计算机科学与工程学院 实验报告

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **实验课程名称** | | **数据库原理** | | | **实验总成绩** |  |
| **专业** | **计算机科学与技术** | | **班级** | **计算机1606** | **指导教师签字** |  |
| **学号** | **20164625** | | **姓名** | **戚子强** | **实验报告批改时间** |  |
| **实验报告分项成绩**   |  |  |  | | --- | --- | --- | | **序号** | **实验项目** | **成绩** | | **1** | 数据库管理系统安装与数据库定义 |  | | **2** | SQL的数据操作实验 |  | | **3** | 数据库的调优、安全性和完整性 |  | | **4** | 数据库的事务管理与并发控制 |  | | | | | | | |
|  | | | | | | |

# 实验1: 数据库管理系统安装与数据库定义

实验报告1.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告1.1 | | | | |
| 题目：数据库原理实验环境安装 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.5.10 |
| 1． 数据库管理系统安装  数据库原理实验所使用的软件环境（可自行选择在自己机器上安装）   |  |  | | --- | --- | | 操作系统： | Windows 10 1809 | | 数据库管理系统： | PostgreSQL | | 数据库管理系统版本： | 10.7 |   2． 实验环境搭建中遇到了什么问题？如何解决的？  先前只安装了PostgreSQL，自带了pgAdmin4，在安装目录/ pgAdmin 4/bin。或者再安装独立的pgAdmin也是可以的。  3．数据库管理系统运行的操作系统环境。查找资料，了解各数据库管理系统对操作系统平台的支持情况，完成下表。   |  |  |  |  |  | | --- | --- | --- | --- | --- | | RDBMS | 最新版本号 | Windows | Linux（centos/ubuntu） | Mac OS | | Oracle | 19c | √ | √ | √ | | SQL Server | 2017 | √ | √ | √ | | MYSQL | 8.0.16 | √ | √ | √ | | PostgreSQL | 11.3 | √ | √ | √ | | DB2 | 11.1 | √ | √ | √ |   4\*. 了解各数据库管理系统的客户机服务器机制，完成下表：   |  |  |  | | --- | --- | --- | | RDBMS | 客户端软件名称 | 服务器端进程名 | | Oracle | win32\_版本号gRx\_client | serverprocess | | SQL Server | SQL Server 版本号 Rx | sqlservr | | MYSQL | MySQL 版本号 | mysqld | | PostgreSQL | pgAdmin版本号 | postgresql-大版本号 |   5\*. 查询相关资料，说明PostgreSQL与MYSQL的主要差异？（选做）   * mysql的各种text字段有不同的限制，需要手动区分small text,middle text, large text； * pg没有该限制，可以支持text的各种大小； * 按照SQL标准，null判断只能用is null,不能用 ==null； * pg可以设置transform\_null\_equals 把 = null 翻译成 is null 避免踩坑； * MySQL 的事务隔离级别 repeatable read 并不能阻止常见的并发更新, 得加锁才可以, 但悲观锁会影响性能, 手动实现乐观锁又复杂. 而 Pg 的列里有隐藏的乐观锁 version 字段, 默认的 repeatable read 级别就能保证并发更新的正确性, 并且又有乐观锁的性能. 附带一个各数据库对隔离级别的行为差异比较调查； * MySQL 不支持多个表从同一个序列中取 id, 而 Pg 可以； * MySQL 不支持 OVER 子句, 而 Pg 支持. OVER 子句能简单的解决 “每组取 top 5” 的这类问题.几乎任何数据库的子查询 (subquery) 性能都比 MySQL 好； * pg它可以存储 array 和 json, 可以在 array 和 json 上建索引, 甚至还能用表达式索引. 为了实现文档数据库的功能, 设计了 jsonb 的存储结构. 有人会说为什么不用 Mongodb 的 BSON 呢? Pg 的开发团队曾经考虑过, 但是他们看到 BSON 把 [“a”, “b”, “c”] 存成 {0: “a”, 1: “b”, 2: “c”} 的时候就决定要重新做一个 jsonb 了… 现在 jsonb 的性能已经优于 BSON； * 任何系统都有它的性能极限，在高并发读写，负载逼近极限下，PG的性能指标仍可以维持双曲线甚至对数曲线，到顶峰之后不再下降，而 MySQL 明显出现一个波峰后下滑； * pg对json支持比较好，还有fdw功能； * PG有极其强悍的 SQL 编程能力（9.x 图灵完备，支持递归），有非常丰富的统计函数和统计语法支持，比如分析函数（ORACLE的叫法，PG里叫window函数），还可以用多种语言来写存储过程，对于R的支持也很好。