**三级模式**

子模式：也称子模式(Subschema)或用户模式，是数据库用户(包括应用程序员和最终用户)能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。

模式：对应于概念级也称概念模式逻辑模式，是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图。

内模式：对应于物理级内模式又称物理模式(Physical Schema)，它给出了数据库物理存储结构与物理存取方法，如数据存储的文件结构、索引、集簇及hash等存取方式与存取路径，内模式的物理性主要体现在操作系统及文件级上，它还不深入到设备级上(如磁盘及磁盘操作)，但近年来有向设备级发展的趋势(如原始磁盘、磁盘分块技术等)，DBMS一般提供相关的内模式描述语言(内模式DDL)。

内模式和模式只能有一个

**索引：**索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。（其实就是添个序号）

**属性：**属性通常指的是一个元组（表中的一行）中的一个数据项。属性是元组的组成部分，对应于表中的列。

**字段：**字段是表中的列，用于存储特定类型的数据。

每个字段都有个字段名，即列名，且可以定义该字段存储的数据类型（如整数、字符等）。

**表中的每一行叫做一个记录**

在关系数据库中，属性是表的列，而元组是表的行。每个属性包含特定行的数据

不等号:<>

Sql的数据定义

**表**

1. **表的建立**

**Create table student(**

**列名 数据类型 列级完整性约束**

**列名 数据类型 列级完整性约束**

**……**

**表级完整性约束）**

1. **表的删除**

**Drop table fx[ cascade|restrict ]**

**一般默认为cascade**

1. **Cascade,在删除表时，该表中的数据，表本身以及在该表上所建的索引和视图将全部随之消失；**
2. **restrict，只有先清除了表中的全部记录行数据，以及在该表上所建立的索引和视图后，才能删除一个空表，否则拒绝删除表**
3. **表的修改**
4. **表中新加列**

**Alter table fx add(name varchar(10))**

1. **删除列**

**Alter table fx drop name [ cascade | restrict ]**

**Cascade:删除某列时，所有引用该列的视图和约束也自动删除**

**Restrict：在没有视图或约束引用该属性时，才能被删除**

1. **修改列类型**

**Alter table fx modify name char(10);**

**索引**

**使用索引的情况：1.查询少量数**

**2.没有很多重复值，**

**3.需要重复操作**

**建立索引的目的：提供多种存取路径，加快查找速度**

**索引的建立**

**Crate [unique][cluster]index<索引名>**

**On<表名>（<列名1>[asc|desc],<列名2>[asc|desc]）**

**asc升序**

**Desc降序**

**例按学号升序建立唯一索引，按课程号升序建立唯一索引，按学号升序，课程号降序建立唯一索引**

**Create unique index sindex on student(sno);**

**Create unique index cindex on course(cno);**

**Create unique indext s\_cindex on s\_c(sno ASC,cno DESC)**

**索引的删除**

**Drop index [on <表名>]<索引名>**

**例：删除课程号中所建立的索引**

**Drop index cindex;**

**(不是非要注明哪个表)**

**Sql数据的查找**

**单表查询**

**聚集函数：**

**count([distinct | all]\*) //统计元组的个数**

**Count([distinct|all]<列名>) 变量名 //统计这一列中各个值的个数**

**SUM([distinct|all]<列名>） //计算数值型列名的综合**

**Avg（[distinct| all] <列名>）**

**Max([distinct| all]<列名>)**

**Min([distinct | all] <列名>)**

**如果指定distinct短语，在计算的时候不会算重复值。all为默认值，全部计算**

**查询的一般格式**

**Select[all|distinct] <目标列1>……**

**From <表名或视图名><表明或视图名>……**

**Where<条件表达式>**

**Group by<列名>[having<条件表达式>]**

**Order by<列名>[asc|desc]**

1. **all | distint,查找全部或者不同的，默认为all**
2. **使用group by的时候，select后的列只能是group by后接的列名，或者使用聚合函数，不能直接出现其他的列(也不一定，如果是一一对应的也可以，比如groupby sno,查找找sname也是可以的)。having是用于在计算出聚集函数值之后对查询结果进行控制，在个分组中选择满足条件的小组进行输出**
3. **比较：having:短语是在各组中选择满足条件的小组，而where子句是在表中选择满足条件的元组**

