1. SQL概述
2. SQL介绍

SQL语言原名SEQUEL，是一个通用的、功能极强的关系数据库语言。同时也是一种介于关系代数与关系演算之间的结构化查询语言，其功能包括**数据定义、数据查询、数据操纵和数据控制**

SQL已经成为**关系数据库的查询标准**

SQL也是现在和将来DBMS的标准

SQL促进了分布式数据库和客户/服务器数据库的开发

1. SQL特点

**大小写不敏感**

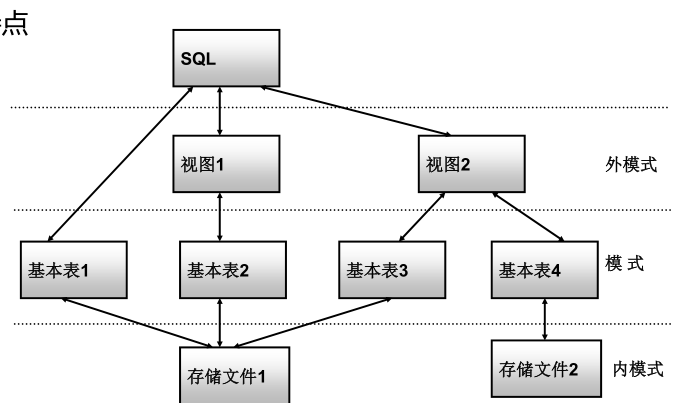
综合统一：DBMS都支持SQL语言

高度非过程化：只需要告诉做什么，不需要告诉怎么做

**面向集合**的操作方式

两种使用方式，统一的语法结构，既是独立的语言，也是嵌入语言

简洁易学



1. 基本概念
2. 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

1. 存储文件

逻辑结构组成了关系数据库的内模式

物理结构对用户是隐蔽的

1. 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

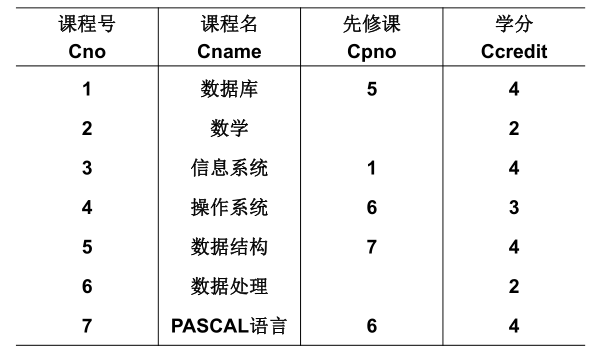
视图是一个虚表

用户可以在视图上再定义视图

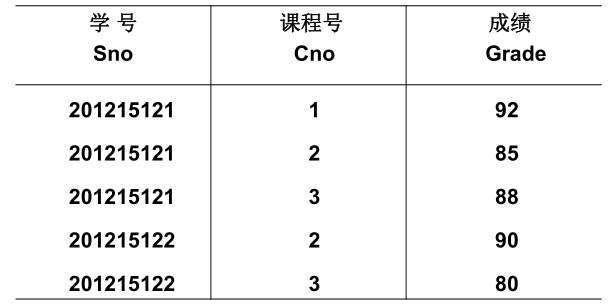
1. 学生-课程数据库
2. Student表



1. Course表



1. SC表



1. 数据定义
2. 数据定义概述

SQL的数据定义功能: 模式定义、表定义、视图和索引的定义

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**

**·一个模式下通常包括多个表、视图和索引等数据库对象**

1. 数据库的创建和修改
2. 创建数据库**CREATE DATABASE <database\_name>**

例：CREATE DATABASE student

1. 使用（切换）数据库 **use <database\_name>**

例：use student

1. 删除数据库 **drop database <database\_name>**

例：drop database student

**不能删除当前数据库**（不能自己删除自己）

1. 模式的定义和删除
2. 模式概述

模式是一个独立于数据库用户的非重复命名空间，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。**可以将模式视为数据库对象的容器**