这一点上MYSQL就差的很远，很多分析功能都不支持； * PG 的有多种集群架构可以选择，plproxy 可以支持语句级的镜像或分片，slony 可以进行字段级的同步设置，standby 可以构建WAL文件级或流式的读写分离集群，同步频率和集群策略调整方便，操作非常简单； * mysql的innodb引擎，可以充分优化利用系统所有内存，超大内存下PG对内存使用的不那么充分； * MySQL的复制可以用多级从库，但是在9.2之前，PGSQL不能用从库带从库。 | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  第一次使用PostgreSQL，比较陌生，但是根据实验书和网上的经验，还是很顺利地完成了这次试验。  2 实验自我评价。  实验过程比较顺利，了解了当前主流关系数据库系统、RDBMS的系统框架，掌握了主流DBMS的安装。 | | | | |

实验报告1.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告1.2 | | | | |
| 题目：数据库定义与基本操作 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.05.10 |
| 1. 请将ER图转化为用关系数据模型描述的关系模式。   student(sno,sname,sage,sdept)  sc(sno,cno,grade)  course(cno,cname,cperiod,ccredit,cpno)   1. 创建学生选课数据库Enrolled   使用实验1.1中方法创建用户user1，并用该用户创建数据库“Enrolled”。 可以使用SQL语句和图形界面  1）方法1：使用SQL语句  点击菜单中的Tool->Query Tool，输入以下SQL语句并执行  CREATE DATABASE "Enrolled"  WITH  OWNER = user1  ENCODING = 'UTF8'  CONNECTION LIMIT = -1;  2）方法2：使用图形界面  右键点击左侧“database”目录，选择其中create->database，出现下图界面，在General选项卡中填写数据库名称（“database”）、用户（“Owner”），在Definition选项卡中可以定义数据库所使用的字符编码方式（“Encoding”）、表空间（“tablespace”），允许的连接数量“connection limit”等设置，最后点击“save”创建。创建后可以在左侧列表的“database”目录中查看数据库“Enrolled”。     1. 定义数据库基本表   1）关系基本表定义语句  在 PostgreSQL中创建关系表同样可以使用SQL和图形界面两种方式，这里我们介绍使用SQL语句方法。右键点击“Enrolled”数据库，选择其中Query Tool打开查询界面。  学生关系表：  CREATE TABLE Student (  Sno VARCHAR (20),  Sname VARCHAR(30),  Ssex VARCHAR (2),  Sage INTEGER,  Sdept VARCHAR (30),  PRIMARY KEY (Sno))  课程关系表：  CREATE TABLE Course (  Cno VARCHAR (10) not null,  Cname VARCHAR (30),  Chour INTEGER,  Credit VARCHAR (30),  Cpno VARCHAR (10)  )  执行结果如何？  成功创建了两个关系表：  2)实体完整性约束  i 具有主键的关系表  insert into Student values('95020', '陈东', 'M', 18,'CS')  insert into Student values('95020', '陈东', 'M', 18,'CS')  插入以上两条记录数据，依次执行上面的SQL语句，执行结果如何？为什么？  第一次成功插入：  第二次插入失败：  原因是第二次与第一次插入时主键相同，不能重复插入。  ii 未声明主键的关系表  insert into Course values('c1', '数据库原理', 40,2,null)  insert into Course values('c1', '数据库原理', 40,2,null)  插入以上两条记录数据，依次执行上面的SQL语句，次执行结果如何？为什么？  两次都成功插入：  因为这两句都没有主键约束。  3）定义完整性约束  在已建立的关系模式之上（已插入一些数据）建立主键约束，参照约束和用户定义的约束（要求学生年龄不小于14岁，不大于35岁），如果约束不能建立，请分析原因，修改后建立上述约束。  i 定义主键约束  执行以下语句，返回结果如何？为什么？  alter table Course add constraint pk\_courses primary key(Cno)  错误：键值重复  delete from Course  where Cno='c1'  如何可以解决定义主键的问题？  delete from Course  where Cno='c1'  约束建立后，请试着利用如下语句插入重复元组，结果如何？为什么？  