**数据库系统实验报告**

学号： 19030130448 姓名： 吕思勤

手机： 15771923571 邮箱：serena052900@qq.com

阿里云账号： qwerq110p

完成时间：2021年12月20日

# 线上实验：阿里PolarDB与XDSQL

### 1. PolarDB远程连接

实验内容

阿里PolarDB申请使用。

高级语言远程连接PolarDB。

实验目的

熟悉PolarDB for MySQL使用方式，了解阿里云dms实验平台，并学会使用Python或Java语言实现对数据库的远程连接操作。

实验步骤

1.登录阿里云dms控制台，dms控制台网址为https://dmslab.aliyun.com/

2. 开通PolarDB for MySQL数据库，开通后系统会默认生成三个数据表，分别是courses、students、student\_course表

3. 右键点击侧边栏中PolarDB for MySQL，在弹出的菜单栏中点击开通应用程序访问，获取连接信息

4. 根据连接信息提示，使用Python或Java实现对数据库的远程访访问

详细实现

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。

import pymysql

connection = pymysql.connect(host='Host',

port=Port,

user='User',

passwd='Password',

db='Database')

try:

with connection.cursor() as cursor:

# 以查询courses表为例

sql = "SELECT \* FROM `courses`"

cursor.execute(sql)

for result in cursor:

# 输出查询结果

print(result)

finally:

connection.close()

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

实验结果一張含有 文字 的圖片

自動產生的描述

### 2. XDSQL与PolarDB对比实验

实验内容

XDSQL执行的SQL语句及结果。

PolarDB执行的SQL语句及结果。

实验目的

XDSQL数据库是自主开发的一款微型数据库系统，具有简单的建库、建表、增删改查等功能，其SQL解释层采用lex和yacc实现，执行层采用C++实现，存储引擎则是使用最基础的方式实现，即每个数据库对应操作系统的一个文件夹，每个数据表对应操作系统中的一个文件。

通过对XDSQL数据库和PolarDB数据库的测试对比，了解这两款数据库所支持的SQL语句类型，体会在数据库软件中不同SQL语句实现的困难度，同时去发现XDSQL中存在的问题。

实验步骤

1.登录阿里云dms控制台，进入PolarDB for MySQL数据库，控制台网址：https://dmslab.aliyun.com/；

2. 下载XDSQL压缩包到本地电脑；

3. 解压压缩包，运行XDSQL.exe，启动XDSQL；

4. 在两个数据库内执行测试集内的SQL语句，根据运行结果去发现XDSQL的不足和不同SQL语句实现的难易程度。测试集来自教材内容，具体见下文所示。

详细实现

创建数据库

SQL语句如下：

CREATE DATABASE xjgl;

polarDB运行失败（普通用户没有权限创建数据库）。

XDSQL运行成功。

切换数据库

SQL语句如下：

USE DATABASE xjgl;

polarDB运行失败（普通用户没有权限切换数据库）。

XDSQL运行成功。

定义模式/删除模式

polarDB与XDSQL都不支持此类SQL语句。

基本表的定义、删除与修改

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);

删除基本表

删除Student表

DROP TABLE Student;

polarDB运行成功。

XDSQL运行成功。

修改基本表

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);

以上SQL语句，在polarDB运行成功。

XDSQL显示”语法错误/不支持本条SQL语句“。

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。

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));

以上SQL语句，在polarDB运行成功。

XDSQL显示”语法错误/不支持本条SQL语句“。

索引

建立索引

CREATE UNIQUE INDEX Stusno ON Student(Sno);

CREATE UNIQUE INDEX Coucno ON Course(Cno);

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

以上SQL语句，在polarDB运行成功。

XDSQL显示”语法错误/不支持本条SQL语句“。

删除索引

DROP INDEX Stusno ON Student;

以上SQL语句，在polarDB运行成功。

XDSQL显示”语法错误/不支持本条SQL语句“。

数据查询

以下语句在未加约束的三个表上执行。

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);

polarDB运行成功。

XDSQL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);

polarDB运行成功。

XDSQL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);

polarDB运行成功。

XDSQL运行成功。

更新数据

运行以下语句

UPDATE Student SET Sage=Sage+1;

polarDB运行成功。

XDSQL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';

polarDB运行成功。

XDSQL运行成功。

删除数据

运行以下SQL语句：

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

DELETE FROM SC;

polarDB运行成功。

XDSQL运行成功。

查询数据

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;

polarDB运行成功。

XDSQL运行成功。

运行以下SQL语句：

SELECT Sname,2021-Sage FROM Student;

polarDB运行成功。XDSQL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;

polarDB运行成功。

XDSQL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;

polarDB运行成功。

XDSQL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\_';

polarDB运行成功。

XDSQL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;

polarDB运行成功。

XDSQL显示”语法错误/不支持本条SQL语句”