**例：求学生关系中湖北籍男生的每一年龄组（不少于30人）共有多少人，要求查询结果按人数升序排序，人数相同时按年龄降序排序**

**Select age,count(sno)number**

**From student**

**Where sex=’男’and place=’湖北省’**

**Group by age**

**Having number>30**

**Order by number , age DESC;**

**或者**

**Select age，avg(age)**

**From student**

**Where sex=’男’and place=’湖北省’**

**Group by age**

**Having count(sno)>30**

**Order by age DESC;(只展示年龄，学习having的写法)**

1. **%的使用,字符匹配,like的使用。查找所有姓刘或者名字第二个字为‘晓’字的学生姓名，学号，性别**

**大概率考，还要结合varchar和char在这里下滑线有什么区别**

**Select sname,sno,sex**

**From student**

**Where sname like ‘刘%’ or sname like ‘\_ \_晓%’；**

**中文汉字占两个字节**

**Char和varchar使用有什么区别**

**多表查询**

1. **嵌套查询（尽量用连接查询不用嵌套查询）**

**Select里面嵌套，having里面嵌套**

**子查询可以有选择的跟在谓词in，some，all，exist，not exist后面**

**例：找出学生表中年龄超过平均年龄的学生**

**Select sno**

**From student**

**Where age>**

**(select avg(age)**

**From student);**

**错误示例**

**Select sno**

**From student**

**Where age>avg(age)**

**这样写就不对，报错：invalid use of group founction**

**思考：聚集函数到底怎么用为什么这样写就不对，应该放在什么地方使用。**

**自己的理解：如果在where里面，对应只是一个age，但是如果放在select里面，对应的就是所有满足条件的age，可以使用avg这种函数**

**说明：子查询是嵌套在父查询的where条件中的。**

**子查询不能使用order by，因为order by只能对最终结果进行排序**

**例2（谓词in的使用）**

**查找选修了数据库原理课程的学生的学号，成绩（用连接查询就很方便）**

**Select sno,grade**

**From s\_c**

**Where cno in**

**(**

**Select cno**

**From course**

**Where cname = ‘数据库原理’**

**)**

**例3(谓词all和some的使用)**

**找出平均成绩最高的学生号：**

**Select sno**

**From s\_c**

**Group by sno**

**Having avg(grade)>=ALL**

**(**

**Select avg(grade)**

**From s\_c**

**Group by sno);**

**总结：all表示所有，some表示存在**

1. **条件连接查询（比较重要）**

**通过连接使查询的数据从多个表中获得**

**查询中用来连接两个表的条件称为连接条件**

**嵌套查询的例2**

**Select sno,grade**

**From s\_c,course**

**Where course.cno = s\_c.cno and cname=’数据库原理’**

**例2：**

**按平均成绩降序给出所有课程都及格的学生（号，名）及其平均成绩，其中成绩统计时不包括008号考查课**

**Select student.sno,student.sname,avg(grade) avg\_g**

**From student, s\_c**

**Where student.sno = s\_c.sno and cno<>’008’ //666好别致的写法**

**Group by sno**

**Having MIN(grade)>=60**

**Order by avg\_g desc;**

**注意：最后得到的列名为avg\_g，如果不起别名的话查找出来的列名就是avg（grade）**

1. **表自身与表自身连接查询很巧妙，自己不一定想的出来**