insert into Course values('c1', '数据库原理', 40,2, null)  insert into Course values('c1', '数据库原理', 40,2, null)  第一次成功，第二次失败：  M%6CG939]%[HE_]6KM{J_%I  FGHXR%]%OQN(0M(ED5`]M67  因为有了键值约束。  ii 创建用户自定义约束  执行以下语句  alter table Student add constraint u\_Student check (Sage between 14 and 35)  图片包含 屏幕截图  描述已自动生成  执行以下语句，结果如何？为什么？  insert into Student values('95021', '王二', 'M', 12,'IS')  插入失败，因为违反了自定义约束条件：  4）关系模式定义语句联系  编写创建选课关系表SC（Sno,Cno,Grade），要求添加主键和外键，相关SQL语句如何编写？  create table SC (  Sno VARCHAR (20),  Cno VARCHAR(10),  Grade integer,  primary key (Sno, Cno),  FOREIGN KEY (Cno) REFERENCES Course (Cno),  FOREIGN KEY (Sno) REFERENCES Student (Sno))  图片包含 屏幕截图  描述已自动生成 | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  难点主要在不熟悉PostgreSQL的一些操作，之前习惯了MySQL，大部分操作是一样的，有很少的一部分不同，另外就是对图形化操作很不习惯。  2 实验自我评价。  在本次实验中，熟悉了PostgreSQL的图形化和命令行操作，整个过程比较顺利。 | | | | |

实验报告1.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告1.3 | | | | |
| 题目：TPC-H数据库创建于数据导入 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.5.10 |
| 1创建TPC-H数据库  参考步骤：  （1）在PostgreSQL中创建数据库TPC  使用具有创建数据库权限的用户登录，创建数据库  CREATE DATABASE TPC；  （2）在数据库TPC中创建schema  创建模式TPCH  CREATE SCHEMA TPCH；  注：schema的作用：   * 方便管理多个用户共享一个数据库,但是又可以互相独立. * 方便管理众多对象,更有逻辑性 * 方便兼容某些第三方应用程序,创建对象时是有schema的   比如要设计一个复杂系统.由众多模块构成,有时候模块间又需要有独立性.各模块存放单独的数据库显然是不合适的。这时候使用schema来分类各模块间的对象,再对用户进行适当的权限控制.这样逻辑也非常清晰。  2创建TPC-H基准测试的关系表  基于TPC-H数据模式，结合以下表结构信息，编写建表SQL语句，创建关系表  **(1)** 零件**PART Table**   |  |  |  | | --- | --- | --- | | Column Name | Data Type | Comment | | P\_PARTKEY | identifier | SF\*200,000 | | P\_NAME | variable text, size 55 |  | | P\_MFGR | fixed text, size 25 |  | | P\_BRAND | fixed text, size 10 |  | | P\_TYPE | variable text, size 25 |  | | P\_SIZE | integer |  | | P\_CONTAINER | fixed text, size 10 |  | | P\_RETAILPRICE | decimal |  | | P\_COMMENT | variable text, size 23 |  | | PRIMARY KEY: P\_PARTKEY | | |   **(2)** 供应商**SUPPLIER Table Layout**   |  |  |  | | --- | --- | --- | | Column Name | Data Type | Comment | | S\_SUPPKEY | identifier | SF\*10,000 | | S\_NAME | fixed text, size 25 |  | | S\_ADDRESS | variable text, size 40 |  | | S\_NATIONKEY | Identifier | Foreign Key to N\_NATIONKEY | | S\_PHONE | fixed text, size 15 |  | | S\_ACCTBAL | decimal |  | | S\_COMMENT | variable text, size 101 |  | | PRIMARY KEY: S\_SUPPKEY | | |   **(3) 零件供应PARTSUPP Table Layout**   |  |  |  | | --- | --- | --- | | Column Name | Data Type | Comment | | PS\_PARTKEY | identifier | Foreign Key to P\_PARTKEY | | PS\_SUPPKEY | Identifier | Foreign Key to S\_SUPPKEY | | PS\_AVAILQTY | integer |  | | PS\_SUPPLYCOST | Decimal |  | | PS\_COMMENT | variable text, size 199 |  | | PRIMARY KEY: PS\_PARTKEY，PS\_SUPPKEY | | |   **(4) 客户CUSTOMER Table Layout**   |  |  |  | | --- | --- | --- | | Column Name | Data Type | Comment | | C\_CUSTKEY | identifier | SF\*150,000 | | C\_NAME | variable text, size 25 |  | | C\_ADDRESS | variable text, size 40 |  | | C\_NATIONKEY | Identifier | Foreign Key to N\_NATIONKEY | | C\_PHONE | fixed text, size 15 |  | | C\_ACCTBAL | decimal |  | | C\_MKTSEGMENT | fixed text, size 10 |  | | C\_COMMENT | variable text, size 117 |  | | PRIMARY KEY: C\_CUSTKEY | | |   **(5) 订单ORDERS Table Layout**   |  |  |  | | --- | --- | --- | | Column Name | Data Type | Comment | | O\_ORDERKEY | identifier | SF\*1,500,000 | | O\_CUSTKEY | identifier | Foreign Key to C\_CUSTKEY | | O\_ORDERSTATUS | fixed text, size 1 |  | | O\_TOTALPRICE | Decimal |  | | O\_ORDERDATE | Date |  | | O\_ORDERPRIORITY | fixed text, size 15 |  | | O\_CLERK | fixed text, size 15 |  | | O\_SHIPPRIORITY | Integer |  | | O\_COMMENT | variable text, size 79 |  | | PRIMARY KEY: O\_ORDERKEY | | |   备注：Orders are not present for all customers. In fact, one-third of the customers do not have any order in the database. The orders are assigned at random to two-thirds of the customers (see Clause 4: ). The purpose of this is to exercise the capabilities of the DBMS to handle "dead data" when joining two or more tables  **(6) 明细支出 LINEITEM Table Layout**   |  |  |  | | --- | --- | --- | | Column Name | Data Type | Comment | | L\_ORDERKEY | identifier | Foreign Key to O\_ORDERKEY | | L\_PARTKEY | identifier | Foreign key to P\_PARTKEY, first part of the compound Foreign Key to (PS\_PARTKEY, PS\_SUPPKEY) with L\_SUPPKEY | | L\_SUPPKEY | identifier | Foreign key to S\_SUPPKEY, second part of the compound Foreign Key to (PS\_PARTKEY,  PS\_SUPPKEY) with L\_PARTKEY | | L\_LINENUMBER | integer |  | | L\_QUANTITY | decimal |  | | L\_EXTENDEDPRICE | decimal |  | | L\_DISCOUNT | decimal |  | | L\_TAX | decimal |  | | L\_RETURNFLAG | fixed text, size 1 |  | | L\_LINESTATUS | fixed text, size 1 |  | | L\_SHIPDATE | date |  | | L\_COMMITDATE | date |  | | L\_RECEIPTDATE | date |  | | L\_SHIPINSTRUCT | fixed text, size 25 |  | | L\_SHIPMODE | fixed text, size 10 |  | | L\_COMMENT | variable