一个数据库可以有多个模式，模式隶属于数据库  
可以不创建模式

1. 模式定义 **CREATE SCHEMA < 模式名> AUTHORIZATION < 用户名>**

如果没有指定模式名，则模式名隐含为用户名

权限：使用该命令，用户必须具有DBA权限，或获得了DBA授权CREATE SCHEMA 的权限

例：CREATE SCHEMA Test AUTHORIZATION ZHANG

1. 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

1. 表的定义、修改和删除
2. 数据类型

SQL中**域**的概念用**数据类型**来实现

定义列时需要指明其数据类型及长度，选用哪种数据类型取决于取值范围和要做哪些运算



Char(n) n字节 一个英文字符（大小写字母、符号、数字）占1个字节，一个汉字两个字节

Int 4个字节

Smallint 2个字节

字符串使用单引号’’，如果字符串本身也包括单引号，则使用两个单引号’’’’（不是双引号）

1. 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>] ] …

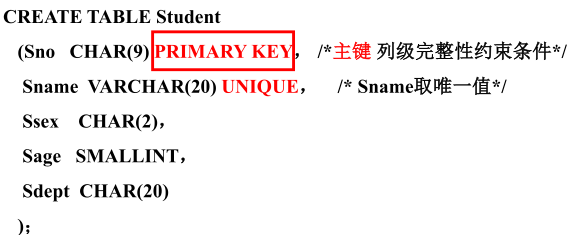
[，<表级完整性约束条件> ]

）;

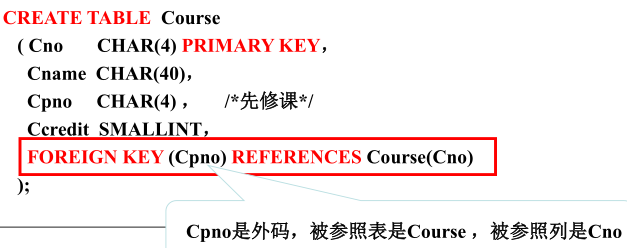
若未写模式名，则默认把架构加载公共区域dbo上。Schema模式就像创建的一个一个小房间（一个房间由一个用户管理），不建小房间就放在公共区域dbo里，但这样效率会低

若想要把student表加在Test模式上**Create table Test.student()**

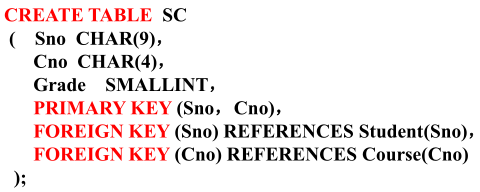
例1：建立“学生”表Student。学号是主码，姓名取值唯一



例2：建立一个“课程”表Course



例3：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



常用的完整性约束

·主码约束： PRIMARY KEY

·唯一性约束：UNIQUE

·非空值约束：NOT NULL

·参照完整性约束： FOREIGN KEY

PRIMARY KEY与 与 UNIQUE 的区别：一个表只能有一个PRIMARY KEY，但是可以有好多个UNIQUE，而且UNIQUE可以为NULL值，PRIMARY KEY不能为NULL

1. 修改基本表

**ALTER TABLE <表名>**

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

[ **ADD** <表级完整性约束>]

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

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

[ **ALTER COLUMN** <列名> <数据类型> ];

·<表名>：要修改的基本表

·ADD子句：增加新列和新的完整性约束条件

·DROP子句：删除指定的完整性约束条件

·ALTER COLUMN子句：用于修改列名和数据类型

例1：向Student表增加“入学时间”列，其数据类型为日期型

ALTER TABLE Student ADD S\_entrance DATE;

**不论基本表中原来是否已有数据，新增加的列一律为空值**

例2：将年龄的数据类型由字符型（设原来的数据类型是字符型）改为整数

ALTER TABLE Student ALTER COLUMN Sage INT;

例3：增加课程名称必须取唯一值的约束条件

ALTER TABLE Course ADD UNIQUE(Cname);

1. 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**

注意：如果选择CASCADE时可以删除表，视图也自动被删除

DROP TABLE Student CASCADE;

1. 索引的建立与删除
2. 概述

建立索引是**加快查询速度**的有效手段，索引是相对**列**的

DBA或表的属主（即建立表的人）根据需要建立索引

**DBMS自动建立PRIMARY KEY、UNIQUE列上的索引。在一个基本表上最多只能建立一个聚簇索引，系统为PRIMARY KEY建立CLUSTER聚簇索引，为UNIQUE列上建立UNIQUE的索引。这两种索引不可删，要删除主码约束才可以删除此列的聚簇索引。自己建立的索引可以随便删除。**

