP INDEX S_SNO;

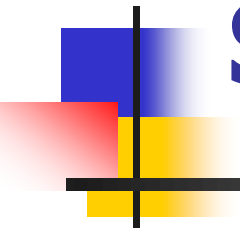
索引的选择

- 索引为性能所带来的好处是有代价的。
  - 对某个属性建立索引，能极大提高对该属性上的值的检索效率；在使用该属性进行连接操作时，还可以加快连接速度。
  - 带索引的表在数据库中会占据更多的空间。
  - 索引的维护需要一些额外的计算代价。对数据进行插入、删除和更新操作时，所花费的时间会更长。
- 应根据数据的性质和基于表的查询的性质，来决定是否创建索引，应确保对性能的提高程度大于在存储空间和处理资源方面的额外开销。

关于索引的问题

- 建立索引的目的？
- 谁可以建立索引？
- 谁维护索引？
- 怎么用索引？

索引是关系数据库的内部实现技术，属于内模式的范畴.



SQL数据查询

第4章 关系数据库标准语言SQL

- 4.1SQL简介
- 4.2SQL的系统结构
- 4.3SQL的数据定义
- 4.4SQL的数据操纵
  - 4.4.1 数据查询
  - 4.4.2 数据更新
- 4.5SQL中的视图
- 4.6SQL的数据控制
- 4.7嵌入式SQL
- 4.8小结

主要内容

- 1.查询语句的一般格式
- 2.单表查询
- 3.连接查询
- 4.嵌套查询
- 5.集合查询
- 6.基于派生表的查询

1.查询语句一般格式

SELECT [ALL|DISTINCT] <目标列表达式> [,<目标列表达式>] …

FROM <表名或视图名>[, <表名或视图名>] …

[WHERE <条件表达式1>]

[GROUP BY <列名1> [HAVING <条件表达式2>]]

[ORDER BY <列名2> [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。通常会在每组中使用聚集函数。
- 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
- DISTINCT表示去掉重复元组，ALL则容许重复元组

查询语句的基本结构

- **Select** A1, A2, ..., An
  From R1, R2, ..., Rm
  Where P

$$\Pi_{A1, A2, ..., An}(\sigma_p(R_1 \times R_2 \times ... \times R_m))$$

Select　Where　From

示例数据库：学生—课程数据库

- 学生表：

  Student(Sno，Sname，Sage，Ssex，Sdept)

- 课程表：

  Course(Cno，Cname，Ccredit，Cpno)

- 学生选课表：

  SC(Sno，Cno，Grade)

2.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
  - 使用聚集函数
  - 对查询结果分组

单表查询示例

- 【例】查询学生的学号和姓名。

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

- 【例】查询计算机系学生的学号和姓名。

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

- 【例】查询年龄在18到25岁之间的学生信息。

```
SELECT *
   FROM Student
   WHERE Sage BETWEEN 18 AND 25;
```

- 【例】查全体学生的姓名及其出生年份

SELECT Sname, 2000-Sage

FROM Student;

输出结果：

Sname	2000-Sage
李勇	1976
刘晨	1977
王名	1978
张立	1978

- 【例】查询已经选修了课程的学生学号，并按学号升序排列。

```
SELECT DISTINCT Sno
    FROM SC
    ORDER BY Sno;
```

- 【例】查询每门课的选修人数。

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

- 【例】查询平均成绩在85分以上的学生的学号和平均成绩

```
SELECT Sno, AVG(Grade)
    FROM SC
    GROUP BY Sno
    HAVING AVG(Grade)>85;
```

- 【例】查询成绩在75～85分之间的学生的学号和成绩

```
SELECT Sno, Grade
    FROM SC
    WHERE Grade>=75 AND Grade<=85;
```

- 【例】查询年龄为19岁的所有姓李的学生姓名。

```
SELECT Sname
    FROM Student
    WHERE Sname LIKE '李%' AND Sage=19;
```

- 【例】查询缺考学生的学号和课程号。

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

查询经过计算的值

- **SELECT**子句的<目标列表达式>不仅可以是表中的属性列，也可以是表达式
  - 算术表达式
  - 字符串常量
  - 函数
  - 列别名等

- 例：查询全体学生的姓名、出生年份和所在系，要求用小写字母表示所在系名。

```
SELECT Sname, 'Year of Birth:', 2000-Sage,
    LOWER(Sdept)  FROM Student;
```