**找出年龄比王迎同学大的学生的姓名和年龄**

**Select s1.sname,s1,age**

**From student s1,student s2**

**Where s1.age>s2.age and s2.sname = ‘王迎’；**

1. **相关子查询**

**例：查询所有选修了005号课程的同学的姓名和学号**

**Select sname,sno**

**From student**

**Where sno exists(**

**Select \***

**From s\_c**

**Where student\_sno=s\_c.sno and cno = ‘005’**

**)**

**原理：现在学生表里面遍历，每一个学号，看在嵌套查询里面是否有满足条件的。**

**Exists最后的返回值就是true或者false，所以子查询就都写\*，写列名没有意义**

1. **查询没有选修001号课程的学生学号及姓名**

**方法一（not exists）**

**Select sno ,student**

**From student**

**Where sno not exists(**

**Select \***

**From s\_c**

**Where student.sno=s\_c.sno and cno = ‘001’**

**);**

**方法二（all）太妙了**

**Select sno,sname**

**From student**

**Where sno<>all**

**(**

**Select sno**

**From s\_c**

**Where cno=’001’**

**)**

**总结1：一些带exists,not exists的谓词不能被其他谓词替换，但是in,some,all,比较运算符对应的子查询都能用exists和not exists替换**

**总结2：Exists 和not exists**

**不管是exists还是not exists，关键是要理解相关子查询什么时候返回**

1. **子查询找到一个匹配的立刻反回**
2. **子查询遍历关系所有的元组，没有找到匹配的返回为空**

**！！！最容易犯得错误是not exists查询，当找到一个不匹配的时候不会返回，要全部遍历完以后才会返回，从exists出发**

**Not exists 谓词的使用**

**究极难题，双嵌套**

**例：找出选修了全部课程的同学**

**Select sname**

**From student**

**Where not exists**

**(**

**Select \***

**From course**

**Where not exists**

**(select \***

**From s\_c**

**Where s\_c.sno=student.sno and s\_c.cno = course.cno));**

**思考：怎么理解这个双嵌套查询？为什么要用两个not exists？**

**逻辑：双重否定等于肯定，可以这么理解查询这样的一些学生，没有一门课不是他选修的**

**Ans\_1 = select cno**

**From course**

**where not exists**

**(select sc.sno from sc where sc.sno=student.sno and sc.cno=student.cno)**

**Ans\_2 = select sc.cno**

**From sc**

**Where sc.sno=student.sno and sc.cno=course.cno);**

**过程理解，先从第一层选一个元组，再从第二层选一个元组，再从第三层选一个元组。如果符合要求则ans\_2返回一组值，由于not\_exists,ans\_1继续为空。因此以此类推，如果一个学生选修了所有课他对应的course遍历完之后ans\_1仍然为空，因此会返回一个空，最后结果为真。如果一个学生没有选修所有的课程，存在一门课在最内层子查询肯定全部为空，最后返回一个空,ans\_1为真，ans\_1就得返回，然后最外层就为假，不输出。**