**更新、删除操作多的表不适合建立索引，查询多的表适合建立索引**

1. 建立索引

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

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

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

**CLUSTER**表示要建立的索引是聚簇索引

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**

**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1：为Student表按学号升序建唯一索引

CREATE UNIQUE INDEX Stusno ON Student(Sno)；

例2：为Course表按课程号升序建唯一索引

CREATE UNIQUE INDEX Coucno ON Course(Cno)；

例3：为SC表按学号升序和课程号降序建唯一索引

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

1. 修改索引名

ALTER INDEX <旧索引名> RENAME TO <新索引名>

例：将SC表的SCno索引名改为SCSno

ALTER INDEX SCno RENAME TO SCSno;

1. 删除索引名

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Student表的Stusname索引

DROP INDEX Student.Stusname

Q：常见的数据库对象有哪些？

答：表、视图、索引、序列、同义词

Q：SCHEMA和数据库对象之间关系是怎样的

Q：一般来说，建立索引可以提高查询效率，索引建得越多越好吗

答：更多的索引意味着也需要更多的空间（索引也是需要空间来存放的）

Q：哪些情况不适合给表建立索引

答：对于在查询过程中很少使用或参考的列、对于那些只有很少数据值的列、当修改性能远大于检索性能

1. 数据查询
2. 查询语句概述
3. 基本语法

**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>]

**[GROUP BY** 〈列名〉[，〈列名〉]…

**[HAVING** <内部函数表达式>] ]

**[ORDER BY** 〈列名〉 [ASC│DESC] [，〈列名〉[ASC│DESC]]…]

1. **子句功能**

**SELECT子句与FROM子句是必选子句**

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聚集函数。

HAVING短语：只有满足指定条件的**组**才予以输出

ORDER BY子句：对查询结果表按指定列值的升序或降序排序

1. 单表查询
2. 选择表中的若干列
3. 查询指定列

例：查询全体学生的学号与姓名SELECT Sno,Sname FROM Student;

1. 查询全部列

例：查询全体学生的详细记录

·在SELECT关键字后面列出所有列名

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

·将<目标列表达式>指定为 \*

SELECT \* FROM Student;

1. 查询经过计算的值

SELECT子句的<目标列表达式>不仅可以为表中的属性列，也可以是表达式

例：查全体学生的姓名及其出生年份

SELECT Sname,2020-Sage FROM Student;

1. 起别名

SELECT Sname,2020-Sage [as] 出生年份 FROM Student;

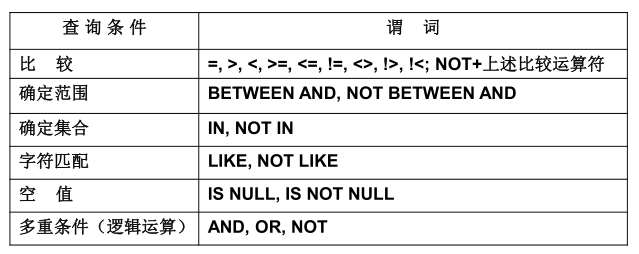
As可省，一个空格即可

1. 选择表中的若干元组
   1. DISTINCT消除取值重复的行

如果没有指定DISTINCT关键词，则缺省为ALL

如果没有where语句进行筛选元组，是不会有重复行的

* 1. 常用的查询条件



* 1. 比较

=表示相等，SQL中无双等号

例1：查询计算机科学系全体学生的名单

SELECT Sname FROM Student WHERE Sdept=‘CS’;

例2：查询所有年龄在20岁以下的学生姓名及其年龄

SELECT Sname,Sage FROM Student WHERE Sage < 20;

例3：查询考试成绩有不及格的学生的学号

SELECT **DISTINCT** Sn FROM SC WHERE Grade<60;

* 1. 确定范围 BETWEEN…AND… NOT BETWEEN…AND…

例：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;

* 1. 确定集合 **IN <值表>, NOT IN <值表>**

例：查询CS、MA和IS系学生的姓名和性别

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

* 1. 字符匹配 [NOT] **LIKE**‘<匹配串>’ [ESCAPE ‘ <换码字符>’]