输出结果：

Sname	'Year of Birth:'	2000-Sage	ISLOWER(Sdept)
李勇	Year of Birth:	1976	cs
刘晨	Year of Birth:	1977	is
王名	Year of Birth:	1978	ma
张立	Year of Birth:	1977	is

使用列别名改变查询结果的列标题

```
SELECT Sname AS NAME,
       'Year of Birth: '  BIRTH,
        2000-Sage BIRTHDAY,
        LOWER(Sdept) DEPARTMENT
 FROM Student;
```

输出结果：

NAME	BIRTH	BIRTHDAY	DEPARTMENT
李勇	Year of Birth: 1976		cs
刘晨	Year of Birth: 1977		is
王名	Year of Birth: 1978		ma
张立	Year of Birth: 1977		is

使用DISTINCT短语消除取值重复的行

假设SC表中有下列数据

Sno	Cno	Grade
95001	1	92
95001	2	85
95001	3	88
95002	2	90
95002	3	80

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

结果：

Sno
95001
95002

```
SELECT Sno  FROM SC;
```

或

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

结果：

Sno
95001
95001
95001
95002
95002

- 注意 **DISTINCT**短语的作用范围是所有目标列

  错误的写法

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

  正确的写法

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

WHERE子句常用的查询条件

比较表达式	<列名1> 比较算符 <列名2（或常量）> 比较算符：=、>、>=、<、<=、<>（或!=）
逻辑表达式	<条件表达式1> 逻辑算符 <条件表达式2> 逻辑算符：AND、OR、NOT
BETWEEN	<列名1> （NOT）BETWEEN <常量1或列名2> AND <常量2或列名3>
IN	<列名>（NOT）IN （常量表列 或 SELECT语句）
LIKE	<列名>（NOT）LIKE ‘匹配字符串’ 匹配符：“_”表示匹配一个字符，“%”表示匹配任意字符串
NULL	<列名> IS（NOT） NULL
EXISTS	（NOT）EXISTS （SELECT语句）

比较大小

- 在**WHERE**子句的**<**比较条件**>**中使用比较运算符
  - **=，>，<，>=，<=， <>**或**!= ，!>，!<**
  - **NOT ＋** 比较运算符
- 【例】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;

SELECT Sname, Sage FROM  Student
WHERE NOT Sage >= 20;
```

确定范围

- 使用谓词 **BETWEEN … AND …**

  NOT BETWEEN … AND …

- 【例】查询年龄在**20~23**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】查询年龄不在**20~23**岁之间的学生姓名、系别和年龄

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

确定集合

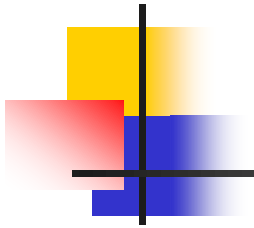
- 使用谓词　**IN <值表>，NOT IN <值表>**
  - **<值表>**：用逗号分隔的一组取值
- 【例】查询信息系（**IS**）、数学系（**MA**）和计算机科学系（**CS**）学生的姓名和性别。

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

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

字符串匹配

- **[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
    - 当匹配模板为固定字符串时，可用 **=** 运算符取代 **LIKE** 谓词，用 **!=** 或 **<>**运算符取代 **NOT LIKE**。
  - 通配符
    - **%** 代表任意长度（长度可以为**0**）的字符串
    - **_ (**下横线**)** 代表任意单个字符
    - 例：
      a%b表示以**a**开头，以**b**结尾的任意长度的字符串；
      a_b表示以**a**开头，以**b**结尾的长度为**3**的任意字符串。



- **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。

字符串匹配示例

- 匹配模板为固定字符串

```
SELECT *
    FROM  Student
    WHERE  Sno LIKE '95001';
```

等价于：

```
SELECT  *
    FROM  Student
    WHERE Sno = '95001'
```

- 匹配模板为含通配符的字符串

【例】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】查询姓“欧阳”且全名为三个汉字的学生姓名。

```
SELECT Sname
  FROM   Student
  WHERE  Sname LIKE ‘欧阳_ _’；
```

- 使用换码字符将通配符转义为普通字符

【例】查询DB_Design课程的课程号和学分。

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

【例】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 '\';
```

