**数据库系统实验报告**

学号：21009200038

姓名：江昱峰

手机：13914250795

邮箱：21009200038@stu.xidian.edu.cn

阿里云账号：昨夜如梦0795

完成时间：2023 年12月27日

# 线上作业：阿里PolarDB与XDSQL

### PolarDB远程连接

#### 1.1 阿里PolarDB申请使用

##### 1.1.1 准备环境和资源

按以下步骤准备环境和资源：

（1）访问阿里云免费试用。单击页面右上方的登录/注册按钮，并根据页面提示完成账号登录（已有阿里云账号）、账号注册（尚无阿里云账号）或实名认证（根据试用产品要求完成个人实名认证或企业实名认证）。

（2）成功登录后，在产品类别下选择数据库 > 关系型数据库，

（3）在免费试用产品区域的云原生数据库 PolarDB MySQL版 Serverless 5000PCU 100 GB卡片上单击立即试用。

（4）在云原生数据库PolarDB MySQL版面板，完成参数配置。

VPC网络和交换机：

* 如果已创建符合您网络规划的VPC和交换机，直接选择该VPC和交换机。
* 如果未创建符合您网络规划的VPC和交换机，可以使用默认的VPC和交换机。
* 如果以上已创建的和默认的VPC和交换机无法满足要求，可以自行创建VPC和交换机，详情请参见创建和管理专有网络。

其他参数：保持默认值或按需修改。

（5）勾选服务协议后，单击立即试用，并根据页面提示完成试用申请。

完成试用申请后，将会自动创建集群。创建集群需要10~15分钟，可以在集群列表中看到新创建的集群。当集群状态为运行中时，即可正常使用。

##### 1.1.2 查看Serverless资源包信息

（1）可以通过以下两种方法查看Serverless资源包的信息：

* 申请试用成功后，单击管理试用。
* 登录PolarDB控制台，左侧导航栏单击计算包，选择资源维度为资源包。

（2）在实例汇总页签，找到试用Serverless资源包，可以查看资源包总量和剩余量等信息。

（3）在使用明细页签，找到试用Serverless资源包，可以查看每次抵扣的抵扣量、实例原始用量和抵扣系数等信息。

##### 1.1.3 创建数据库账号

（1）登录PolarDB控制台，单击左侧集群列表，然后选择云产品资源提供的地域。例如：华东1（杭州）。

（2）创建数据库账号。

1. 在集群列表页面，单击集群ID，进入基本信息页面。
2. 单击左侧导航栏配置与管理>账号管理。
3. 单击左上方创建账号。
4. 参考说明配置账号信息，然后单击确定。请记录此处创建的数据库账号及密码，后续步骤中会反复使用。

* 数据库账号：输入数据库账号名称，例如：temp。
* 账号类型：选择高权限账号。
* 密码：设置账号密码。
* 确认密码：再次输入密码。

##### 1.1.4 创建数据库

（1）在PolarDB控制台单击左侧集群列表，然后选择集群所在地域。例如：华东1（杭州）。

（2）找到目标集群，单击集群ID。

（3）单击左侧导航栏配置与管理>数据库管理。

（4）单击创建数据库。

（5）在创建数据库对话框中，输入数据库（DB）名称，其他参数保持默认配置即可。然后单击确定。

请勿使用保留关键字作为数据库名称，如test、mysql等。

**1.1.5 使用DMS连接集群**

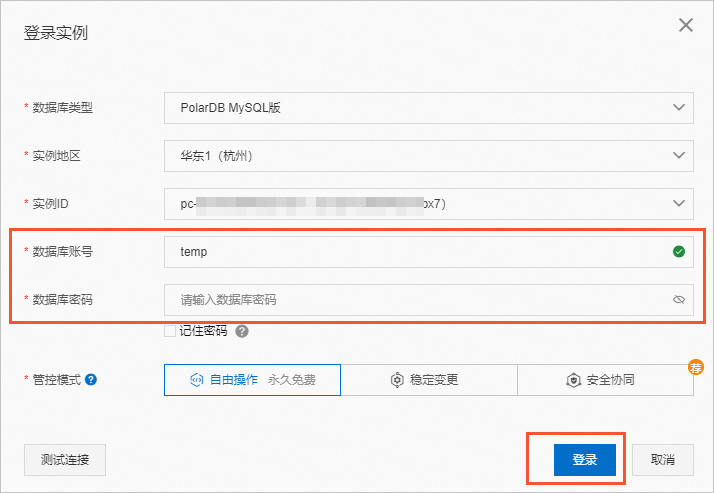
（1）在PolarDB控制台单击左侧集群列表，然后选择集群所在地域。例如：华东1（杭州）。

（2）找到目标集群，单击集群ID。

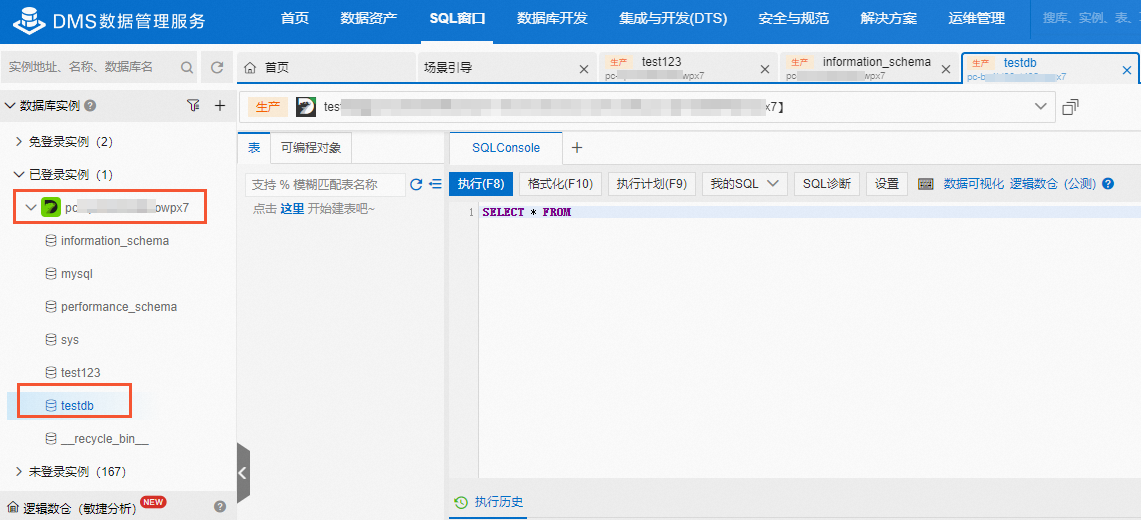
（3）在基本信息页面右上角，单击登录数据库。



（4）在弹出的对话框中，输入PolarDB MySQL版Serverless集群中创建的数据库账号和数据库密码，单击登录。



（5）登录DMS后，在左侧导航栏的已登录实例列表中，单击目标集群名称，找到并双击目标数据库名称，即可切换到目标数据库进行管理。



#### **1.1.6 使用数据库**

连接PolarDB MySQL版Serverless集群后，您可以和正常使用自建数据库一样对PolarDB MySQL版Serverless集群进行操作（DDL、DML等）。

1. 在测试数据库testdb中创建1个测试表test\_table，并插入3条数据。

-- 创建一个名为test\_table的测试表

CREATE TABLE test\_table (

  id INT NOT NULL AUTO\_INCREMENT,

  name VARCHAR(50) NOT NULL,

  age INT NOT NULL,

  PRIMARY KEY (id)

);

-- 向test\_table表中插入3条数据

INSERT INTO test\_table (name, age) VALUES

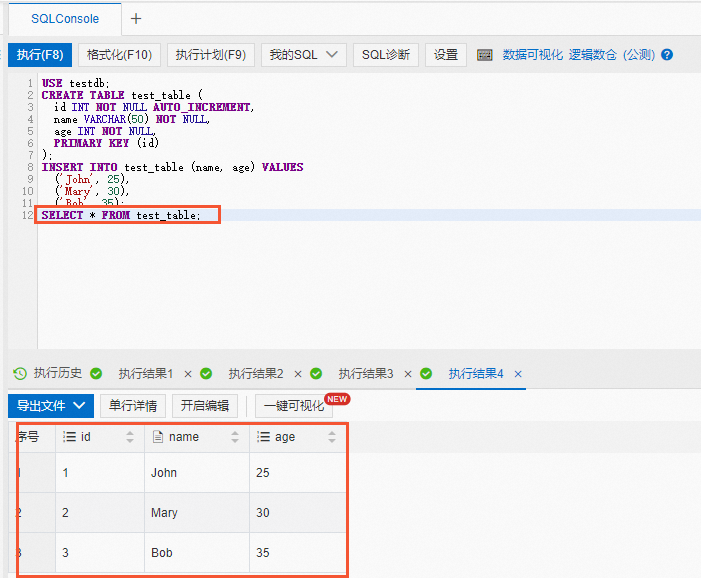
  ('John', 25),

  ('Mary', 30),

  ('Bob', 35);

1. 查询test\_table表中数据。

 SELECT \* FROM test\_table;



（3）修改test\_table表中数据。

UPDATE test\_table

SET name = 'Johnny', age = 26

WHERE id = 1;