<匹配串>可以是一个完整的字符串（等价=），也可以含有通配符%和 \_

**·% （百分号） 代表任意长度（长度可以为0）的字符串**

a%b表示以a开头，以b结尾的任意长度的字符串

例：查询所有不姓刘学生的姓名、学号和性别

SELECT Sname, Sno, Ssex

FROM Student

WHERE Sname NOT LIKE '刘%';

**·\_ （下横线） 代表任意单个字符**

a\_b表示以a开头，以b结尾的长度为3的任意字符串

例：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\_\_’;

**·**[ESCAPE ‘<换码字符>’]

例：查询课程名以’DB\_’开头，且倒数第三个字符为i的课程的课程号和学分

Select cno, credit

From course

Where cname like ‘DB\_\\_%i\_\_’ escape ‘\’

* 1. 涉及空值的查询 **IS NULL** 或 **IS NOT NULL**

**IS NULL 或 IS NOT NULL**空值是不确定的值，所以不可以=NULL

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

·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;

* 1. 多重条件查询

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

例：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;

1. 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

对于空值，排序时显示的次序由具体系统实现来决定

例1：查询选修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2：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;

1. 聚集函数（聚集函数对NULL不作处理）
   1. 统计元组个数 COUNT(\*)

有多少行，是数有多少个，不是有多少种，重复的也算，**只要有值就统计（空格也算）**，但**不统计空值NULL**

**“”空字符串也是字符串，而NULL是什么都没有**

COUNT(distinct 列名)得到列中不重复行的个数

例：查询学生总人数

SELECT COUNT(\*) FROM Student;

* 1. 统计**一列中**值的个数 COUNT([DISTINCT|ALL] <列名>)

例：**查询选修了课程的学生人数**

SELECT COUNT(DISTINCT Sno) FROM SC;

* 1. 计算一列值的总和（此列必须为数值型）SUM([DISTINCT|ALL] <列名>)

例：查询学生201215012选修课程的总学分数

SELECT SUM(Ccredit)

FROM SC,Course

WHERE Sno='201215012' AND SC.Cno=Course.Cno;

* 1. 计算一列值的平均值（此列必须为数值型）AVG([DISTINCT|ALL] <列名>)

例：计算1号课程的学生平均成绩

SELECT AVG(Grade)

FROM SC

WHERE Cno= ' 1 ';

* 1. 求一列中的最大值和最小值

例：查询选修1号课程的学生最高分数

SELECT MAX(Grade)

FROM SC

WHERE Cno='1';

1. GROUP BY子句
   1. GROUP BY子句分组

Group by必须和聚集函数一起使用，先分组再对每个组作用聚集函数

用于细化**聚集函数**的作用对象

**如果未对查询结果分组，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**

按指定的一列或多列值分组，值相等的为一组

例1：求各个课程号及相应的选课人数

SELECT Cno，COUNT(Sno)

FROM SC

GROUP BY Cno;

例2：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3：查询平均成绩大于等于90分的学生学号和平均成绩

SELECT **Sno**, **AVG(Grade)**

FROM SC

GROUP BY **Sno**

HAVING AVG(Grade)>=90;

注意：有聚合函数的select的语句，列名部分只能包含分组列和聚合列

* 1. 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，对**聚合结果**进行筛选

WHERE子句中是不能用聚集函数作为条件表达式

**HAVING短语作用于组**，从中选择满足条件的组

WHERE子句中要用聚集函数作为条件表达式

1. 连接查询
2. 连接查询

同时涉及多个表的查询称为连接查询

用来**连接两个表**的条件称为连接条件或连接谓词，其一般格式为：

[<表名1>.]<列名1> < 比较运算符> [<表名2>.]<列名2>

比较运算符：=、>、<、>=、<=、!=

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

1. 等值与非等值连接查询
2. 等值连接

若连接运算符为 = 时，称为等值连接

在等值连接中，去掉目标列中的重复属性则为自然连接

FROM Student，SC是笛卡尔积

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；

采用等值连接，会出现重复列

以下采用**自然连接**：

SELECT Student.Sno, Sname , Ssex , Sage , Sdept , Cno ,Grade

FROM Student , SC

WHERE **Student.Sno = SC.Sno**;

1. 非等值连接

使用其他运算符时，称为非等值连接

1. 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 FIRST，Course SECOND

WHERE FIRST.Cpno = SECOND.Cno;