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';

polarDB运行成功。

XDSQL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);

polarDB运行成功。

XDSQL显示”语法错误/不支持本条SQL语句”

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');

polarDB运行成功。

XDSQL显示”语法错误/不支持本条SQL语句“

视图

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;

polarDB运行成功。

XDSQL显示”语法错误/不支持本条SQL语句”

删除视图

运行以下SQL语句

DROP VIEW V2;

polarDB运行成功。

XDSQL显示”语法错误/不支持本条SQL语句”

查询视图

运行以下SQL语句

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

polarDB运行成功。

XDSQL显示”语法错误/不支持本条SQL语句”

实验结果

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

### 3. XDSQL数据库优化

实验目的

在DBMS原型数据库开发中，select语句功能实现是较为复杂的，涉及到单表查询、多表查询、条件查询和嵌套查询等功能的实现。为了提高学生动手能力，加深学生对笛卡尔积和多表查询的理解，本次实验在XDSQL数据库源代码的基础上，提供给学生一款只支持两张表连接查询的数据库系统XDSQL\_for\_select和其源代码，由学生将其改为支持多表查询的数据库系统。

实验内容

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;

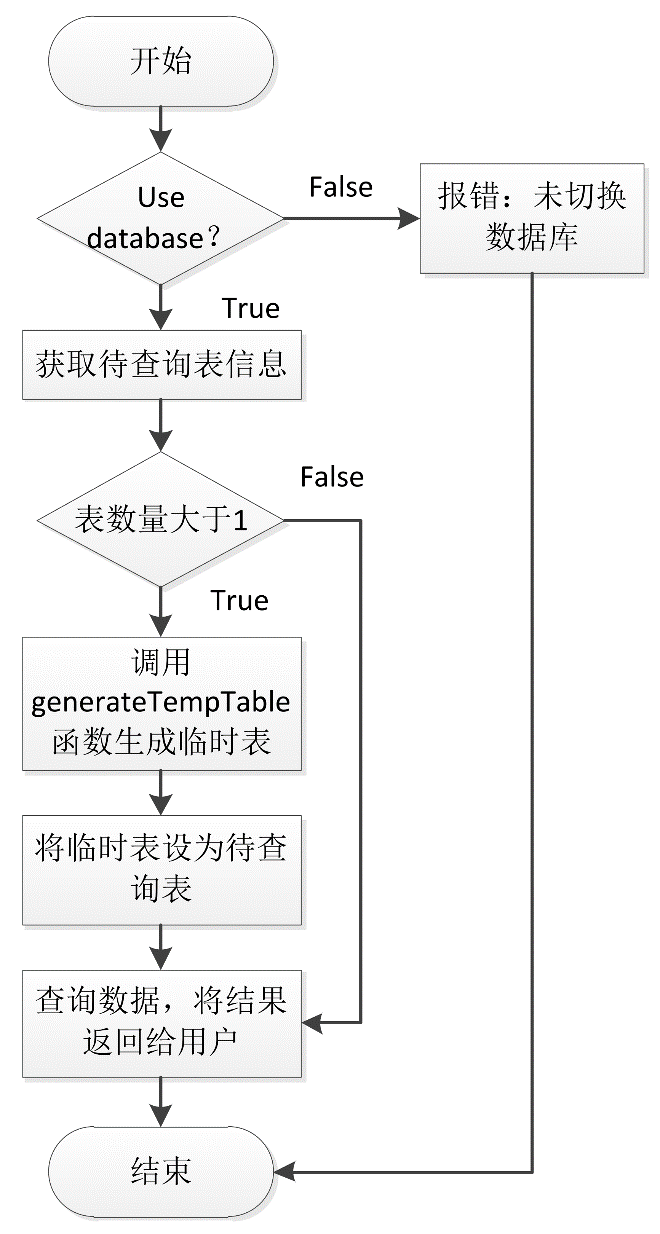
show tables;