##### 1.1.7 完成

使用SELECT \* FROM test\_table;命令，可以查看到修改后的表数据。



##### 1.1.8 清理及后续

* 清理：PolarDB MySQL版Serverless集群到期未释放可能产生欠费。如不再使用集群，请登录PolarDB控制台，找到目标集群，在操作列单击更多>释放，按照界面提升手动释放集群。
* 后续：在试用有效期期间，您还可根据希望测试的其它业务场景继续使用PolarDB MySQL版Serverless集群。

#### 1.2 高级语言远程连接PolarDB

根据连接信息提示，使用Python或Java实现对数据库的远程访问：

下面我选择用Python连接PolarDB for MySQL数据库（以Python3为例）：1. 安装Python，版本3.6.5以上即可

2. 在命令行中输入以下语句，安装pymysql模块

pip install pymysql

3. 创建connect.py文件，在文件内输入以下代码，其中中需要把Host、Port、Database、User、Password换为实验步骤第4步中连接信息内的相关参数。Host换为连接地址值，Port换为端口值，Database换为数据库名称，User换为账号，Password换为为密码。

1. 运行connect.py程序，实现远程连接PolarDB for MySQL。

### XDSQL与PolarDB对比实验

**2.1 XDSQL执行的SQL语句及结果**

##### 2.1.1 创建数据库

SQL语句如下：

CREATE DATABASE xjgl;

结果如下图所示：



运行成功。

##### 2.1.2 切换数据库

##### SQL语句如下：

USE DATABASE xjgl;

结果如下图所示：



运行成功。

##### 2.1.3 定义模式/删除模式

##### 不支持此类语句。

##### 2.1.4 基本表的定义、删除与修改

（1）定义基本表

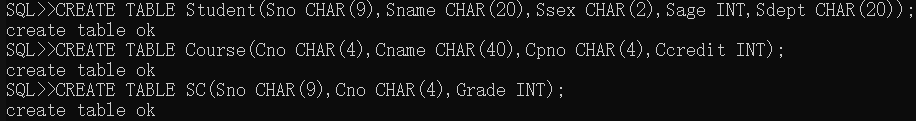
创建学生表Student、课程表Course、学生选课表SC，SQL语句如下：

CREATE TABLE Student(Sno CHAR(9),Sname CHAR(20),Ssex CHAR(2),Sage INT,Sdept CHAR(20));

CREATE TABLE Course(Cno CHAR(4),Cname CHAR(40),Cpno CHAR(4),Ccredit INT);

CREATE TABLE SC(Sno CHAR(9),Cno CHAR(4),Grade INT);

结果如下图所示：



运行成功。

（2）删除基本表

SQL语句如下：

DROP TABLE Student;

结果如下图所示：



运行成功。

（3）修改基本表

SQL语句如下：

向Student表增加”出生日期“列，数据类型为日期型：

ALTER TABLE Student ADD birthdate DATE;

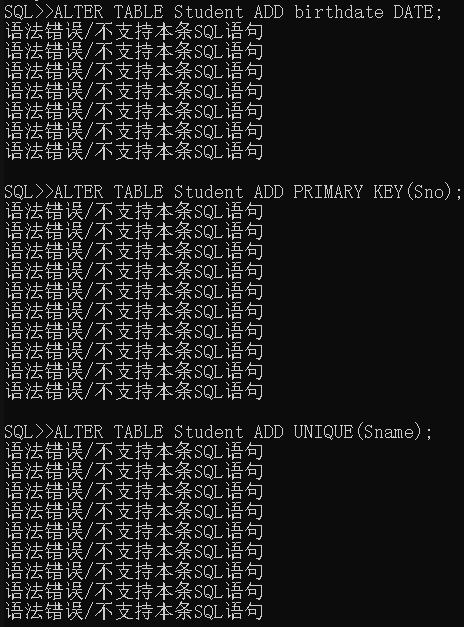
向Student表增加Sno列为主码的约束条件：

ALTER TABLE Student ADD PRIMARY KEY(Sno);

向Student表增加Sname列为唯一值的约束条件：

ALTER TABLE Student ADD UNIQUE(Sname);

结果如下图所示：



显示”语法错误/不支持本条SQL语句“。

##### 2.1.5 完整性约束条件

##### 重新创建带有完整性约束条件的用户数据表，SQL语句如下：

CREATE TABLE Student(Sno CHAR(9) PRIMARY KEY,Sname CHAR(20) UNIQUE,Ssex CHAR(2),Sage SMALLINT,Sdept CHAR(20));

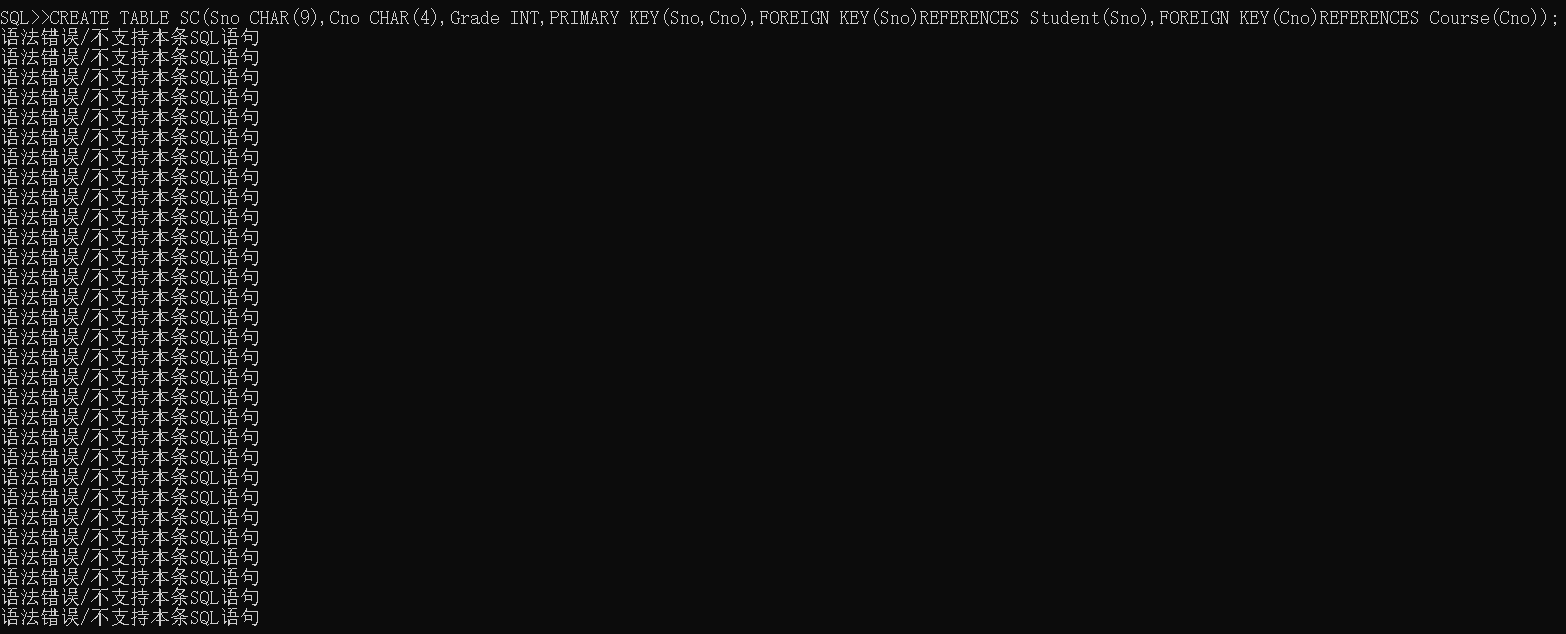
CREATE TABLE Course(Cno CHAR(4) PRIMARY KEY,Cname CHAR(40) NOT NULL,Cpno CHAR(4),Ccredit INT,FOREIGN KEY(Cpno) REFERENCES Course(Cno));

CREATE TABLE SC(Sno CHAR(9),Cno CHAR(4),Grade INT,PRIMARY KEY(Sno,Cno),FOREIGN KEY(Sno)REFERENCES Student(Sno),FOREIGN KEY(Cno)REFERENCES Course(Cno));

结果如下图所示：







显示”语法错误/不支持本条SQL语句“。

##### 2.1.6 索引

（1）建立索引

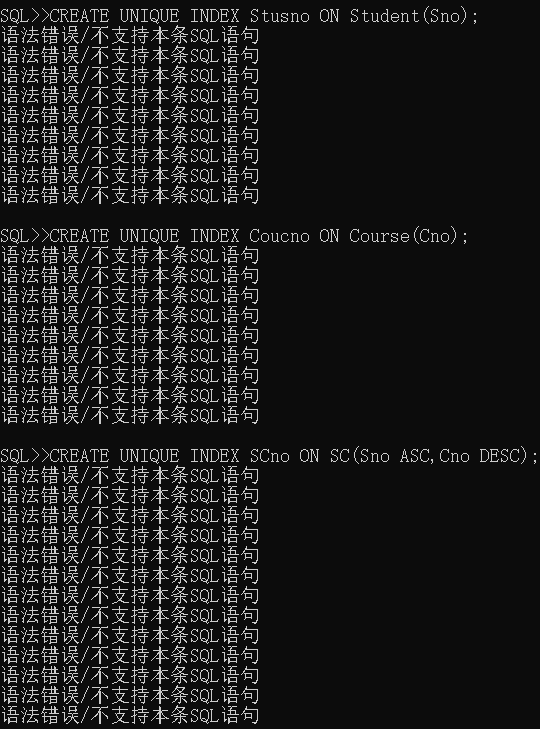
SQL语句如下：

CREATE UNIQUE INDEX Stusno ON Student(Sno);