**4.集合查询（集合查询所用的谓词union,intersect,except）**

**顾名思义就是一个集合一个集合的查询,union：两个集合求并集，intersect：两个集合求交集，except：两个集合求差集**

**例：某学院有三张表：一般研究生表st1，在职研究生表st2，全体研究生中的学生干部表st3**

1. **查询全部研究生情况**

**(select \*from st1)**

**Union**

**(select\* from st2)**

1. **查询非干部的在职研究生的学生情况**

**(select \* from st2)**

**Except**

**(select \* from st3)**

1. **求选修了1号或2号，但没有选修3号课程的同学的学生号**

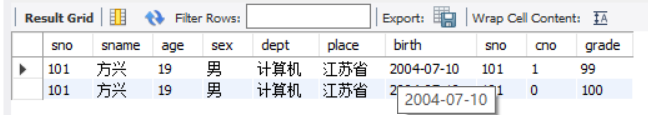
**(select )**

**连接查询**

1. **自然连接查询（等值连接）**

**select \***

**from student join s\_c**

**where dept = '计算机' and s\_c.sno=student.sno;**

将两个表强行的连在了一起

2.内连接查询，和等值查询相同

3.外连接查询

1）左外连接：包含表一所有数据和表二与表一重叠的数据

Select 列名 from 表1 left outer join 表二 on 条件



1. 右外连接：包含表二所有数据和表一与表二重叠的数据

Select 列名 from 表1 right outer join 表二 on 条件

1. 全外连接：包含表一表二所有数据

Select 列名 from 表1 full outer join 表二 on 条件

**递归子查询**

**With查询：将一个复杂的查询分解成几个子查询，计算出中间结果，对中间结果继续查询。**

**With cte\_name as**

**(**

**Query**

**)**

**Query**

**例子：查询选修课程数超过三门的同学数**

**正差写法**

**Select sno,sum(cno)cnum**

**From s\_c**

**Where**

**Group by sno**

**Having cnum>3**

**With写法**

**With c\_num as(**

**Select sno,count(cno)cnum from s\_c group by sno)**

**Select sno,cnum**

**From c\_num**

**Where cnum>=3)**

**sql中的空值问题**

**空值的判断采用is null，不能用=**

**不是空值：is not null**

**例：查询选课但是没有成绩的同学的学号，课程号**

**Select student.sno,cno**

**From student,s\_c**

**Where grade is null;**

**in子查询中的空值**

**错误示例 select employee\_id**

**from employee**

**where employee\_id not in (**

**select manager\_id**

**from employee)大概意思是找到不算管理者的员工（没有下属），但是如果manager\_id存在空值，那查询就为空，因为空值即不知道，没有办法和其他值比较**

**正确的sql语句：**

**Select employee\_id**

**From employee**

**Where employee\_id not in(**

**Select manager\_id**

**From employee**

**Where manager\_id is not null)**

视图的创建

嵌套查询的一般求解方法时先求解子查询，其结果用于建立父查询的查找条件

Create or replace view fz(no,name)

//or replace 表示如果表已经存在了则替换掉表

//注意：括号用来指定索引名，这个括号可加可不加，如果不加的话就用原表中的索引。且括号内的列名数量必须与select后的查找数量相同

As

（子查询）

select sno,sname

from student

where age >=19;

删除视图

drop view fx {restrict | cascade}

视图查询

查看创建视图的语句：show create view 视图名称

show create view fx

查看视图的数据：select \*from 视图名词……；（同表的查询操作相同）

视图的更新

不是所有的视图都是可以更新的，比如某视图的一个属性是成绩表各科的平均分。

1.

视图的更新可以转化成对于表的更新，即对查询内容本身进行了修改

看似修改了视图，实则删除了表里面的数据

如

Delete

From fx

Where sno = 101

但是要注意了

如果对视图的修改不满足定义视图的条件

比如

Create view fx as select sname,age from student where age>=19 with check option;

Insert into fx values(‘王晓琴’,18);

加入with check option 就会报错，阻止这条数据的插入。可以避免这种情况发生

1. create or replace +……和创建视图相同

3.也可以不对视图的内容进行修改，即修改的只是视图查找的范围，比如减少查找的属性

alter view fx

as

select sname from student where age>=19

约束

默认约束（default）定义一个默认的缺省值

Create table fx(

Sno int default 100;

)

存储过程

相当于写了个函数

创建存储过程

模板

Create procedure 存储过程名称（[参数列表]）

Begin

--sql语句

End;