涉及空值的查询

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

【例】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。

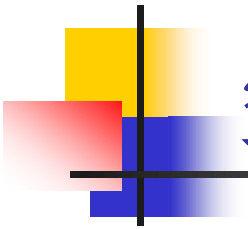
```
SELECT Sno，Cno  FROM SC
    WHERE Grade IS NULL;
```

【例】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
```

空值

- **SQL允许属性有一个特殊值NULL称作空值。**
  - 未知值：有值但是不知道是什么，例如未知生日
  - 不适用的值：例如配偶的名字
  - 保留的值：无权知道的值，例未公布的电话号码
- 空值的运算
  - 空值不同于空白或零值。没有两个相等的空值。空值和任何值进行算术运算，结果仍为空值。
    - 执行计算时消除空值很重要，因为包含空值列的某些计算（如平均值）会不准确。
  - 当使用逻辑运算符和比较运算符，有可能返回结果 **UNKNOWN**， 是与**TRUE** 和 **FALSE** 相同的布尔值



空串

- 空串是指长度为零的字符串
  - 当 **m** 为**0**或负数时，**RIGHT('123', m)** 返回空字符串 **RTRIM('　　')** 返回空字符串。

多重条件查询

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

【例】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;
```

对查询结果排序

- 使用**ORDER BY**子句
  - 可以按一个或多个属性列排序
  - 升序：**ASC**；降序：**DESC**；缺省值为升序
- 空值将作为最大值排序
  - **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】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;
```

查询结果

Sno	Grade
95010	
95024	
95007	92
95003	82
95010	82
95009	75
95014	61
95002	55

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

使用聚集函数

- **SQL提供了许多聚集函数，用来实现统计查询**
  - 计数
    COUNT([DISTINCT|ALL] *)
    COUNT([DISTINCT|ALL] <列名>)
  - 计算总和　　**SUM ([DISTINCT|ALL] <列名>)**
  - 计算平均值　**AVG ([DISTINCT|ALL] <列名>)**
  - 求最大值　　**MAX ([DISTINCT|ALL] <列名>)**
  - 求最小值　　**MIN ([DISTINCT|ALL] <列名>)**
- **选项DISTINCT表示在计算时要取消指定列中的重复值；ALL表示不取消重复值；默认为ALL。**

【例】查询学生总人数。

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

【例】查询选修了课程的学生人数。

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

【例】计算1号课程的学生平均成绩。

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

【例】查询选修1号课程的学生最高分数。

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

对查询结果分组

- 使用**GROUP BY**子句分组
- 细化聚集函数的作用对象
  - 未对查询结果分组，聚集函数将作用于整个查询结果
  - 对查询结果分组后，聚集函数将分别作用于每个组
- 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聚集函数
- **GROUP BY**子句的作用对象是查询的中间结果表
- 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

【例】求各个课程号及相应的选课人数。

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

【例】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；
```

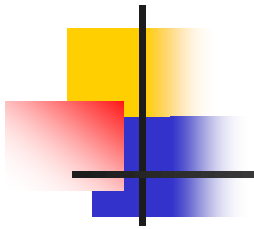
【例】查询平均成绩大于等于90分的学生学号和平均成绩

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

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

【例】查询有3门以上课程是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;
```



- **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**(tuple)**。
    - **HAVING**短语作用于组，从中选择满足条件的组**(group)**
  - **WHERE**子句中不能使用聚集函数；而**HAVING**短语中可以使用聚集函数

- 分组查询示例

```
SELECT productid,SUM(quantity)
      AS total_quantity
 FROM orderhist
 GROUP BY productid
```

productid	*orderid*	*quantity*
1	1	5
1	1	10
2	1	10
2	2	25
3	1	15
3	2	30

productid	*total_quantity*
1	15
2	35
3	45