CREATE UNIQUE INDEX Coucno ON Course(Cno);

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

结果如下图所示：



显示”语法错误/不支持本条SQL语句“。

（2）删除索引

SQL语句如下：

DROP INDEX Stusno ON Student;

结果如下图所示：



显示”语法错误/不支持本条SQL语句“。

**2.1.7 数据查询**

以下语句在未加约束的三个表上执行。

（1）插入数据

运行以下SQL语句：

INSERT INTO Student VALUES('201215121','李勇','男',20,'CS');

INSERT INTO Student VALUES('201215122','刘晨','女',19,'CS');

INSERT INTO Student VALUES('201215123','王敏','女',18,'MA');

INSERT INTO Student VALUES('201215124','张立','男',19,'IS');

INSERT INTO COURSE VALUES('1','数据库','5',4);

INSERT INTO COURSE VALUES('3','信息系统','1',4);

INSERT INTO COURSE VALUES('4','操作系统','6',3);

INSERT INTO COURSE VALUES('5','数据结构','7',4);

INSERT INTO COURSE VALUES('7','PASCAL语言','6',4);

INSERT INTO SC VALUES('201215121','1',92);

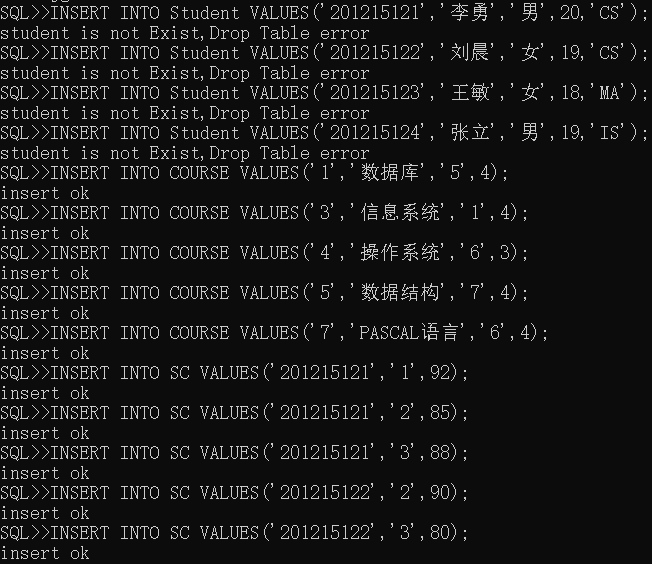
INSERT INTO SC VALUES('201215121','2',85);

INSERT INTO SC VALUES('201215121','3',88);

INSERT INTO SC VALUES('201215122','2',90);

INSERT INTO SC VALUES('201215122','3',80);

结果如下图所示：



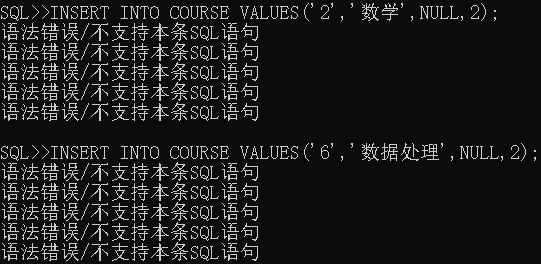
运行成功。

运行以下语句：

INSERT INTO COURSE VALUES('2','数学',NULL,2);

INSERT INTO COURSE VALUES('6','数据处理',NULL,2);

结果如下图所示：



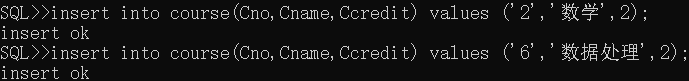
显示”语法错误/不支持本条SQL语句“。

运行以下语句：

insert into course(Cno,Cname,Ccredit) values ('2','数学',2);

insert into course(Cno,Cname,Ccredit) values ('6','数据处理',2);

结果如下图所示：



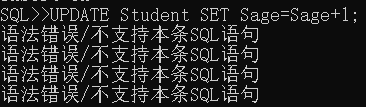
运行成功。

（2）更新数据

运行以下语句：

UPDATE Student SET Sage=Sage+1;

结果如下图所示：



显示”语法错误/不支持本条SQL语句“。

运行以下语句：

UPDATE Course SET Cpno='1' WHERE Cno='6';

结果如下图所示：



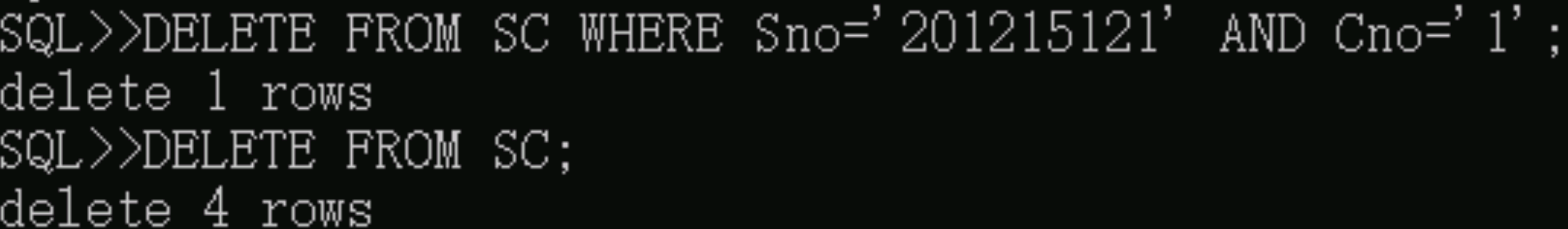
运行成功。

（3）删除数据

DELETE FROM SC WHERE Sno='201215121' AND Cno='1';

DELETE FROM SC;

结果如下图所示：

 运行成功。

（4）查询数据

a. 单表查询

运行以下SQL语句：

SELECT \* FROM Student;

SELECT \* FROM Sc;

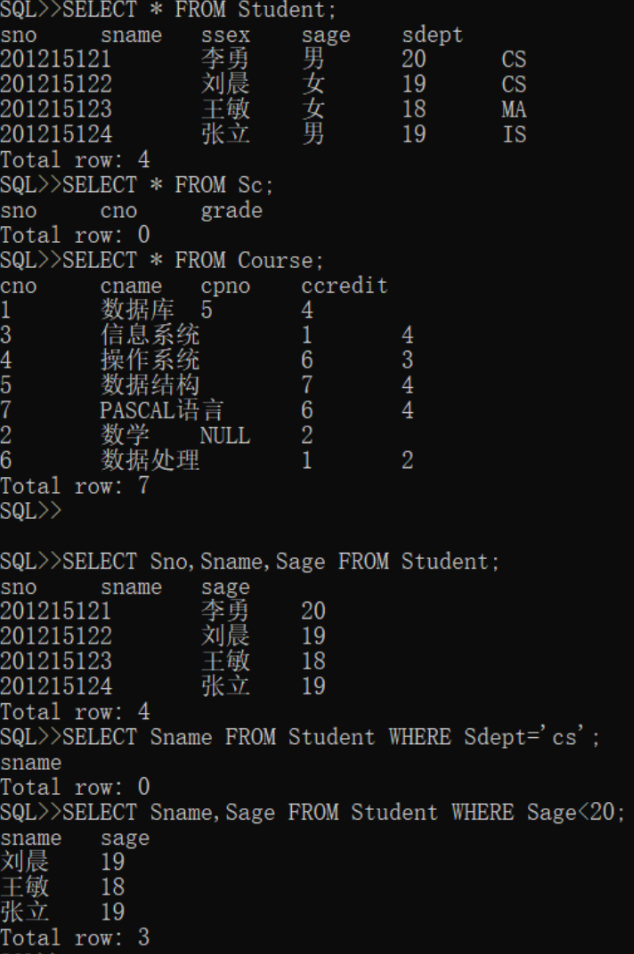
SELECT \* FROM Course;

SELECT Sno,Sname,Sage FROM Student;

SELECT Sname FROM Student WHERE Sdept='cs';

SELECT Sname,Sage FROM Student WHERE Sage<20;

结果如下图所示：

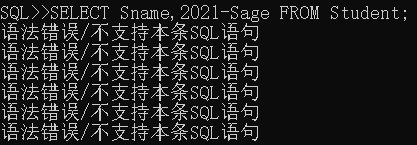


运行成功。

运行以下SQL语句：

SELECT Sname,2021-Sage FROM Student;

结果如下图所示：

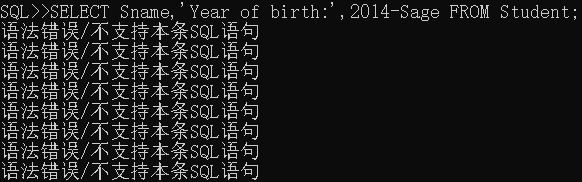


显示”语法错误/不支持本条SQL语句“。

运行以下SQL语句：

SELECT Sname,'Year of birth:',2014-Sage FROM Student;

结果如下图所示：



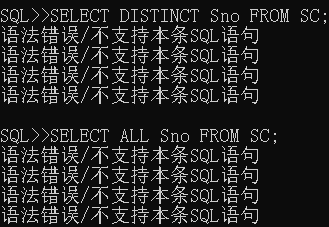
显示”语法错误/不支持本条SQL语句“。

运行以下SQL语句：

SELECT DISTINCT Sno FROM SC;