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。

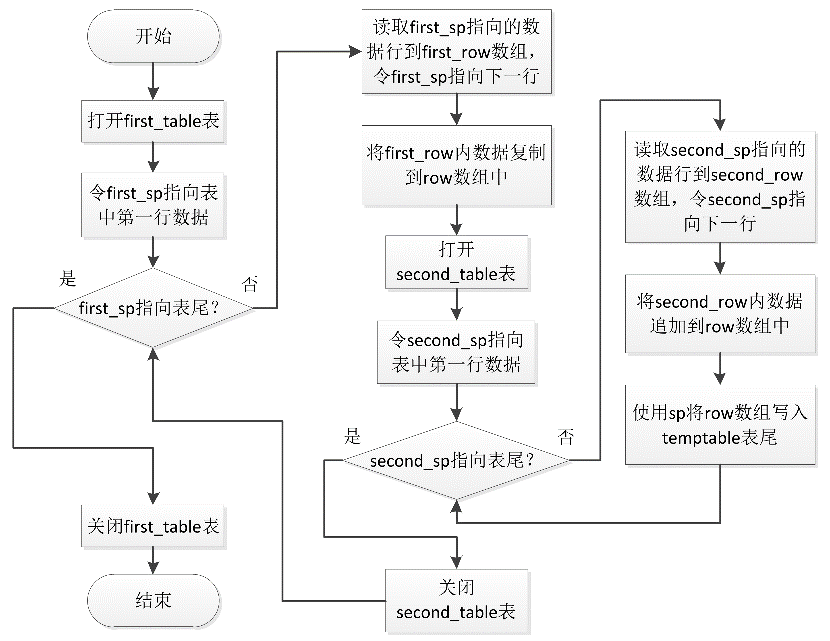
char\* rnd\_next(storageNode\* sp);

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。

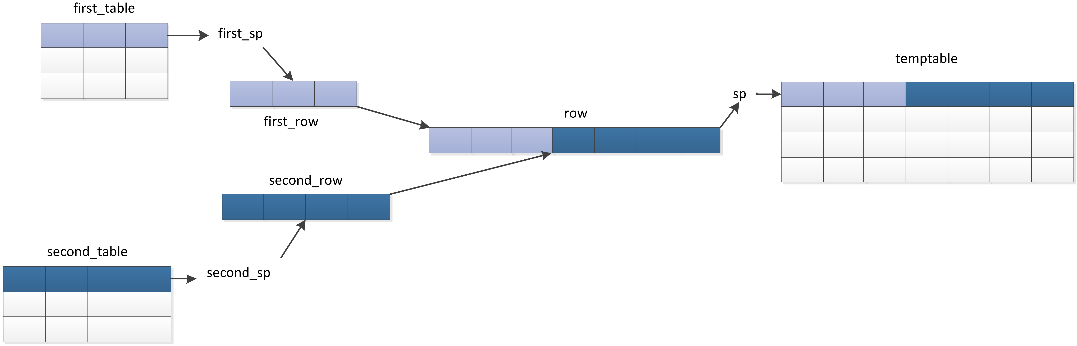
void write\_row(storageNode\* sp, char\* row);