```
SELECT productid,SUM(quantity)
      AS total_quantity
 FROM orderhist
 WHERE productid = 2
 GROUP BY productid
```

productid	orderid	quantity
1	1	5
1	1	10
2	1	10
2	2	25
3	1	15
3	2	30

productid	total_quantity
2	35

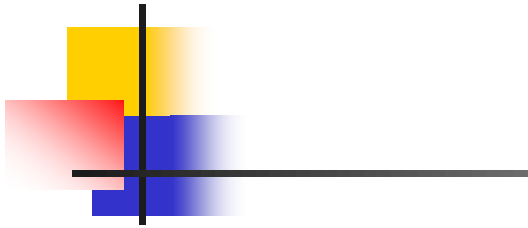
```
SELECT productid,
       SUM(quantity)
    AS total_quantity
 FROM orderhist
 GROUP BY productid
 HAVING SUM(quantity)>=30
```

productid	orderid	quantity
1	1	5
1	1	10
2	1	10
2	2	25
3	1	15
3	2	30

productid	total_quantity
2	35
3	45

3. 连接查询

- 同时涉及两个或两个以上表的查询称为连接查询
  - 用来连接两个表的条件称为连接条件或连接谓词
  - 连接谓词中的列名称为连接字段
  - 连接条件中各连接字段的类型必须是可比的，但不必是相同的
- 连接条件的一般格式
  - [<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>
    其中比较运算符为：=、>、<、>=、<=、!=



- **SQL中连接查询的主要类型**
  - 广义笛卡尔积
  - 等值(含自然连接)
  - 非等值连接查询
  - 自身连接查询
  - 外连接查询
  - 复合条件连接查询

广义笛卡尔积

- 不带连接谓词的连接（即没有**WHERE**子句）。
- 广义笛卡尔积是两表元组的交叉乘积，其连接的结果会产生没有意义的元组，实际上很少使用。

【例】

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

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省略表名。

【例】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;
```

SNO	Sname	Ssex	Sage	Sdept
95001	李勇	男	20	CS
95002	刘晨	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS

Student表

Sno	Cno	Grade
95001	C1	85
95001	C2	70
95001	C3	78
95002	C2	75
95002	C3	90

SC表

Student.Sno	Sname	Ssex	Sage	Sdept	SC.Sno	Cno	Grade
95001	李勇	男	20	CS	95001	C1	92
95001	李勇	男	20	CS	95001	C2	85
95001	李勇	男	20	CS	95001	C3	88
95002	刘晨	女	19	IS	95002	C2	90
95002	刘晨	女	19	IS	95002	C3	80

结果表

自然连接

- 等值连接的一种特殊情况，把目标列中重复属性去掉

【例】

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

SELECT  *
   FROM     Student NATURAL JOIN SC；
```

Sno	Sname	Ssex	Sage	Sdept
95001	李勇	男	20	CS
95002	刘晨	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS

Student表

Sno	Cno	Grade
95001	C1	85
95001	C2	70
95001	C3	78
95002	C2	75
95002	C3	90

SC表

Student.Sno	Sname	Ssex	Sage	Sdept	Cno	Grade
95001	李勇	男	20	CS	C1	92
95001	李勇	男	20	CS	C2	85
95001	李勇	男	20	CS	C3	88
95002	刘晨	女	19	IS	C2	90
95002	刘晨	女	19	IS	C3	80

结果表

非等值连接查询

- 连接运算符不是 = 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：>、<、>=、<=、!=

自身连接

- 一个表与其自己进行连接，称为表的自身连接
- 需要给表起别名以示区别
- 由于所有属性名都是同名属性，因此必须使用别名前缀

【例】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 AS FIRST，Course AS SECOND
        WHERE FIRST.Cpno = SECOND.Cno；
```

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

FIRST表（Course表）

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

WHERE FIRST.Cpno = SECOND.Cno

FIRST. Cno	FIRST.Cpno	SECOND.Cno	SECOND.Cpno
1	5	5	7
3	1	1	5
4	6	6	
5	7	7	6
7	6	6	