SELECT ALL Sno FROM SC;

结果如下图所示：



显示”语法错误/不支持本条SQL语句“。

运行以下SQL语句：

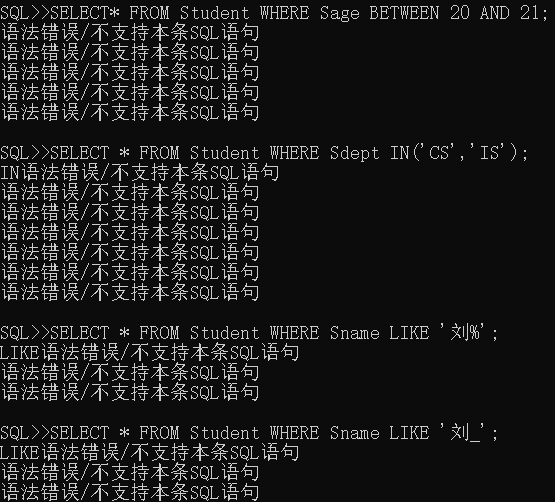
SELECT\* FROM Student WHERE Sage BETWEEN 20 AND 21;

SELECT \* FROM Student WHERE Sdept IN('CS','IS');

SELECT \* FROM Student WHERE Sname LIKE '刘%';

SELECT \* FROM Student WHERE Sname LIKE '刘\_';

结果如下图所示：



显示”语法错误/不支持本条SQL语句“。

运行以下SQL语句：

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

SELECT COUNT(\*) FROM Student;

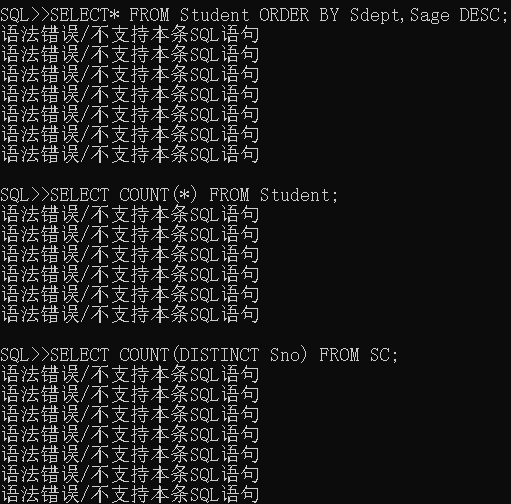
SELECT COUNT(DISTINCT Sno) FROM SC;

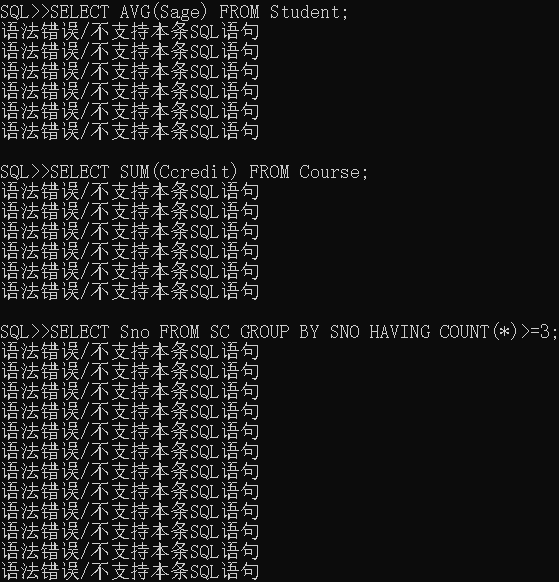
SELECT AVG(Sage) FROM Student;

SELECT SUM(Ccredit) FROM Course;

SELECT Sno FROM SC GROUP BY SNO HAVING COUNT(\*)>=3;

结果如下图所示：





显示”语法错误/不支持本条SQL语句“。

b. 多表查询

运行以下SQL语句：

SELECT Student.Sno,Sname FROM Student,SC WHERE Student.Sno=SC.Sno AND SC.Cno='2';

结果如下图所示：



运行成功。

运行以下SQL语句：

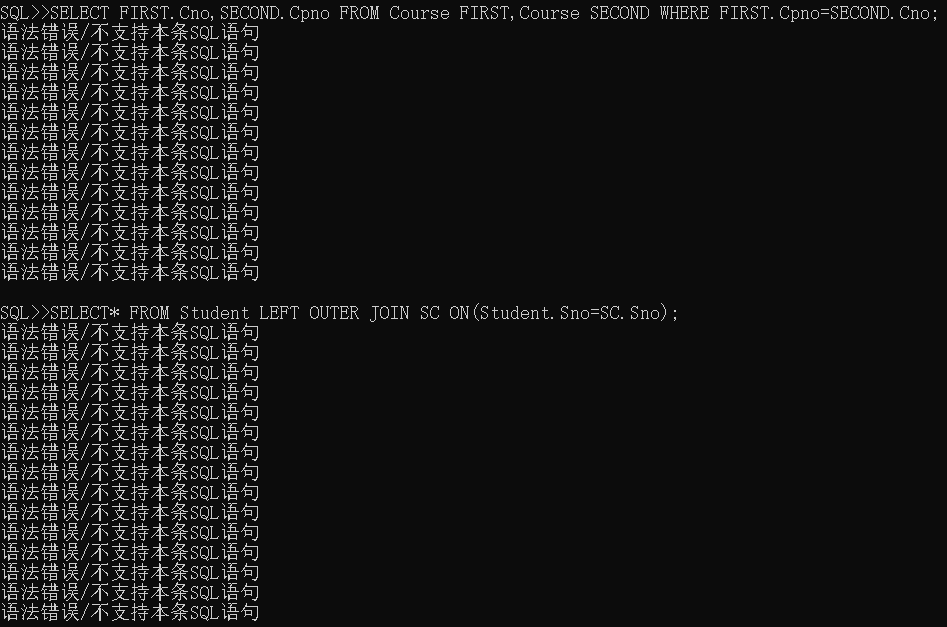
SELECT FIRST.Cno,SECOND.Cpno FROM Course FIRST,Course SECOND WHERE FIRST.Cpno=SECOND.Cno;

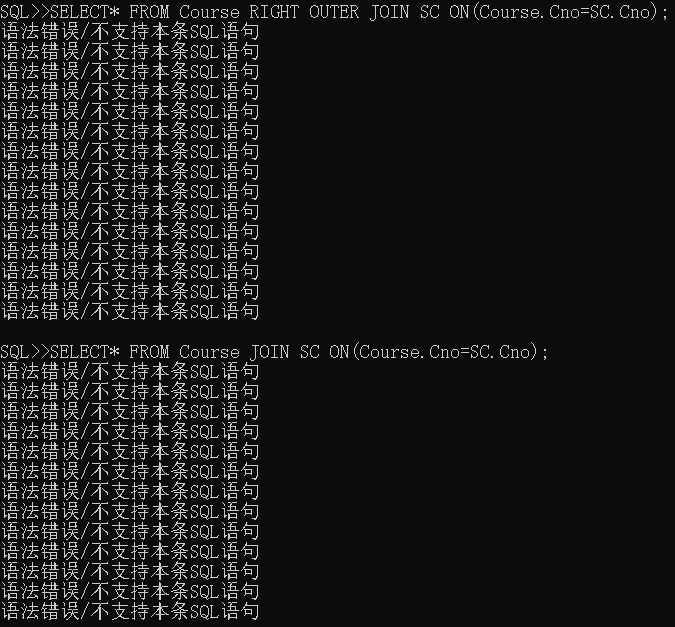
SELECT\* FROM Student LEFT OUTER JOIN SC ON(Student.Sno=SC.Sno);

SELECT\* FROM Course RIGHT OUTER JOIN SC ON(Course.Cno=SC.Cno);

SELECT\* FROM Course JOIN SC ON(Course.Cno=SC.Cno);

结果如下图所示：





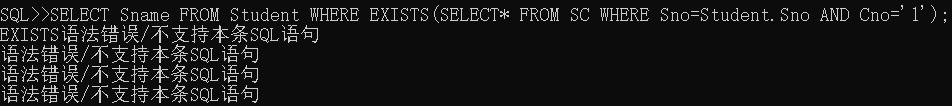
显示”语法错误/不支持本条SQL语句“。

c. 嵌套查询

运行以下SQL语句：

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

结果如下图所示：



显示”语法错误/不支持本条SQL语句“。

**2.1.8 视图**

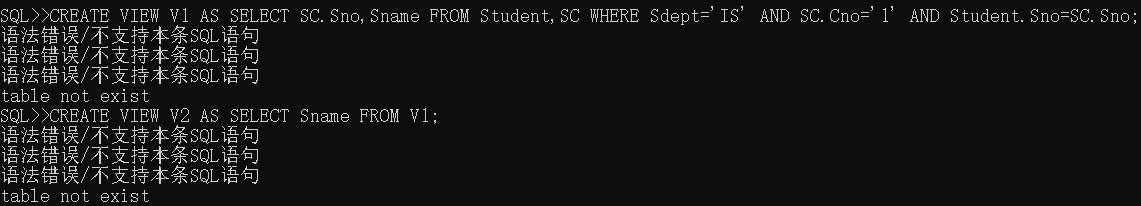
（1）定义视图

运行以下SQL语句：

CREATE VIEW V1 AS SELECT SC.Sno,Sname FROM Student,SC WHERE Sdept='IS' AND SC.Cno='1' AND Student.Sno=SC.Sno;