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：

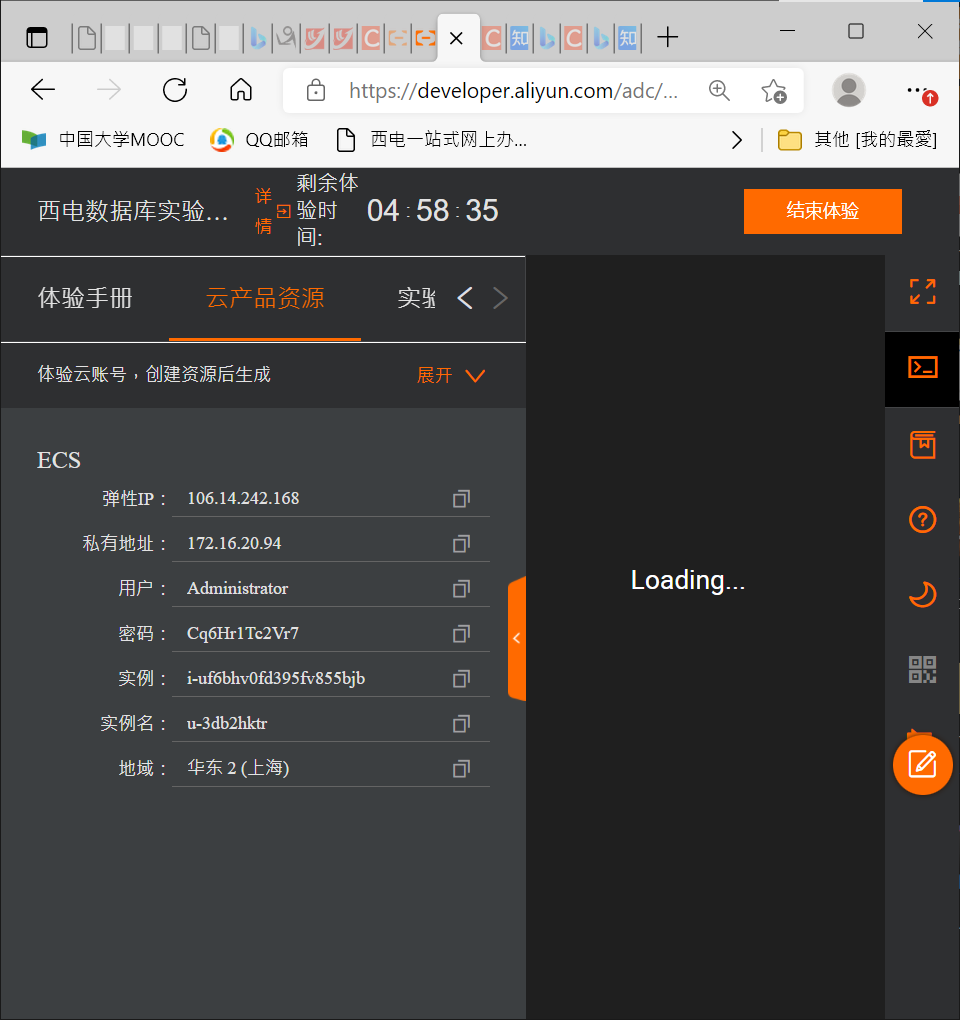


现需对tablejoin函数进行修改，使该函数具有多张表笛卡尔积功能，使newSQLdemo数据库支持多表查询。

实验步骤

本次实验在windows server 2016操作系统下进行，编译工具为VS2019，具体步骤如下：

1.申请云主机，点击创建资源，获得ECS公网IP，登录名和密码



2. 打开开始菜单，搜索远程桌面连接并运行，依次填写连接信息

3. 点击连接，登录到ECS服务器。第一次连接可能会出现连接失败，等待一两分钟后再次连接即可登录

4. 将Git-2.31.1-64-bit.exe复制到云主机（直接复制即可）

5. 安装git，安装步骤直接Next即可

6. 运行cmd，输入以下指令，将XDSQL\_for\_select数据库源代码下载到云主机

cd C:\

mkdir code

cd code

git config --global --unset http.proxy

git clone https://github.com/flytigerman/XDSQL\_for\_select.git

执行结果如图所示

一張含有 文字 的圖片

自動產生的描述

7. 将VS2019复制到云主机，安装。

8. 打开VS2019，点击右上角的文件->打开->项目，打开XDSQL\_for\_select项目（路径为C:\code\XDSQL\_for\_select）

9. 打开QueryProcessor.cpp文件，对tablejoin函数进行修改，使其支持多表连接

代码修改如下

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 /\*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); //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+firstnode->rowlength, secondnode->rowlength, second\_row, secondnode -> rowlength); //将第二张表的一行数据添加到row数组末尾 write\_row(sp, row); } close(secondsp); deleteStorageNode(secondsp); }

if (tablenode == NULL) { write\_row(sp, row); return; } else { char\* tablefilename = dbnameADDtablename(usedatabase, tablenode->tablename); storageNode\* tsp = createStorageNode(); open(tsp, tablefilename, tablenode->rowlength); rnd\_init(tsp); while (!isEOF(tsp)) { char\* t\_row = rnd\_next(tsp); if (t\_row == NULL)

continue;//读取被删除的数据，返回空值 memcpy\_s(row+index, tablenode->rowlength,t\_row, tablenode->rowlength); tablejoin(tablenode->next, row, index + tablenode->rowlength, sp); } /\* 清除缓存 \*/ close(tsp); deleteStorageNode(tsp); }

}

10. 修改完毕后，将select函数中待查询表数量大于2的判断条件注释或删除，如下所示

if (tablelist->num > 1) {

/\*if (tablelist->num > 2) {

printf("error: the number of tables is not more than 2");

return;

}\*/ //删除即可

tablenode = generateTempTable(tablelist);

}

11. 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;

实验结果

一張含有 文字 的圖片

自動產生的描述

### 4. 遇到的主要问题及解决方法

在实验一中，没有遇到什么大问题，遇到的问题主要是python的环境配置错误，通过安装所需的环境即能解决问题。实验二也是比较简单的实验，按着阿里云上的操作手册上的步骤即可成功完成XDSQL与PolarDB的对比实验。实验三相较于前两个实验困难许多，要对数据库源码进行修改，首先要能看懂并理解XDSQL里的代码，在理解代码这部份上花费了我不少时间去研究、阅读，修改代码时使用的是递归的方式实现多表查询，一开始尝试修改的代码因为函数调用顺序错误等，无法成功实现递归方式，后来经过一连串的查错调试，成功解决问题。

### 5. 总结

通过这三次的实验，让我更加理解数据库的工作方式及原理，以及如何在高级语言(Python)上远程连接数据库，也比较了XDSQL与PolarDB两者之间不同数据库之间的差异性，最后通过对XDSQL数据库源码的重写实现多表连接的查询方式，在实验过程中，让我对笛卡儿积了解的更加透彻，将书本上的理论实际的应用到计算机上并加以实现，虽然过程中也有遇到一些报错等难题，但是通过仔细的阅读代码及查找资料后逐渐地将这些难题一一克服，更是让我收获到满满的成就感，相信拖过这次的锻炼对我日后的编码能力能有所提升。