1. 外连接复合条件
   1. 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

* 1. 在表名后面加外连接操作符指定主体表

非主体表有一“万能”的虚行，该行全部由空值组成

虚行可以和主体表中所有不满足连接条件的元组进行连接

由于虚行各列全部是空值，因此与虚行连接的结果中，来自非主体表的属性值全部是空值

例：查询每个学生及其选修课程的情况包括没有选修课程的学生----用外连接操作

SELECT Student.Sno,Sname,Ssex,Sage,Sdept,Cno,Grade

FROM Student **LEFT OUTER JOIN** SC

**ON** Student.Sno = SC.Sno;

外连接：列出左右两边关系中所有的元组

SELECT FIRST.Cno，SECOND.Cpno

FROM Course FIRST **FULL OUTER JOIN** Course SECOND

**ON** FIRST.Cpno = SECOND.Cno；

左连接：列出左边关系中所有的元组

SELECT FIRST.Cno，SECOND.Cpno

FROM Course FIRST **LEFT OUTER JOIN** Course SECOND

**ON** FIRST.Cpno = SECOND.Cno；

右连接：列出右边关系中所有的元组

SELECT FIRST.Cno，SECOND.Cpno

FROM Course FIRST **RIGHT OUTER JOIN** Course SECOND

**ON** FIRST.Cpno = SECOND.Cno；

* 1. 复合条件连接

WHERE子句中含多个连接条件时，称为复合条件连接

例：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.SnoAND /\* 连接谓词\*/

SC.Cno= ' 2 ' AND /\* 其他限定条件 \*/

SC.Grade > 90； /\* 其他限定条件 \*/

* 1. 多表连接

两个以上的表进行连接

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 /\* 多表连接\*/

WHERE Student.Sno = SC.Sno

and SC.Cno = Course.Cno;

1. 嵌套查询
2. 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**

**子查询不能使用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

1. 分类
   1. 不相关子查询

内层子查询可以单独执行，子查询的查询条件不依赖于父查询

先查子查询，再执行外部的查询

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

* 1. 相关子查询

子查询的查询条件依赖于父查询

由父查询拿着父表的一个一个元组循环的去子查询中找

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

1. 带有**IN**谓词的子查询

例1：查询与“刘晨”在同一个系学习的学生

SELECT Sno，Sname，Sdept

FROM Student

WHERE **Sdep**t **IN**

*(SELECT* ***Sdept***

*FROM Student*

*WHERE Sname=’刘晨’)；*

例2：查询选修了课程名为“信息系统”的学生学号和姓名。

SELECT Sno，Sname

FROM Student

WHERE Sno IN

(SELECT Sno

FROM SC

WHERE Cno IN

(SELECT Cno

FROM Course

WHERE Cname= ‘信息系统’));

1. 拓展例子

查询选修课程号为2的所有学生的姓名

Select sname

From student

Where sno in (

Select sno

From sc

Where cno=’2’)

可以将查询结果当做一张表进行连接查询

Select sname

From student, (select sno from sc where cno=’2’) temp//起别名

Where student.sno = temp.sno

1. 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< >）父查询参与子查询，父查询拿出每一行去子查询中试探

与ANY或ALL谓词配合使用

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);

1. 带有ANY或ALL谓词的子查询

ANY：任意一个值

ALL：所有值

例：> ANY 大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

注意：>ANY( )与<(MAX( ))不同，ANY考虑空值，MAX（聚集函数）不考虑空值

例：查询其他系中比信息系任意一个(其中某一个)学生年龄小的学生姓名和年龄

SELECT Sname，Sage

FROM Student

WHERE Sage < ANY

(SELECT Sage

FROM Student

WHERE Sdept= ' IS ')

AND Sdept <> ' IS ' ;

1. 带有EXISTS谓词的子查询
   1. EXISTS谓词

存在量词，带有EXISTS谓词的子查询不返回任何数据，只产生逻辑真值“true”或逻辑假值“false”