CREATE VIEW V2 AS SELECT Sname FROM V1;

结果如下图所示：



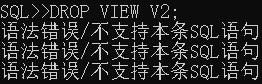
显示”语法错误/不支持本条SQL语句“。

（2）删除视图

运行以下SQL语句：

DROP VIEW V2;

结果如下图所示：



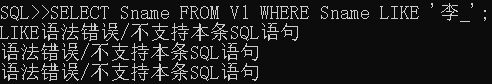
显示”语法错误/不支持本条SQL语句”。

（3）查询视图

SQL语句如下：

SELECT Sname FROM V1 WHERE Sname LIKE '李\_';

结果如下图所示：



显示”语法错误/不支持本条SQL语句”。

**2.2 PolarDB执行的SQL语句及结果**

##### 2.2.1 创建数据库

SQL语句如下：

CREATE DATABASE xjgl;

运行失败（普通用户没有权限创建数据库）。

##### 2.2.2 切换数据库

##### SQL语句如下：

USE DATABASE xjgl;

结果如下图所示：



运行失败（普通用户没有权限切换数据库）。

##### 2.2.3 定义模式/删除模式

##### 不支持此类语句。

##### 2.2.4 基本表的定义、删除与修改

（1）定义基本表

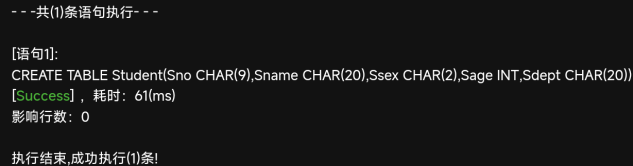
创建学生表Student、课程表Course、学生选课表SC，SQL语句如下：

CREATE TABLE Student(Sno CHAR(9),Sname CHAR(20),Ssex CHAR(2),Sage INT,Sdept CHAR(20));

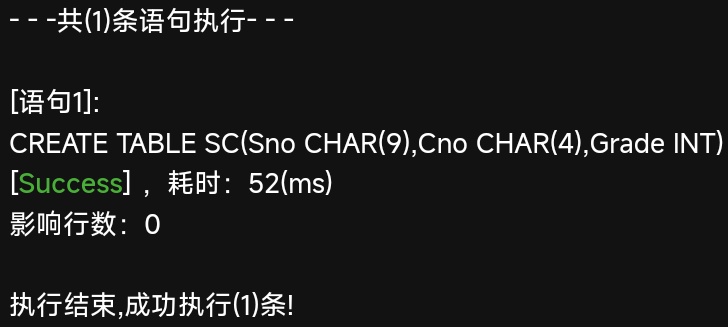
CREATE TABLE Course(Cno CHAR(4),Cname CHAR(40),Cpno CHAR(4),Ccredit INT);

CREATE TABLE SC(Sno CHAR(9),Cno CHAR(4),Grade INT);

结果如下图所示：







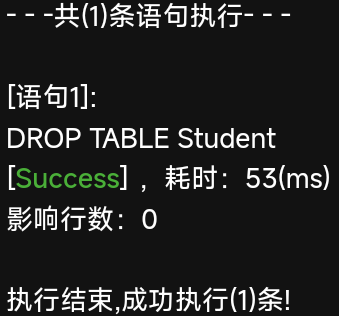
运行成功。

（2）删除基本表

SQL语句如下：

DROP TABLE Student;

结果如下图所示：



运行成功。

（3）修改基本表

SQL语句如下：

向Student表增加”出生日期“列，数据类型为日期型：

ALTER TABLE Student ADD birthdate DATE;

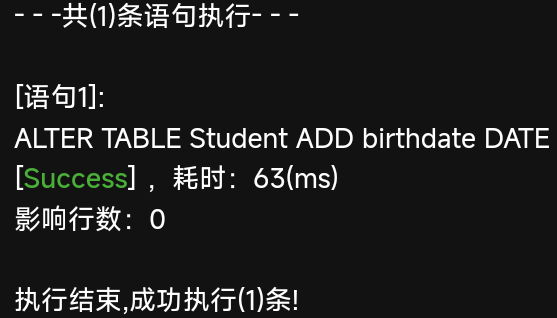
向Student表增加Sno列为主码的约束条件：

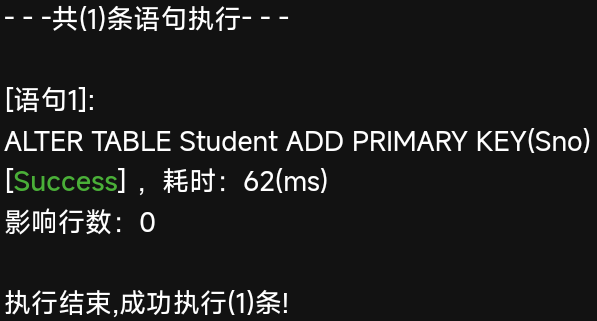
ALTER TABLE Student ADD PRIMARY KEY(Sno);

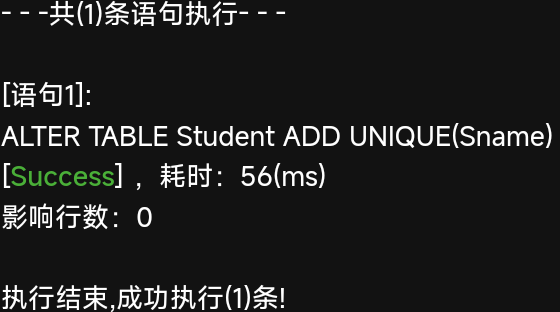
向Student表增加Sname列为唯一值的约束条件：

ALTER TABLE Student ADD UNIQUE(Sname);

结果如下图所示：







运行成功。

##### 2.2.5 完整性约束条件

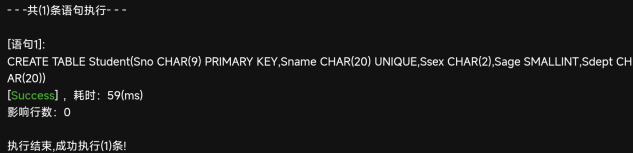
##### 重新创建带有完整性约束条件的用户数据表加，SQL语句如下：

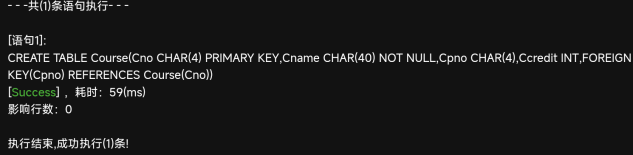
CREATE TABLE Student(Sno CHAR(9) PRIMARY KEY,Sname CHAR(20) UNIQUE,Ssex CHAR(2),Sage SMALLINT,Sdept CHAR(20));

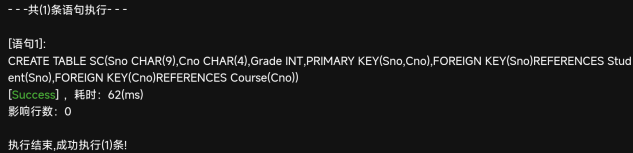
CREATE TABLE Course(Cno CHAR(4) PRIMARY KEY,Cname CHAR(40) NOT NULL,Cpno CHAR(4),Ccredit INT,FOREIGN KEY(Cpno) REFERENCES Course(Cno));

CREATE TABLE SC(Sno CHAR(9),Cno CHAR(4),Grade INT,PRIMARY KEY(Sno,Cno),FOREIGN KEY(Sno)REFERENCES Student(Sno),FOREIGN KEY(Cno)REFERENCES Course(Cno));

结果如下图所示：







运行成功。

##### 2.2.6 索引

（1）建立索引

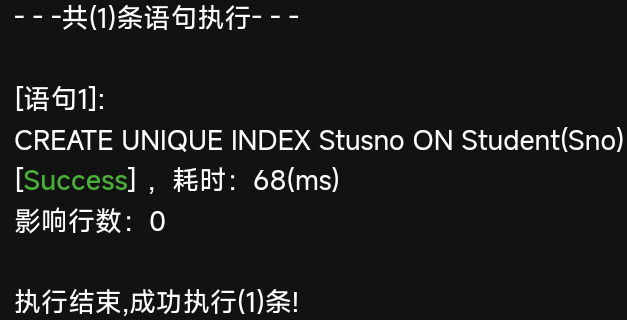
SQL语句如下：

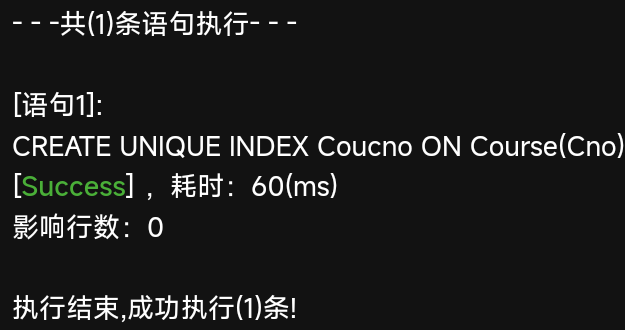
CREATE UNIQUE INDEX Stusno ON Student(Sno);