SELECT FIRST.Cno，SECOND.Cpno

	Cno	Cpno
▶	1	7
	3	5
	4	
	5	6
	7	

内连接

- 典型的连接运算，使用像 = 或 <> 之类的比较运算符）
- 包括相等连接和自然连接
- 内连接使用比较运算符根据每个表共有的列的值匹配两个表中的行
- 【例】

```
SELECT buyer_name, sales.buyer_id, qty
    FROM buyers INNER JOIN sales
        ON buyers.buyer_id = sales.buyer_id
```

```
SELECT buyer_name, sales.buyer_id, qty
    FROM buyers INNER JOIN sales
      ON buyers.buyer_id = sales.buyer_id
```

buyers

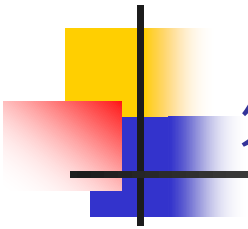
buyer_name	buyer_id
Adam Barr	1
Sean Chai	2
Eva Corets	3
Erin O'Melia	4

sales

buyer_id	prod_id	qty
1	2	15
1	3	5
4	1	37
3	5	11
4	2	1003

Result

buyer_name	buyer_id	qty
Adam Barr	1	15
Adam Barr	1	5
Erin O'Melia	4	37
Eva Corets	3	11
Erin O'Melia	4	1003



外连接

- 外连接的概念：**3.2.3**
- 外连接与普通连接的区别
  - 普通连接操作只输出满足连接条件的元组
  - 外连接操作以指定表为连接主体，将主体表中不满足连接条件的元组一并输出
- 外连接分类
  - 左外连接(**LEFT OUTER JOIN**)
  - 右外连接(**RIGHT OUTER JOIN**)
  - 全外连接(**FULL OUTER JOIN**)
  - **SQL2**支持

【例】查询每个学生的选课情况，包括没有选课的学生

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

Sno	Sname	Ssex	Sage	Sdept
95001	李勇	男	20	CS
95002	刘晨	女	19	IS
95003	王敏	女	18	MA
95004	张立	男	19	IS

Student表

Sno	Cno	Grade
95001	C1	85
95001	C2	70
95001	C3	78
95002	C2	75
95002	C3	90

SC表

Student.Sno	Sname	Ssex	Sage	Sdept	Cno	Grade
95001	李勇	男	20	CS	C1	92
95001	李勇	男	20	CS	C2	85
95001	李勇	男	20	CS	C3	88
95002	刘晨	女	19	IS	C2	90
95002	刘晨	女	19	IS	C3	80
95003	王敏	女	18	MA		
95004	张立	男	19	IS		

结果表

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。
  - 复合条件连接可以看作（普通）连接后得到的关系（表）又进行一次选择运算

【例】查询选修**2**号课程且成绩在**90**分以上的所有学生的 学号、姓名

```
SELECT Student.Sno, Student.Sname
    FROM   Student, SC
    WHERE Student.Sno = SC.Sno AND
          SC.Cno= '2' AND
          SC.Grade > 90;
```

多表连接

- 连接操作涉及到两个以上的表的连接

【例】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
95001	李勇	数据库	92
95001	李勇	数学	85
95001	李勇	信息系统	88
95002	刘晨	数学	90
95002	刘晨	信息系统	80

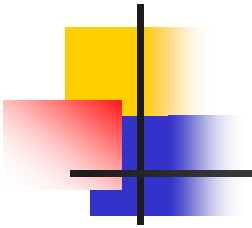
4. 嵌套查询

- 嵌套查询概述
- 嵌套查询分类与求解方法
- 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

嵌套查询概述

- 一个**SELECT-FROM-WHERE**语句称为一个查询块
- 将一个查询块嵌套在另一个查询块的**WHERE**子句或**HAVING**短语的条件中的查询称为嵌套查询
- 【例】

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



- 嵌套查询的实现
  一般是从里到外，先进行子查询，再把其结果用于父查询作为条件
- 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
- 子查询的限制
  - 不能使用**ORDER BY**子句

嵌套查询的分类与求解方法

- 不相关子查询
  - 子查询的查询条件不依赖于父查询
  - 由里向外逐层处理。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。
- 相关子查询
  - 子查询的查询条件依赖于父查询
  - 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。

- 【例】找出每个学生所选修课程成绩超过该门课程平均成绩的课程号。

```
SELECT Sno, Cno /*外层查询/父查询*/
    FROM  SC  x
    WHERE Grade >=(SELECT AVG(Grade)
                   FROM  SC y
                   WHERE y.Cno=x.Cno);
                   /*内层查询/子查询*/
```

带有IN谓词的子查询

【例】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
95001	刘晨	IS
95004	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' );
```

- 父查询和子查询中的表均可以使用别名

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

- 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'
```

【例】查询选修了“信息系统”课程的学生的学号和姓名

```
SELECT Sno, Sname
    FROM    Student
    WHERE Sno  IN
      (SELECT Sno
          FROM    SC
          WHERE  Cno IN
              (SELECT Cno
                  FROM Course
                  WHERE Cname= ‘信息系统’ ));
```

- 用连接查询

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

带有比较运算符的子查询

- 当能确切知道内层查询返回单值时，可用比较运算符（>，<，=，>=，<=，!=或< >）
- 【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，则在左例可以用 = 代替IN.