不关心子查询中查到了什么，只关心查到了没有，只需要告诉我有或者没有

若内层查询结果非空，则外层的WHERE子句返回真值

若内层查询结果为空，则外层的WHERE子句返回假值

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

例：查询所有选修了1号课程的学生姓名

SELECT Sname

FROM Student

WHERE EXISTS

(SELECT \*

FROM SC /\*相关子查询\*/

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

* 1. NOT EXISTS谓词

**存在与不存在的区别：**

**在用父的一行去子查询里找，EXISTS只要检测到一个，后边就不找了**

**NOT EXISTS如果真的不存在，需要把所有的都查了**

若内层查询结果非空，则外层的WHERE子句返回假值

若内层查询结果为空，则外层的WHERE子句返回真值

例：查询没有选修了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')；

* 1. 不同形式的查询间的替换

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：查询与“刘晨”在同一个系学习的学生

SELECT Sno，Sname，Sdept

FROM Student

WHERE **Sdept IN**

(SELECT Sdept

FROM Student

WHERE Sname= ‘ 刘晨 ’)；

SELECT Sno，Sname，Sdept

FROM Student S1

WHERE **EXISTS**

(SELECT \*

FROM Student S2

WHERE **S2.Sdept = S1.Sdept** AND S2.Sname = '刘晨 ';)

* 1. 用EXISTS/NOT EXISTS实现全称量词

SQL 语言中没有全称量词

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：



例：查询选修了全部课程的学生姓名

SELECT Sname

FROM Student

WHERE **NOT EXISTS**//不存在

SELECT \*

FROM Course

WHERE **NOT EXISTS**//不是它的

(SELECT \*

FROM SC

WHERE Sno= Student.Sno AND Cno= Course.Cno);

* 1. 用EXISTS/NOT EXISTS实现逻辑蕴函

可以利用谓词演算将逻辑蕴函谓词等价转换为：

例：查询至少选修了学生201215122选修的全部课程的学生号码

即不存在这样的课程y，学生201215122选修了y，而学生x没有选

SELECT DISTINCT Sno

FROM SC SCX

WHERE **NOT EXISTS**//不存在

(SELECT \*

FROM SC SCY

WHERE **SCY.Sno = ' 201215122 '** **AND NOT EXISTS**//它没有

(SELECT \*

FROM SC SCZ

WHERE **SCZ.Sno=SCX.Sno** AND **SCZ.Cno=SCY.Cno**))；

1. 集合查询
2. 并操作

<查询块>

UNION

<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：将多个查询结果合并起来时，系统自动去掉重复元组

UNION ALL：将多个查询结果合并起来时，保留重复元组

例：查询计算机科学系的学生及年龄不大于19岁的学生

SELECT \*

FROM Student

WHERE Sdept= 'CS'

**UNION**

SELECT \*

FROM Student

WHERE Sage<=19;

或

SELECT DISTINCT \*

FROM Student

WHERE Sdept= 'CS' **OR** Sage<=19;

1. 交操作

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

或

SELECT \*

FROM Student

WHERE Sdept= 'CS' AND Sage<=19；

1. 差操作

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;

或

SELECT \*

FROM Student

WHERE Sdept= 'CS' AND Sage>19;

1. 任何情况下，ORDER BY子句只能出现在最后

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**

对集合操作结果排序时，ORDER BY子句中用数字指定排序属性

1. 数据更新

DBMS在执行修改语句时会检查修改操作是否破坏表上已定义的完整性规则（实体完整性和用户定义完整性）

1. 数据的插入
2. 插入元组

INSERT

INTO <表名> [(<属性列1>[，<属性列2 >…)]

VALUES (<常量1> [，<常量2>] … )

将新的元组插入到指定表

·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，这样必须指定属性列**

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

示例：将一个新学生记录（学号：201215128；姓名：陈冬；性别：男；所在系：IS；年龄：18岁）（此顺序与表定义的顺序不一致）插入到Student表中。

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

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

1. 插入子查询结果

INSERT INTO <表名> [(<属性列1>[，<属性列2 >…)]

子查询

将子查询结果插入指定表中

子查询的结果必须包含和insert的字段列表一样多的字段，并且数据类型兼容

示例：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（无VALUES）

INSERT INTO Deptage(Sdept，Avgage)

SELECT Sdept，AVG(Sage) /\*\*子查询\*/

FROM Student

GROUP BY Sdept；

1. 数据的修改

UPDATE < 表名>

SET < 列名>=< 表达式>[ ，< 列名>=< 表达式>]…

[WHERE < 条件>];