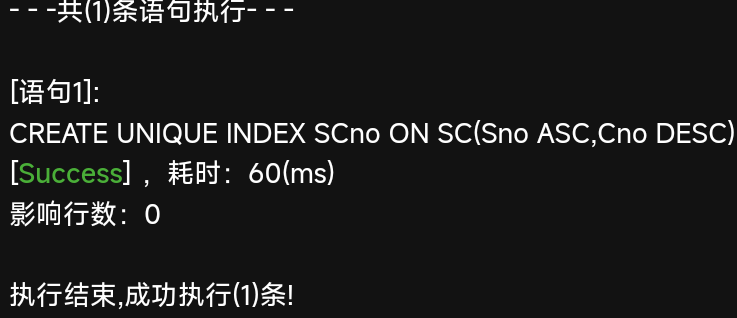
CREATE UNIQUE INDEX Coucno ON Course(Cno);

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

结果如下图所示：







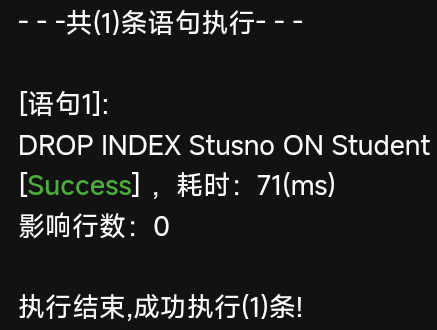
运行成功。

（2）删除索引

SQL语句如下：

DROP INDEX Stusno ON Student;

结果如下图所示：



运行成功。

**2.2.7 数据查询**

以下语句在未加约束的三个表上执行。

（1）插入数据

运行以下SQL语句：

INSERT INTO Student VALUES('201215121','李勇','男',20,'CS');

INSERT INTO Student VALUES('201215122','刘晨','女',19,'CS');

INSERT INTO Student VALUES('201215123','王敏','女',18,'MA');

INSERT INTO Student VALUES('201215124','张立','男',19,'IS');

INSERT INTO COURSE VALUES('1','数据库','5',4);

INSERT INTO COURSE VALUES('3','信息系统','1',4);

INSERT INTO COURSE VALUES('4','操作系统','6',3);

INSERT INTO COURSE VALUES('5','数据结构','7',4);

INSERT INTO COURSE VALUES('7','PASCAL语言','6',4);

INSERT INTO SC VALUES('201215121','1',92);

INSERT INTO SC VALUES('201215121','2',85);

INSERT INTO SC VALUES('201215121','3',88);

INSERT INTO SC VALUES('201215122','2',90);

INSERT INTO SC VALUES('201215122','3',80);

结果如下图所示：





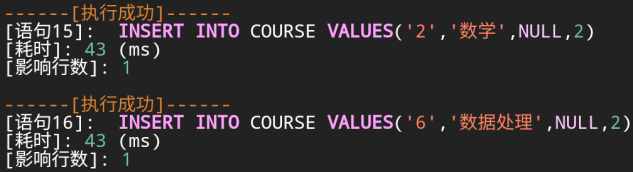
运行成功。

运行以下语句：

INSERT INTO COURSE VALUES('2','数学',NULL,2);

INSERT INTO COURSE VALUES('6','数据处理',NULL,2);

结果如下图所示：



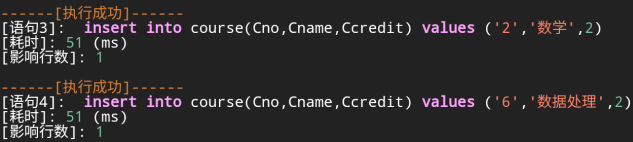
运行成功。

运行以下语句：

insert into course(Cno,Cname,Ccredit) values ('2','数学',2);

insert into course(Cno,Cname,Ccredit) values ('6','数据处理',2);

结果如下图所示：



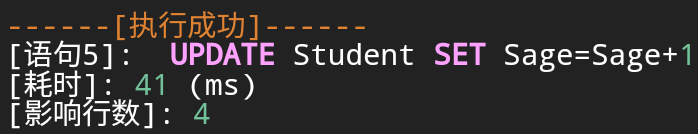
运行成功。

（2）更新数据

运行以下语句：

UPDATE Student SET Sage=Sage+1;

结果如下图所示：

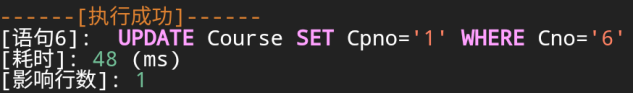


运行成功。

运行以下语句：

UPDATE Course SET Cpno='1' WHERE Cno='6';

结果如下图所示：



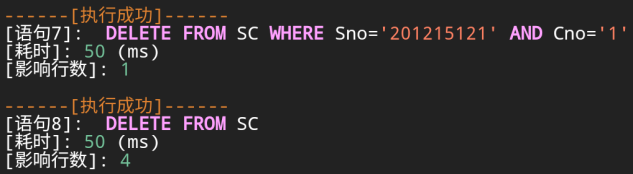
运行成功。

（3）删除数据

DELETE FROM SC WHERE Sno='201215121' AND Cno='1';

DELETE FROM SC;

结果如下图所示：

 运行成功。

（4）查询数据

1. 单表查询

运行以下SQL语句：

SELECT \* FROM Student;

SELECT \* FROM Sc;

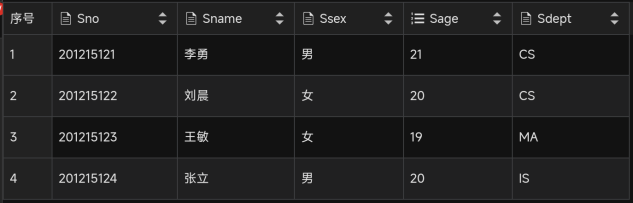
SELECT \* FROM Course;

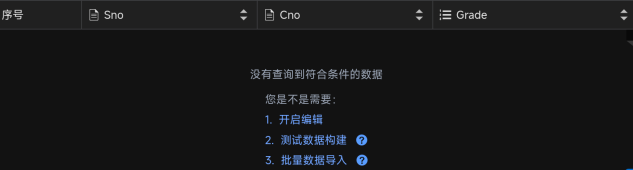
SELECT Sno,Sname,Sage FROM Student;

SELECT Sname FROM Student WHERE Sdept='cs';

SELECT Sname,Sage FROM Student WHERE Sage<20;

结果如下图所示：

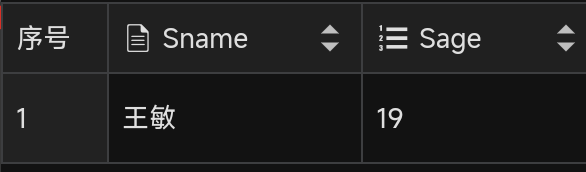












运行成功。

运行以下SQL语句：

SELECT Sname,2021-Sage FROM Student;

结果如下图所示：



运行成功。

运行以下SQL语句：

SELECT Sname,'Year of birth:',2014-Sage FROM Student;

结果如下图所示：



运行成功。

运行以下SQL语句：

SELECT DISTINCT Sno FROM SC;

SELECT ALL Sno FROM SC;

结果如下图所示：





运行成功。

运行以下SQL语句：

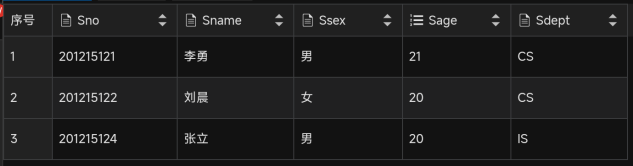
SELECT\* FROM Student WHERE Sage BETWEEN 20 AND 21;

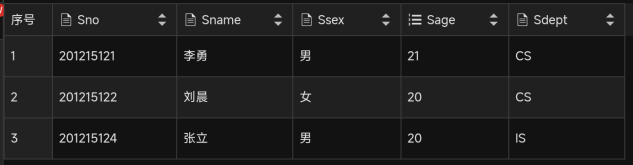
SELECT \* FROM Student WHERE Sdept IN('CS','IS');

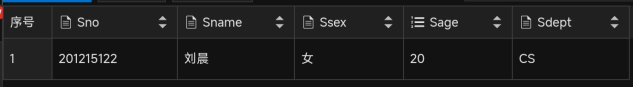
SELECT \* FROM Student WHERE Sname LIKE '刘%';

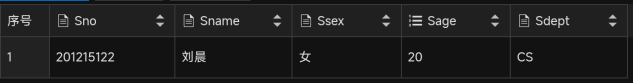
SELECT \* FROM Student WHERE Sname LIKE '刘\_';

结果如下图所示：









运行成功。

运行以下SQL语句：

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

SELECT COUNT(\*) FROM Student;

SELECT COUNT(DISTINCT Sno) FROM SC;

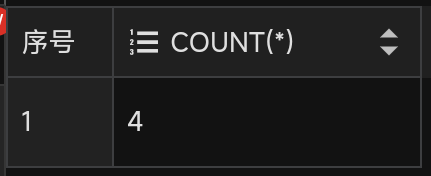
SELECT AVG(Sage) FROM Student;

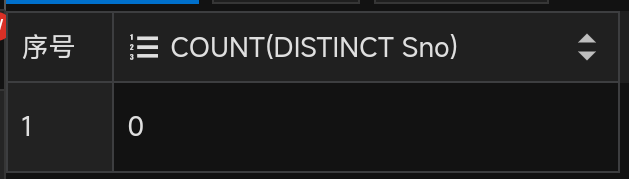
SELECT SUM(Ccredit) FROM Course;