text size 44 |  | | Primary Key**:** L\_ORDERKEY, L\_LINENUMBER | | |   **(7) 国家NATION Table Layout**   |  |  |  | | --- | --- | --- | | Column Name | Data Type | Comment | | N\_NATIONKEY | identifier | 25 nations are populated | | N\_NAME | fixed text, size 25 | Foreign Key to R\_REGIONKEY | | N\_REGIONKEY | identifier |  | | N\_COMMENT | variable text, size 152 |  | | Primary Key**:** N\_NATIONKEY | | |   **(8)地区 REGION Table Layout**   |  |  |  | | --- | --- | --- | | Column Name | Data Type | Comment | | N\_NATIONKEY | identifier | 25 nations are populated | | N\_NAME | fixed text, size 25 | Foreign Key to R\_REGIONKEY | | N\_REGIONKEY | identifier |  | | N\_COMMENT | variable text, size 152 |  | | Primary Key**:** N\_NATIONKEY | | |   编写SQL DDL语句，创建以上数据库关系表：  (1) PART  CREATE TABLE PART  (P\_PARTKEY INTEGER,  P\_NAME VARCHAR (55),  P\_MFGR CHAR(25),  P\_BRAND CHAR (10),  P\_TYPE VARCHAR (25),  P\_SIZE INTEGER,  P\_CONTAINER CHAR (10),  P\_RETAILPRICE DECIMAL (15,2),  P\_COMMENT VARCHAR (23),  PRIMARY KEY(P\_PARTKEY));  (2) **SUPPLIER**  CREATE TABLE SUPPLIER (  S\_SUPPKEY INTEGER,  S\_NAME CHAR (25),  S\_ADDRESS VARCHAR (40),  S\_NATIONKEY INTEGER,  S\_PHONE CHAR (15),  S\_ACCTBAL DECIMAL (15,2),  S\_COMMENT VARCHAR (101),  PRIMARY KEY(S\_SUPPKEY));  (3) **PARTSUPP**  CREATE TABLE PARTSUPP (  PS\_PARTKEY INTEGER REFERENCES PART(P\_PARTKEY),  PS\_SUPPKEY INTEGER REFERENCES SUPPLIER (S\_SUPPKEY),  PS\_AVAILQTY INTEGER,  PS\_SUPPLYCOST DECIMAL (15,2),  PS\_COMMENT VARCHAR (199),  PRIMARY KEY (PS\_PARTKEY, PS\_SUPPKEY));  (4) **CUSTOMER**  CREATE TABLE CUSTOMER (  C\_CUSTKEY INTEGER PRIMARY KEY,  C\_NAME VARCHAR (25),  C\_ADDRESS VARCHAR (40),  C\_NATIONKEY INTEGER REFERENCES NATION (N\_NATIONKEY),  C\_PHONE CHAR (15),  C\_ACCTBAL DECIMAL (15,2),  C\_MKTSEGMENT CHAR (10),  C\_COMMENT VARCHAR (117));  (5) **ORDERS**  CREATE TABLE ORDERS (  O\_ORDERKEY INTEGER PRIMARY KEY,  O\_CUSTKEY INTEGER REFERENCES CUSTOMER (C\_CUSTKEY),  O\_ORDERSTATUS CHAR (1),  O\_TOTALPRICE DECIMAL (15,2),  O\_ORDERDATE DATE,  O\_ORDERPRIORITY CHAR (15),  O\_CLERK CHAR (15),  O\_SHIPPRIORITY INTEGER,  O\_COMMENT VARCHAR (79));  (6) **LINEITEM**  CREATE TABLE LINEITEM (  L\_ORDERKEY INTEGER REFERENCES ORDERS(O\_ORDERKEY),  L\_PARTKEY INTEGER REFERENCES PART(P\_PARTKEY),  L\_SUPPKEY INTEGER REFERENCES SUPPLIER (S\_SUPPKEY),  L\_LINENUMBER INTEGER,  L\_QUANTITY DECIMAL (15,2),  L\_EXTENDEDPRICE