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

```
SELECT Sno, Sname, Sdept
FROM Student S1
WHERE S1.Sdept  =
    ( SELECT Sdept S2
      FROM Student
      WHERE S2.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;
```

带有ANY或ALL谓词的子查询

- 谓词语义
  - **ANY** 任意一个值
  - **ALL** 所有值
- 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

> 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

- **ANY和ALL谓词有时可以用聚集函数来实现**
  - 用聚集函数实现子查询通常比直接用**ANY**或**ALL**查询效率要高，因为前者通常能够减少比较次数。
  - **ANY**与**ALL**与聚集函数的对应关系如下：

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

=any : in

【例】查询其他系中比CS系任意一个学生年龄小的学生姓名和年龄

- 用ANY谓词实现

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

- 用聚集函数实现

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

【例】查询其他系中比CS系所有学生年龄都小的学生姓名及年龄。

- 用ALL谓词实现

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

- 用聚集函数实现

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

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. 用EXISTS/NOT EXISTS实现全称量词
5. 用EXISTS/NOT EXISTS实现逻辑蕴函

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】查询所有选修了1号课程的学生姓名。

- 思路分析：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,且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 ');
```

用连接运算

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

【例】查询没有选修1号课程的学生姓名。

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

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。
- 带有**EXISTS**谓词的相关子查询只关心内层查询是否有返回值,不需要查具体值,效率不低于相关子查询

【例】查询与“刘晨”在同一个系学习的学生。可以用带EXISTS谓词的子查询替换：

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

```
SELECT Sno, Sname, Sdept
     FROM Student S1
     WHERE EXISTS
          SELECT *
          FROM Student S2
          WHERE S2.Sdept=S1.Sdept
                AND S2.Sname= '刘晨';
```

4.用EXISTS/NOT EXISTS实现全称量词

- SQL语言中没有全称量词 ∀ （For all）
- 可以把带有全称量词的谓词转换为等价的带有存在量词的谓词：

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

$(\forall x)P \equiv \neg (\exists x(\neg P))$
不存在某门课程是他没选的

【例】查询选修了全部课程的学生姓名。

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

这样的课是不存在的

这门课他没选

5. 用EXISTS/NOT EXISTS实现逻辑蕴函

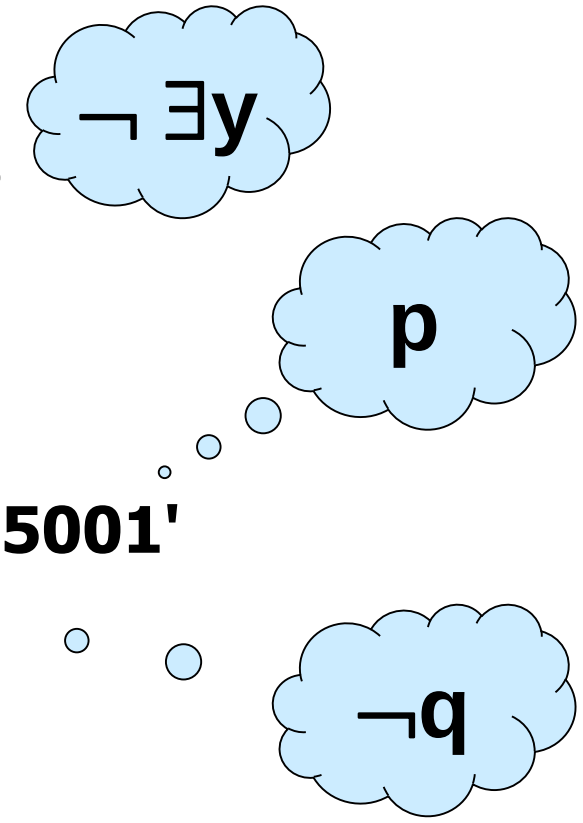
- SQL语言中没有蕴函(Implication)逻辑运算
- 可以利用谓词演算将逻辑蕴函谓词等价转换为：

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

【例】查询至少选修了学生95001选修的全部课程的学生的学号。

- 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95001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95001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：对于学生**x**，不存在这样的课程**y**，学生95001选修了**y**，而学生**x**没有选。

解: SELECT DISTINCT Sno
FROM SC SCX
WHERE NOT EXISTS
(SELECT *
FROM SC SCY
WHERE SCY.Sno = '95001'
AND NOT EXISTS
(SELECT *
FROM SC SCZ
WHERE SCZ.Sno=SCX.Sno AND
SCZ.Cno=SCY.Cno));



5.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(UNION)
  - 交(INTERSECT)
  - 差(MINUS，EXCEPT)

并操作

- 语法形式

  <查询块>

  UNION [ALL]

  <查询块>;

- 两个查询结果的属性列个数相同，对应项的数据类型必须能够通过隐式转换相互兼容。
- 使用**UNION**合并多个结果集时，系统会自动去掉重复元组。
- 使用**UNION ALL**操作符，可以保留重复元组。

【例】查询计算机系的学生或者年龄不大于19岁的学生。

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

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

【例】设数据库中有一教师表**Teacher(Tno, Tname,...)**。查询学校中所有师生的姓名。