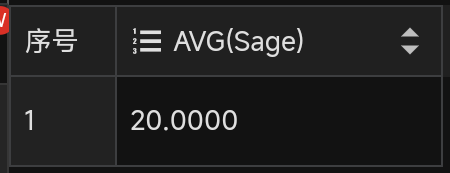
SELECT Sno FROM SC GROUP BY SNO HAVING COUNT(\*)>=3;

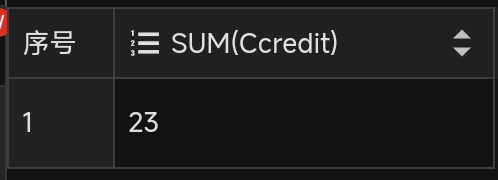
结果如下图所示：













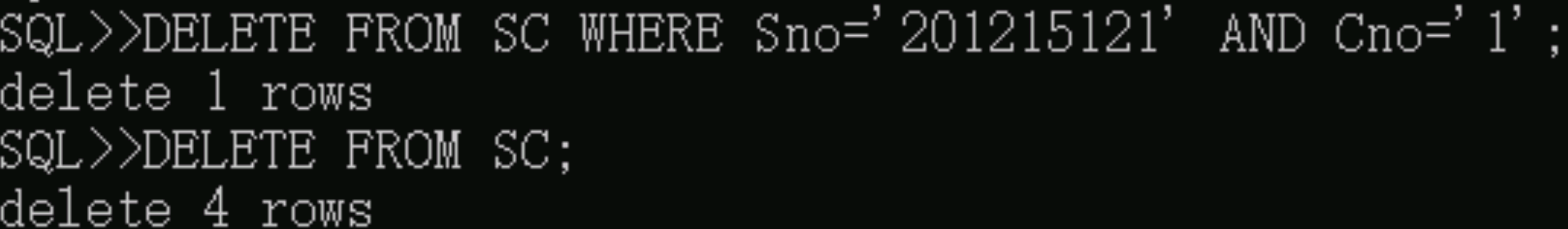
运行成功。

1. 多表查询

运行以下SQL语句：

SELECT Student.Sno,Sname FROM Student,SC WHERE Student.Sno=SC.Sno AND SC.Cno='2';

结果如下图所示：



运行成功。

运行以下SQL语句：

SELECT FIRST.Cno,SECOND.Cpno FROM Course FIRST,Course SECOND WHERE FIRST.Cpno=SECOND.Cno;

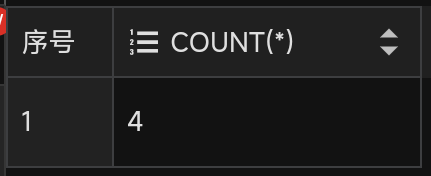
SELECT\* FROM Student LEFT OUTER JOIN SC ON(Student.Sno=SC.Sno);

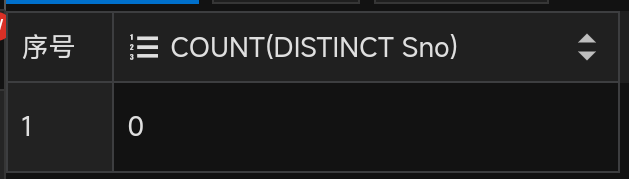
SELECT\* FROM Course RIGHT OUTER JOIN SC ON(Course.Cno=SC.Cno);

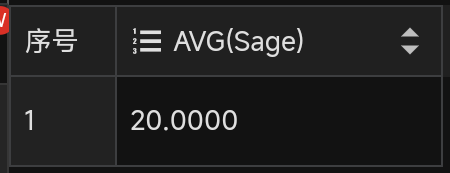
SELECT\* FROM Course JOIN SC ON(Course.Cno=SC.Cno);

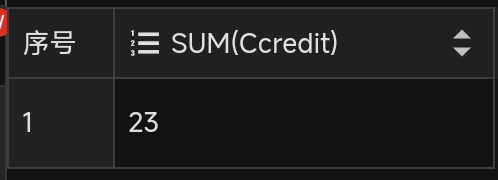
结果如下图所示：













运行成功。

1. 嵌套查询

运行以下SQL语句：

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

结果如下图所示：



运行成功。

**2.2.8 视图**

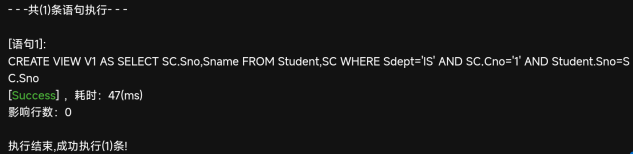
（1）定义视图

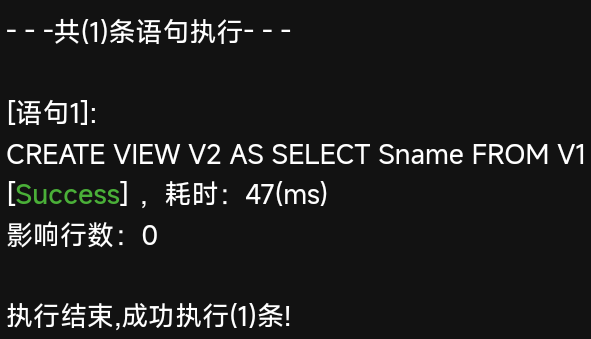
运行以下SQL语句：

CREATE VIEW V1 AS SELECT SC.Sno,Sname FROM Student,SC WHERE Sdept='IS' AND SC.Cno='1' AND Student.Sno=SC.Sno;

CREATE VIEW V2 AS SELECT Sname FROM V1;

结果如下图所示：





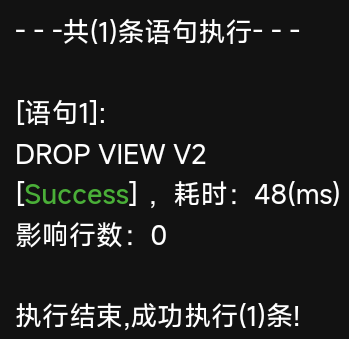
运行成功。

（2）删除视图

运行以下SQL语句：

DROP VIEW V2;

结果如下图所示：



运行成功。

（3）查询视图

SQL语句如下：

SELECT Sname FROM V1 WHERE Sname LIKE '李\_';

结果如下图所示：



运行成功。

### XDSQL数据库优化

**3.1 源码分析**

XDSQL\_for\_select数据库系统是对XDSQL数据库修改得到的，其只支持以下SQL语句：

use database [数据库名称];

show databases;

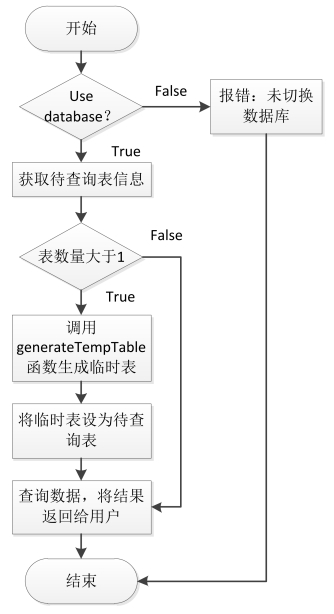
show tables;

select \* from [表名];

select \* from [表1名],[表2名];

该数据库不支持创建数据库、删除数据库、创建表、删除表、插入、删除、更新等操作，目前只支持单表查询和双表查询，不支持条件查询和嵌套查询。

在XDSQL\_for\_select源代码中，负责查询的代码位于XDSQL\_for\_select/QueryProcessor.cpp中select函数内，其功能流程图如下：



当执行多表查询时，程序首先对待查询的多张表进行笛卡尔积操作，将结果保存到临时表中，随后基于临时表进行查询操作，最后删除临时表。生成临时表的函数名为generateTempTable，其中调用tablejoin函数用于多张表的笛卡尔积操作，并将结果存入临时表。目前tablejoin只支持两张表笛卡尔积操作。本次实验的目的便是修改tablejoin函数，使其支持多张表笛卡尔积操作。

涉及到的结构体说明：

* tableNode结构体：具体定义见MetadataManager.h文件，记录数据表的元数据信息，包括表名、表内列数、一行数据的大小（单位：字节）和指向下一个tableNode结构体的指针。
* storageNode结构体：具体定义见StorageEngine.h文件，可以将其视为指向数据表内数据行的指针。

涉及到的函数说明：

void tablejoin(tableNode\* tablenode, char\* row, int index, storageNode\* sp);

负责对多张表进行笛卡尔积操作，并将结果通过sp写入临时表中。其传入参数如下：

* tablenode：数据表节点，记录待查询数据表基本信息。
* row：字节数组，作为临时数据行，负责笛卡尔积过程中数据的暂时存储。
* index：字节数组指针，负责记录下一张表数据行数据在row字节数组中存储的首地址。
* sp：临时数据表的数据行指针。

void open(storageNode\* sp, char\* tablename, int rowlength);

打开tablename数据表，并用sp指向该数据表内的数据行。其中tablename为数据表名称，rowlength为数据表内一行数据所占字节数。

void rnd\_init(storageNode\* sp);

初始化sp指针，使其指向数据表第一行数据。

bool isEOF(storageNode\* sp);