DECIMAL (15,2),  L\_DISCOUNT DECIMAL (15,2),  L\_TAX DECIMAL (15,2),  L\_RETURNFLAG CHAR (1),  L\_LINESTATUS CHAR (1),  L\_SHIPDATE DATE,  L\_COMMITDATE DATE,  L\_RECEIPTDATE DATE,  L\_SHIPINSTRUCT CHAR (25),  L\_SHIPMODE CHAR(10),  L\_COMMENT VARCHAR (44),  PRIMARY KEY (L\_ORDERKEY, L\_LINENUMBER),  FOREIGN KEY (L\_PARTKEY, L\_SUPPKEY) REFERENCES PARTSUPP (PS\_PARTKEY, PS\_SUPPKEY));  (7) NATION  CREATE TABLE NATION (  N\_NATIONKEY INTEGER PRIMARY KEY,  N\_NAME CHAR (25),  N\_REGIONKEY INTEGER REFERENCES REGION (R\_REGIONKEY),  N\_COMMENT VARCHAR (152));  (8) REGION  CREATE TABLE REGION (  R\_REGIONKEY INTEGER PRIMARY KEY,  R\_NAME CHAR (25),  R\_COMMENT VARCHAR (152));  以上建表语句为包含完整性约束建表语句。在进行数据导入时，完整性约束将导致数据导入的性能下降。因此，请将以上建表语句中涉及实体完整性约束和参照完整性约束的部分删除，并在数据库的模式TPCH中创建关系表。  请给出删除完整性约束后的SQL语句：  (1) PART  CREATE TABLE PART  (P\_PARTKEY INTEGER,  P\_NAME VARCHAR (55),  P\_MFGR CHAR(25),  P\_BRAND CHAR (10),  P\_TYPE VARCHAR (25),  P\_SIZE INTEGER,  P\_CONTAINER CHAR (10),  P\_RETAILPRICE DECIMAL (15,2),  P\_COMMENT VARCHAR (23));    (2) **SUPPLIER**  CREATE TABLE SUPPLIER (  S\_SUPPKEY INTEGER,  S\_NAME CHAR (25),  S\_ADDRESS VARCHAR (40),  S\_NATIONKEY INTEGER,  S\_PHONE CHAR (15),  S\_ACCTBAL DECIMAL (15,2),  S\_COMMENT VARCHAR (101));    (3) **PARTSUPP**  CREATE TABLE PARTSUPP (  PS\_PARTKEY INTEGER,  PS\_SUPPKEY INTEGER,  PS\_AVAILQTY INTEGER,  PS\_SUPPLYCOST DECIMAL (15,2),  PS\_COMMENT VARCHAR (199));    (4) **CUSTOMER**  CREATE TABLE CUSTOMER (  C\_CUSTKEY INTEGER,  C\_NAME VARCHAR (25),  C\_ADDRESS VARCHAR (40),  C\_NATIONKEY INTEGER,  C\_PHONE CHAR (15),  C\_ACCTBAL DECIMAL (15,2),  C\_MKTSEGMENT CHAR (10),  C\_COMMENT VARCHAR (117));    (5) **ORDERS**  CREATE TABLE ORDERS (  O\_ORDERKEY INTEGER,  O\_CUSTKEY INTEGER,  O\_ORDERSTATUS CHAR (1),  O\_TOTALPRICE DECIMAL (15,2),  O\_ORDERDATE DATE,  O\_ORDERPRIORITY CHAR (15),  O\_CLERK CHAR (15),  O\_SHIPPRIORITY INTEGER,  O\_COMMENT VARCHAR (79));    (6) **LINEITEM**  CREATE TABLE LINEITEM (  L\_ORDERKEY INTEGER,  L\_PARTKEY INTEGER,  L\_SUPPKEY INTEGER,  L\_LINENUMBER INTEGER,  L\_QUANTITY DECIMAL (15,2),  L\_EXTENDEDPRICE DECIMAL (15,2),  L\_DISCOUNT DECIMAL (15,2),  L\_TAX DECIMAL (15,2),  L\_RETURNFLAG CHAR (1),  L\_LINESTATUS CHAR (1),  L\_SHIPDATE DATE,  L\_COMMITDATE DATE,  L\_RECEIPTDATE DATE,  L\_SHIPINSTRUCT CHAR (25),  L\_SHIPMODE CHAR(10),  L\_COMMENT VARCHAR (44));    (7) NATION  CREATE TABLE NATION (  N\_NATIONKEY