#将//定义为语句分隔符替换；，用；无法运行

delimiter //

create procedure p1()

begin

select \* from student;

end//

调用存储过程

Call 名称（[参数]）

删除存储过程

删除存储过程的时候没有参数不加（）

Drop procedure p1

带参数的存储过程

参数有输入参数in，输出参数out 这一类参数作为输出的返回值,输入输出参数 inout 即时输入也是输出。不加参数的话默认为in

If语句＋带参数

delimiter //

create procedure p3(score int)

begin

-- declare score int default 58;

declare result varchar(2);

if score >=85 then

set result := '优秀';

elseif score >=60 then

set result := '及格';

else

set result := '不及格';

end if;

select result;

end//

例二，含有传入参数，传出参数

delimiter //

create procedure p3(in score int,out result int)

begin

-- declare score int default 58;

declare result varchar(2);

if score >=85 then

set result := '优秀';

elseif score >=60 then

set result := '及格';

else

set result := '不及格';

end if;

end//

delimiter ;

call p3(85,@result);

select @result;

例三：inout，将传入的200分制的分数转化成100分制的返回

delimiter //

create procedure p4(inout score double)

begin

set score := score\*0.5;

end;

set @score = 75;

call p4(@score);

select @score;

将一个被赋过值的变量作为参数

**触发器**

**例：将被修改过的成绩和之前的成绩放在e\_s表中**

**创建表**

create table e\_s

(sno varchar(3),cno varchar(5),o\_grade varchar(3),n\_grade varchar(3));

创建触发器

tridelimiter //

create trigger tri

after update on s\_c #oracle可以指定某个字段比如after update #of grade on s\_c，mysql好像用不了

for each row #这句话在mysql是固定的

begin

insert into e\_s values(old.sno,old.cno,old.grade,new.grade);

end//

#如果不加old或者new，插入的值就是空

注意delete，update，delete的区别：

Delete只要更新后的

Update 前后都要

Delete 只要更新前的

触发操作

update s\_c

set grade = '100'

where sno = '101' and cno = '0'

PL/SQL

用户自定义变量：用户变量不用提前声明，在使用的时候直接用@变量名 使用就可以

Set @a=’’

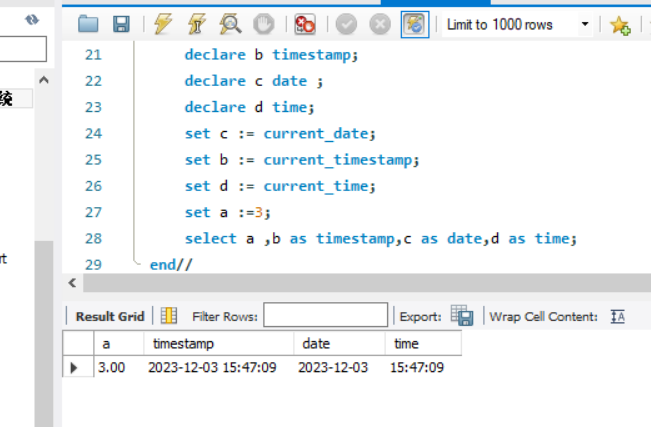
Set @b:=’’

Select @b:=’’ //这个会直接输出

Numeric(a,b),一共a个数字，小数点占b位如果赋值的时候没给全自动用0补上

常量的声明与变量的声明类似。但需要加上constant关键字并立即为其赋值，赋值以后的常量不能再被赋值

时间的写法



循环

while循环：while + 条件+do +end while

满足条件进行循环

-- 用循环计算从1累加到n的值，n为传入的参数值

delimiter //

create procedure p7(in n int)

begin

declare total int default 0;

while n>0 do

set total := total + n;

set n := n -1;

end while;

select total;

end;

call p7(10);

drop procedure p7;

Repeat循环：满足条件推出循环

Repeat

sql逻辑

Until 条件

End repeat；

-- repeat循环尝试

-- 例：计算从1累加到n的值

delimiter //

create procedure p8(in n int)

begin

declare total int default 0;

repeat

set total := total + n;

set n = n-1;

until n=0 -- 后面没有分号

end repeat;

select total;

end//

delimiter ;

call p8(100);

drop procedure p8;