```
SELECT Sname
   FROM  Student
UNION
SELECT  Tname
   FROM  Teacher;
```

UNION结果集中的列名与**UNION**运算中第一个**SELECT**语句的结果集中的列名相同，
其他的**SELECT**语句的结果集列名将被忽略。

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】查询计算机系的学生与年龄不大于19岁的学生的交集。

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

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

【例】查询选修课程1的学生集合与选修课程2的学生集合的交集。

本例实际上是查询既选修了课程1又选修了课程2的学生。

Select Sno From SC Where Cno='1' and Cno='2'

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

【例】查询学生姓名与教师姓名的交集。查询学校中与教师同名的学生姓名。

```
SELECT DISTINCT Sname
    FROM Student
    WHERE Sname IN
          (SELECT Tname
               FROM Teacher);
```

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】查询学生姓名与教师姓名的差集。实际上是查询学校中未与教师同名的学生姓名。

```
SELECT Sname
   FROM   Student
EXCEPT
SELECT  Tname
   FROM   Teacher;
```

```
SELECT DISTINCT Sname
     FROM Student
     WHERE Sname NOT IN
               (SELECT Tname
                    FROM Teacher);
```

【例】查询选修课程1但没有选修课程2的学生姓名

```
SELECT Sname
FROM Student
WHERE Sno IN
    ( (SELECT   Sno
       FROM    SC
       WHERE   Cno='1')
      EXCEPT
      (SELECT Sno
       FROM   SC
       WHERE  Cno='2') );
```

【例】查询选修课程1但没有选修课程2的学生学号。

```
SELECT Sno
    FROM SC
    WHERE Cno=' 1 ' AND
        Sno NOT IN (SELECT Sno
                    FROM SC
                    WHERE Cno=' 2 ');
```

对集合操作结果的排序

- 在执行集合操作时，默认按照最后结果表中第一列数据的升序方式排列记录。
- 各**SELECT**子句不能含有**ORDER BY**子句，但是可以将**ORDER BY**子句放在最后的**SELECT**语句后面，以便对最后的结果表排序。
- **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**子句中最好用数字指定排序的列属性，以免出错。

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;
```

集合查询小结

- 参与集合运算的中间结果集的属性个数必须一致，且对应属性的类型必须兼容；
- 参与运算的属性名不一定相同；
- 最终结果集采用第一个中间结果集的属性名；
- 默认自动删除结果中的重复元组；
- **Order By**子句要求放在整个语句的最后；
- 标准**SQL**中没有提供集合交、差操作，但可用其他方法间接实现。

6.基于派生表的查询

- 子查询出现在**FROM**子句中，这时子查询生成的临时派生表（**Derived Table**）成为主查询的查询对象

【例】找出每个学生超过他自己选修课程平均成绩的课程号。

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