判断数据表是否读取完毕，完毕返回true，否则返回false。

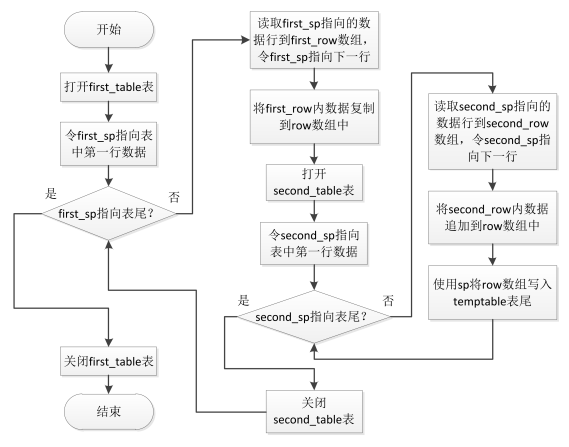
char\* rnd\_next(storageNode\* sp);

读取sp指向的数据行数据，随后令sp指向下一行数据。返回值如果为NULL，说明该行数据已被删除，不对数据进行任何操作。

void write\_row(storageNode\* sp, char\* row);

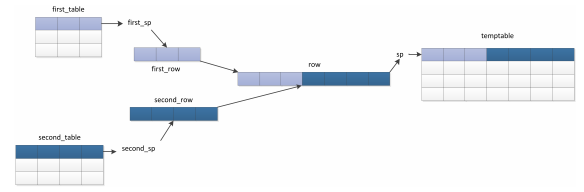
向sp指向的数据表表尾写入一行数据。row字节数组记录待写入的数据行数据。

tablejoin函数功能如下图所示：



其中first\_table和second\_table为待查询的表，temptable为临时表，是待查询表笛卡尔积后的结果表。

下图展示了临时表第一行数据生成过程：



**3.2 优化思路**

由于多表查询可以转化为多次的双表查询，而双表查询目前已经支持，故可以通过对双表查询进行递归操作来实现多表查询。具体地，每次递归操作时，先判断当前表的数量，如果正好只剩两张表，则直接进行双表查询即可；如果超过两张表，则先对当前多张表中的前两张表进行双表查询操作，再将连接后的结果表与剩余的表合并，继续递归地调用tablejoin函数进行多表查询操作。

**3.3 代码实现**

void tablejoin(tableNode\* tablenode, char\* row, int index, storageNode\* sp) {

    //sp：储存temp临时表信息的结构体

    //tablenode：第一个表的表元信息节点

    if (index == 0)

    {

        tableNode\* firstnode = tablenode;

        //第一个用户数据表的表名（带有相对路径）

        char\* firstfilename = dbnameADDtablename(usedatabase, firstnode->tablename);

        storageNode\* firstsp = createStorageNode();     //储存第一个用户数据表信息的结构体

        open(firstsp, firstfilename, firstnode->rowlength);

        rnd\_init(firstsp)

        //对第一个用户数据表中的每一个元组，都打开第二个表读取所有元组并拼接，将结果写入temp临时表

        while (!isEOF(firstsp)) {

            char\* first\_row = rnd\_next(firstsp);        //读取第一个表的一个元组

            if (first\_row == NULL) //读取到被删除数据，返回空值

                continue;

            //将第一张表一行数据放入row数组中

            memcpy\_s(row, firstnode->rowlength, first\_row, firstnode->rowlength);

            int firstindex = index + firstnode->rowlength;

            //第二个表的表元信息节点

            tableNode\* secondnode = firstnode->next;

            char\* secondfilename = dbnameADDtablename(usedatabase, secondnode -> tablename);

            storageNode\* secondsp = createStorageNode();

            open(secondsp, secondfilename, secondnode->rowlength);

            rnd\_init(secondsp);

            //遍历第二个表所有的元组，逐个与第一个表中选中的元组拼接

            while (!isEOF(secondsp)) {

                char\* second\_row = rnd\_next(secondsp);

                if (second\_row == NULL)

                    continue;

                memcpy\_s(row+firstindex, secondnode->rowlength, second\_row, secondnode -> rowlength);

                int secondindex = firstindex + secondnode->rowlength;

                //将第二张表的一行数据添加到row数组末尾

                if (secondnode->next != NULL)

                    tablejoin(secondnode, row, secondindex, sp);

                else

                    write\_row(sp, row);

            }

            close(secondsp);

            deleteStorageNode(secondsp);

        }

        /\* 清除缓存 \*/

        close(firstsp);

        deleteStorageNode(firstsp);

        return;

    }

    tableNode\* secondnode = tablenode->next;

    char\* secondfilename = dbnameADDtablename(usedatabase, secondnode->tablename);

    storageNode\* secondsp = createStorageNode();

    open(secondsp, secondfilename, secondnode->rowlength);

    rnd\_init(secondsp);

    //遍历第二个表所有的元组，逐个与第一个表中选中的元组拼接

    while (!isEOF(secondsp)) {

        char\* second\_row = rnd\_next(secondsp);

        if (second\_row == NULL)

            continue;

        memcpy\_s(row + index, secondnode->rowlength, second\_row, secondnode->rowlength);

        int secondindex = index + secondnode->rowlength;

        //将第二张表的一行数据添加到row数组末尾

        if (secondnode->next != NULL)

            tablejoin(secondnode, row, secondindex, sp);

        else

            write\_row(sp, row);

    }

    close(secondsp);

    deleteStorageNode(secondsp);

}

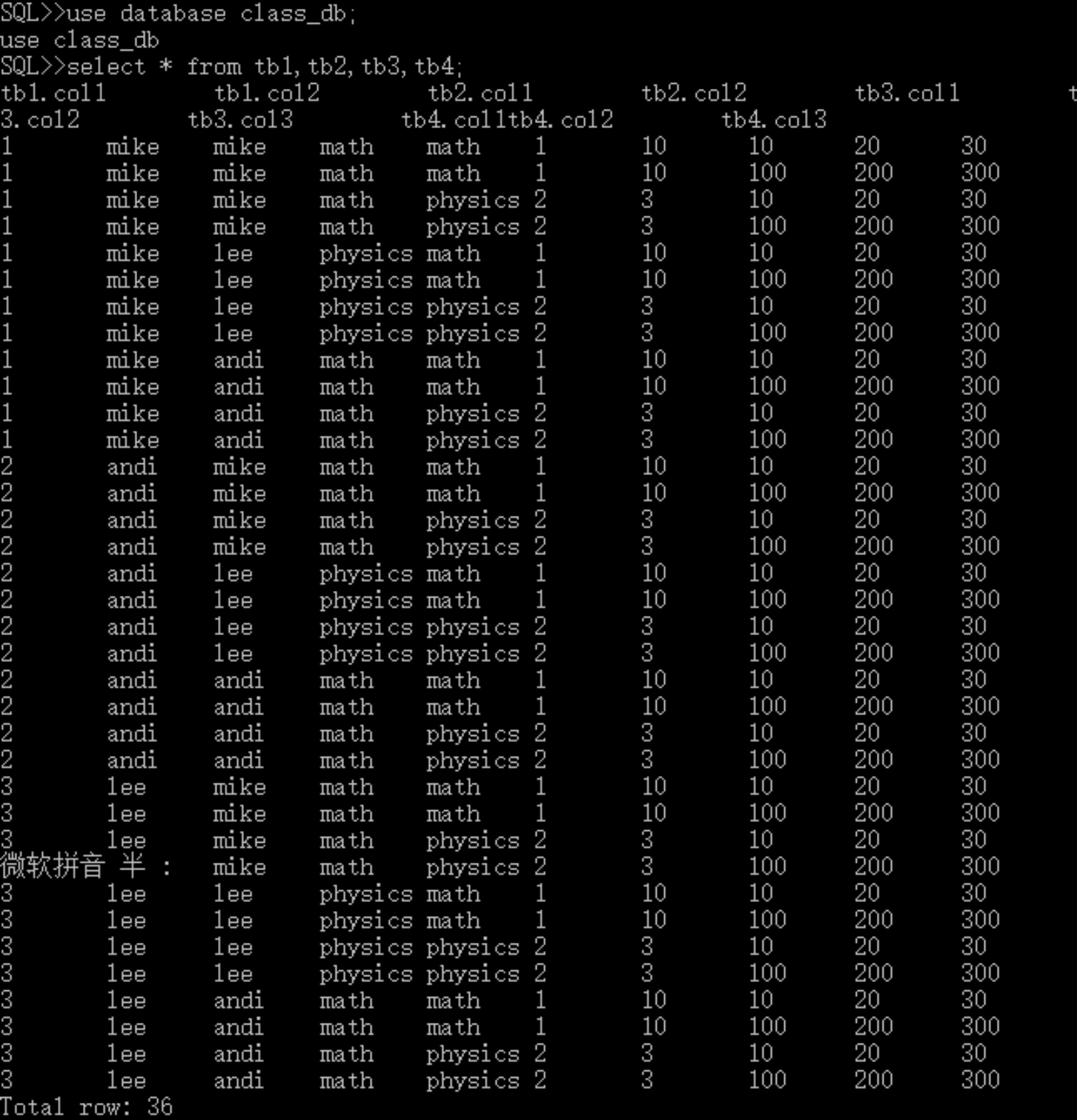
**3.4 测试结果截图**

点击运行，依次输入以下指令进行测试：

use database class\_db;

select \* from tb1,tb2,tb3,tb4;

测试结果如下图所示：



说明成功改为支持多表查询的数据库系统。