Loop循环

Loop配合两个语句一起使用：

Leave:配合循环使用，推出循环。（break）

Iterate：必须用在循环中，作用是跳出当前循环剩下的语句，直接进入下一个循环（continue）

[begin\_label:]loop

sql逻辑

End loop[end\_loop];

例一：使用leave推出循环

-- loop循环，计算从1累加到n的值

delimiter //

create procedure p9(in n int)

begin

declare total int default 0;

sum:loop

if n <= 0 then leave sum;

end if;

set total := total +n;

set n := n-1;

end loop sum;

select total;

end

delimiter ;

call p9(10);

drop procedurep9;

例二：

-- loop循环，计算从1到n偶数累加的值，iterate的使用

delimiter //

create procedure p10(in n int)

begin

declare total int default 0;

sum:loop

if n <= 0 then leave sum;

end if;

if n%2 = 1 then

set n = n-1;

iterate sum;

end if;

set total := total +n;

set n := n-1;

end loop sum;

select total;

end

delimiter ;

call p10(10);

drop procedure p10;

用语句为变量赋值

例：根据职员表emp计算7788号职员的奖金。奖金的计算方法是：职员的工资（sal）\*15%

Declare

Bonus\_rate constrant number(2,2) := 0.15//常量的声明，整数位允许有两个值，允许有两位小数

Bonus number(7,2)

Emp\_id number(4) :=7788;

Begin

Select sal \*bonus\_rate into bonus from emp where emono = emp\_id

Dbms\_output.put\_line(‘employee:’||TO\_CHAR(emp\_id)||‘bonus：’||To\_char(bonus)||’bonus rate:’||to\_char(bonus\_rate));

end

To\_char 将数值型或者日期型转化成字符型

Case语句

当一个字段由于某些条件不同，而赋予不同的值时，可以使用case语句

形式:

Case when 条件1 then 表达式1

When 条件2 then 表达式2

……

Else 表达式

End CASE

小规律：end后面一般都要注明是什么语句，比如end case,end if.

例-- case函数尝试

delimiter //

create procedure p5(in score int,out result varchar(4))

begin

case

when score >=85 then

set result := '优秀';

when score >=60 then

set result := '及格';

else

set result := '不及格';

end case;

end//

delimiter ;

call p5(60,@result);

select @result;

复合型触发器

存储过程和存储函数

游标

Declare 游标名称 cursor for 查询语句；

打开游标 open 游标名称

Fetch 游标名称 into 变量

注意：先声明变量，再声明游标

CREATE PROCEDURE curdemo()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE a CHAR(16);

DECLARE b, c INT;

DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1; DECLARE cur2 CURSOR FOR SELECT i FROM test.t2; DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur1;

OPEN cur2;

read\_loop: LOOP FETCH cur1 INTO a, b;

FETCH cur2 INTO c;

IF done THEN LEAVE read\_loop;

END IF;

IF b < c THEN INSERT INTO test.t3 VALUES (a,b);

ELSE INSERT INTO test.t3 VALUES (a,c);

END IF;

END LOOP;

CLOSE cur1;

CLOSE cur2;

END;

条件处理程序handler：可以用来定义在流程控制结构执行过程中遇到问题时相应的处理步骤

Declare handler\_action handler for condition\_values

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

Handle\_action

Continue:继续执行当前程序

Exit：终止当前程序

Condition\_value

Not found 游标没有值了以后

Handle action:continue:继续执行当前的程序

Exit：终止当前的程序

存储函数

存储函数是有返回值的存储过程，存储函数的参数只能是in类型的

Create function 存储函数名称（[参数列表]）

Returns type [characteristic]

Begin

sql语句

Return……；

End；

Characteristic 说明：

Determinstic ：相同的输入参数总是产生相同的结果

No sql：不包含sql语句

Reads sql data：包含读取数据的语句，但不包含写入数据的语句

域约束和断言

根据生日写出年龄 set age := timestampdiff（year,birth,now())后两个参数相比较，第一个参数为比较的范围