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

=：在where子句中是判等，在别的语句中是赋值

1. 修改某元组的值

示例：将学生201215121的年龄改为22岁

UPDATE Student

SET Sage = 22

WHERE Sno=' 201215121 '；

1. 修改多个元组的值

示例：将所有学生的年龄增加1岁

UPDATE Student

SET Sage= Sage+1;//不支持Sage++

1. 带子查询的修改语句

示例：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)；

1. 数据的删除

DELETE

FROM < 表名>

[WHERE < 条件>];

删除指定表中满足WHERE子句条件的元组

WHERE子句：指定要删除的元组，缺省表示要删除表中的所有元组

1. 删除一个元组的值

示例：删除学号为201215128的学生记录

DELETE

FROM Student

WHERE Sno=' 201215128 '；

1. 删除多个元组的值

示例：删除所有学生的选课记录

DELETE

FROM SC

1. 带子查询的删除语句

示例：删除计算机科学系所有学生的选课记录

DELETE

FROM SC

WHERE 'CS' =

(SELECT Sdept

FROM student

WHERE student.Sno = SC.Sno)

1. 空值的处理

所谓空值（NULL）就是“不知道”或“不存在”或无意义的值

在where语句中，=是判等的意思，NULL是一个不确定的值，不能用=对NULL进行判断，在where语句中判断是不是NULL要用grade is NULL

在别的语句中，=是赋值的意思，可用set grade = NULL

空值NULL与空串’’不同

NULL与算术运算符——NULL

NULL与比较运算符——UNKNOW，不知道真假

1. 视图
2. 视图概述

视图是一个虚表（不存在的表），是从一个或几个基本表（或视图）导出的表

创建视图就是保存了一个查询过程

数据全部来源于基本表

对视图的所有操作（更新）都映射到基本表上

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**

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，保留了码

1. 定义视图

CREATE VIEW < 视图名> [(< 列名> [ ，< 列名>]…)]

AS < 子查询>

[WITH CHECK OPTION];

子查询不允许含有ORDER BY子句和DISTINCT短语

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）

示例：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 Student.Sno=SC.Sno AND Sdept= 'IS' AND SC.Cno= '1';

示例：（视图上定义视图）：建立信息系选修了1号课程且成绩在90分以上的学生视图

CREATE VIEW IS\_S2

AS

SELECT Sno，Sname，Grade

FROM IS\_S1

WHERE Grade>=90；

1. 删除视图

DROP VIEW <视图名> [CASCADE];

该语句从数据字典中删除指定的视图定义

**由该视图导出的其他视图定义仍在数据字典中**，但已不能使用，必须显式删除

删除基表时，由该基表导出的所有视图定义都必须显式删除

如果CASCADE选项，则删除该视图时会把由它导出的视图一块删除

1. 查询视图

从用户角度：查询视图与查询基本表相同

DBMS实现视图查询的方法：实体化视图、视图消解法

1. 实体化视图

有效性检查：检查所查询的视图是否存在

执行视图定义，**将视图临时实体化，生成临时表**

**查询视图转换为查询临时表**

查询完毕删除被实体化的视图(临时表)

1. 视图消解法

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

1. 更新视图

用户角度：更新视图与更新基本表相同

指定**WITH CHECK OPTION**子句后：DBMS在更新视图时会进行检查，防止用户通过视图对**不属于视图范围内**的基本表数据进行更新

CREATE VIEW IS\_Student

AS

SELECT Sno, Sname, Sage

FROM student

WHERE Sdept= 'IS'

WITH CHECK OPTION//更新的数据的Sdept必须是'IS'

DBMS实现视图更新的方法：视图实体化法、视图消解法

一些视图是不可更新的，因为对这些视图的更新不能唯一地有意义地转换成对相应基本表的更新(对两类方法均如此)

例：视图S\_G 为不可更新视图

CREATE VIEW S\_G (Sno，Gavg)//如果括号及内容省略，列名与下边的相同

AS

SELECT Sno，**AVG(Grade)//这列没名字，要在上边写上Gage或此处起别名**

FROM SC

GROUP BY Sno;

1. 视图的作用

视图能够**简化**用户的操作

视图使用户能以**多种角度**看待同一数据

视图对重构数据库提供了一定程度的逻辑独立性

视图能够对机密数据提供安全保护

适当的利用视图可以更清晰的表达查询