密级: 公 开

信息工程大学

操作系统实验报告

Linux初识

单 位： 四院

队 别： 十一大队一队

学 号： 4042017022

姓 名： 毛颖

学科专业： 网络工程

日 期： 2019 年 09 月 08 日

说 明

一、本实验的目的是帮助学员熟悉Linux下基本命令行使用、gcc编译器的使用等操作。

二、 要求学员使用C语言进行代码实现下面的题目。

**数据说明：**

总共包括3个数据文件：customer.txt, lineitem.txt和orders.txt。分别行数为100，1000和4000，列和列之间使用|来分隔。所有文件均为文本文件。

每一个顾客在customer表中有一个唯一的顾客编号c\_custkey，和对应的某个市场部门c\_mkgsegment。每个顾客可能会有任意多个历史订单，每个订单在orders表中有一行，它有一个唯一的订单号o\_orderkey和下单顾客编号o\_custkey和一个下单时间c\_orderdate。每个订单又由多件商品组成，每件商品在lineitem中有一行，记录了它所属的订单号l\_orderkey、价格l\_extendedprice及发货时间l\_shipdate。

目标是找出每个订单延迟发货的商品的总销售额，以供销售方有针对性地分析优化。如下图所示，我们将该问题抽象为以下的一个计算问题：每张表看作一个multi-set，对customer、orders和lineitem三张表建立笛卡尔集合，得到一个mnl行的一个集合（其中m,n,l分别是三张表的行数），然后对该集合按以下条件做过滤：

c\_mktsegment = ？ and c\_custkey=o\_custkey and o\_orderkey=l\_orderkey and o\_orderdate < ? and l\_shipdate > ? ：其中？是动态变参

对满足过滤条件的结果，按照l\_orderkey列的值进行分组，并对l\_orderkey相同的记录的l\_extendedprice做求和，最后结果按照l\_extendedprice求和值排序，并返回topn的结果，其中topn值也是动态变参。

该计算过程，也可以使用以下SQL来描述：

select

l\_orderkey,

o\_orderdate,

sum(l\_extendedprice) as revenue

from

customer,

orders,

lineitem

where

c\_mktsegment = ？

and l\_orderkey = o\_orderkey

and c\_custkey = o\_custkey

and o\_orderdate < ？

and l\_shipdate > ？

group by

l\_orderkey,

o\_orderdate

order by

revenue desc

LIMIT ？

**提交要求：**

需要提供5个文件（夹）

Readme.txt：编译和运行方式

Compile.sh：编译脚本

Run.sh: 执行脚本

Sourcecode 文件夹：包括所有source code和配置文件

可执行文件：可执行文件

**程序执行方式：**

选手需要提前将3个数据文件放到run.sh 相同目录，提交的程序将会以如下格式的命令运行、进行评测。

./run.sh customer.txt orders.txt lineitem.txt n n4个参数

第四个参数表示总共计算的次数

第5~8个参数为第一次计算时的4个参数

第9~12个参数为第二次计算时的4个参数

…

第1+4n~4+4\*n 个参数为第n次计算时的4个参数

四个参数分别对应以下的条件值：

c\_mktsegment = ？

o\_orderdate < ？

l\_shipdate > ？

LIMIT ？

执行示例：

示例1：

./run.sh customer.txt orders.txt lineitem.txt 1 BUILDING 1995-03-29 1995-03-27 5

示例2：

./run.sh customer.txt orders.txt lineitem.txt 3 BUILDING 1995-03-29 1995-03-27 5 BUILDING 1995-02-29 1995-04-27 10 BUILDING 1995-03-28 1995-04-27 2

程序输出：所有结果按行输出，多列结果使用|分割

输出示例：

示例1：

l\_orderkey|o\_orderdate|revenue

249739810|1995-02-28|513890.13

180639074|1995-03-20|502044.66

202071367|1995-03-12|499760.62

31617348|1995-03-27|498949.38

461984355|1995-03-13|491959.42

示例2：

l\_orderkey|o\_orderdate|revenue

249739810|1995-02-28|513890.13

180639074|1995-03-20|502044.66

202071367|1995-03-12|499760.62

31617348|1995-03-27|498949.38

461984355|1995-03-13|491959.42

l\_orderkey|o\_orderdate|revenue

594317283|1995-02-11|457168.01

328093060|1995-02-03|453863.74

322499142|1995-02-25|427615.4

319693698|1995-02-05|425802.97

323689092|1995-02-23|416615.42

128919045|1995-02-28|414996.73

540656743|1995-02-23|411107.69

311469825|1995-02-21|402946.11

164796770|1995-02-27|401863.1

131043138|1995-02-20|399884.41

l\_orderkey|o\_orderdate|revenue

31617348|1995-03-27|498949.38

461984355|1995-03-13|491959.42

|  |  |  |
| --- | --- | --- |
| 一、基本情况 | | |
| 实验环境 | | Lenovo IdeaPad，Ubuntu |
| 二、实验过程  1、设计结构体，实现类似数据库建表功能  typedef struct//定义顾客结构体，包括编号和对应的某个部门信息  {  int c\_custkey;  char c\_mkgsegment[20];  }customer;  typedef struct//定义订单结构体，包括订单号、下单顾客编号、下单时间  {  int o\_orderkey;  int o\_custkey;  char o\_orderdate[10];  }orders;  typedef struct//定义商品信息结构体，包括订单号、额外价格、发货日期  {  int l\_orderkey;  double l\_extendedprice;  char l\_shipdate[10];  }lineitem;  typedef struct//定义查询结果结构体，包括订单号、订货日期、额外价格  {  int l\_orderkey;  char o\_orderdate[10];  double l\_extendedprice;  }select\_result;  2、编写函数，实现类似数据库录入数据功能  customer \* list\_customer\_txt() //读取customer.txt内容  {  FILE \* fp;  customer \*a=NULL;  a = (customer \*)malloc(100\*sizeof(customer));//行数100  int i=0;  char b;  fp = fopen("home\\xiang\\桌面\\操作系统\\customer.txt","r");//引用文件所在位置  if(NULL==fp)  {  printf("Cannot open customer.txt!\n");  return NULL;  }  while(!feof(fp))  {  fscanf(fp,"%d%c%s",&a[i].c\_custkey,&b,a[i].c\_mkgsegment);  i++;  }  fclose(fp);  return a;  }  lineitem \* list\_lineitem\_txt()//读取lineitem.txt内容  {  FILE \* fp;  lineitem \* l=NULL;  l = (lineitem \*)malloc(1000\*sizeof(lineitem));//行数1000  int i=0;  char b,c;  fp = fopen("home\\xiang\\桌面\\操作系统\\lineitem.txt","r");  if(fp==NULL)  {  printf("Cannot open lineitem.txt!\n");  return NULL;  }  while(!feof(fp))  {  fscanf(fp,"%d%c%lf%c%s",&l[i].l\_orderkey,&c,&l[i].l\_extendedprice,&b,l[i].l\_shipdate);  i++;  }  fclose(fp);  return l;  }  orders \* list\_orders\_txt()//读取orders.txt内容  {  int i =0;  orders \* a=NULL;  a = (orders \* )malloc(4000\*sizeof(orders));//行数4000  char b,c;  long long d;  FILE \*fp;  fp = fopen("home\\xiang\\桌面\\操作系统\\orders.txt","r");  if(fp == NULL)  {  printf("Cannot open orders.txt!\n");  return NULL;  }  while(!feof(fp))  {  fscanf(fp,"%d%c%lld%c%s",&a[i].o\_orderkey,&b,&d,&c,a[i].o\_orderdate);  a[i].o\_custkey=d%100;  i++;  }  fclose(fp);  return a;  }  //编写用于读取三个表的数据记录的函数，相当于实现数据库录入数据的功能。调取文件本来可以写成单独一个函数，通用，但是实际用户操作是比较麻烦，由于行数等属性差异，使得需要键入的数据过多。  3、属性连接，实现类似数据库表的连接功能  for(i=0;i<100;i++)  {  for(j=0;j<4000;j++)  {  for(k=0;k<1000;k++)  if(cus[i].c\_custkey==ord[j].o\_custkey&&ord[j].o\_orderkey==item[k].l\_orderkey&&(strcmp(ord[j].o\_orderdate,order\_date)<0)&&(strcmp(item[k].l\_shipdate,ship\_date)>0))  {  printf("%d,%s,%lf\n",item[k].l\_orderkey,ord[j].o\_orderdate,item[k].l\_extendedprice);  result1[l].l\_orderkey=item[k].l\_orderkey;  strcpy(result1[l].o\_orderdate,ord[j].o\_orderdate);  result1[l].l\_extendedprice=item[k].l\_extendedprice;  l++;  }  }  }  4、主函数串联函数功能  customer \* cus = NULL;  orders \* ord = NULL;  lineitem \* item = NULL;  //置空  cus = read\_customer\_txt();  ord = read\_orders\_txt();  item = read\_lineitem\_txt();  //赋值  5、在Ubuntu上运行，创建可执行文件  调试出错  调试如下 | | |
| 1. 创建.sh文件   键入 gcc -std=c99 a.out 操作系统.c  出现错误 段错误（核心已转储）  目前找不到方法解决。 | | |
| 三、总结部分  描述通过实验课学到的知识和建议。  简单了解了Ubuntu的使用方法，复习了C语言的基本基本使用，对于通过创建结构体实现类似数据库建表的功能有了一定的了解和掌握，并且对于外部文件调用的函数编写做了巩固。  实验过程中未按照要求使用shell命令行，而是先在Windows下用C语言实现了实验要求的功能，然后在Ubuntu下使用终端调试。主要问题是%s的使用，在Ubuntu的环境下读取数据时不需地址符&。其次，三个txt文件在Ubuntu环境下无法读取。  程序本身未能实现把调取的数据进行排序的功能，使用递归或者其他排序方法应该基本能实现实验要求的排序功能，但是，对调取的数据的形式和排序手段，我暂时没有考虑清楚。此功能未实现。  总的来说，实验上手有点慢，比较关键的属性连接的部分也是参考了其他同学的程序进行编写。这个实验在数据库中实现很容易，但是等到自己编写程序加以实现的时候，就比较困难了。自身编程能力不够熟练，需要后期继续巩固，另一方面可能也是因为很久没有练习，加上对外部文件处理这部分内容不熟悉，实验实现上浪费了很多不必要的时间。下次改进。  PS：报告本来是在Ubuntu上写的，保存的是.obt文件，在Windows环境下不兼容，无法打开，有些关键步骤的截图丢失。 | | |
|  | | |