INTEGER,  N\_NAME CHAR (25),  N\_REGIONKEY INTEGER,  N\_COMMENT VARCHAR (152));    (8) REGION  CREATE TABLE REGION (  R\_REGIONKEY INTEGER,  R\_NAME CHAR (25),  R\_COMMENT VARCHAR (152));    3导入已生成的TPC-H基准测试数据  将使用工具自动生成的TPC-H基准测试数据导入到数据库中，具体操作步骤如下：  （1）将当前使用的数据库设置为TPC；  （2）使用Query Tool导入数据：  数据打包为tables-out.rar文件，请将文件解压后存储在本地路径；  使用Query Tool执行如下脚本导入数据，注意替换其中红色的路径为本地数据文件路径。  D:\ tables-out  Copy region FROM 'D:\tables-out\region.tbl' WITH DELIMITER AS '|';  Copy nation FROM 'D:\tables-out\nation.tbl' WITH DELIMITER AS '|';  Copy part FROM 'D:/tables-out/part.tbl' WITH DELIMITER AS '|';  Copy supplier FROM 'D:/tables-out/supplier.tbl' WITH DELIMITER AS '|';  Copy customer FROM 'D:/tables-out/customer.tbl' WITH DELIMITER AS '|';  Copy lineitem FROM 'D:/tables-out/lineitem.tbl' WITH DELIMITER AS '|';  Copy partsupp FROM 'D:/tables-out/partsupp.tbl' WITH DELIMITER AS '|';  Copy orders FROM 'D:/tables-out/orders.tbl' WITH DELIMITER AS '|';  注意：Copy命令要求数据文件在服务器本地  4添加约束  数据导入后，向数据库增加数据完整性约束。执行以下脚本：  -- For table REGION  ALTER TABLE REGION  ADD PRIMARY KEY (R\_REGIONKEY);  图片包含 屏幕截图  描述已自动生成  -- For table NATION  ALTER TABLE NATION  ADD PRIMARY KEY (N\_NATIONKEY);  图片包含 屏幕截图  描述已自动生成  ALTER TABLE NATION  ADD FOREIGN KEY (N\_REGIONKEY) references REGION;  图片包含 屏幕截图  描述已自动生成  COMMIT WORK;  -- For table PART  ALTER TABLE PART  ADD PRIMARY KEY (P\_PARTKEY);  COMMIT WORK;  -- For table SUPPLIER  ALTER TABLE SUPPLIER  ADD PRIMARY KEY (S\_SUPPKEY);  ALTER TABLE SUPPLIER  ADD FOREIGN KEY (S\_NATIONKEY) references NATION;  图片包含 屏幕截图  描述已自动生成  COMMIT WORK;  -- For table PARTSUPP  ALTER TABLE PARTSUPP  ADD PRIMARY KEY (PS\_PARTKEY, PS\_SUPPKEY);  图片包含 屏幕截图  描述已自动生成  COMMIT WORK;  -- For table CUSTOMER  ALTER TABLE CUSTOMER  ADD PRIMARY KEY (C\_CUSTKEY);  ALTER TABLE CUSTOMER  ADD FOREIGN KEY (C\_NATIONKEY) references NATION;  COMMIT WORK;  -- For table LINEITEM  ALTER TABLE LINEITEM  ADD PRIMARY KEY (L\_ORDERKEY, L\_LINENUMBER);  COMMIT WORK;  -- For table ORDERS  ALTER TABLE ORDERS  ADD PRIMARY KEY (O\_ORDERKEY);  COMMIT WORK;  -- For table PARTSUPP  ALTER TABLE PARTSUPP  ADD FOREIGN KEY (PS\_SUPPKEY) references SUPPLIER;  COMMIT WORK;  ALTER TABLE PARTSUPP  ADD FOREIGN KEY (PS\_PARTKEY) references PART;  COMMIT WORK;  -- For table ORDERS  ALTER TABLE ORDERS  ADD FOREIGN KEY (O\_CUSTKEY) references CUSTOMER;  图片包含 屏幕截图  描述已自动生成  COMMIT WORK;  -- For table LINEITEM  ALTER TABLE LINEITEM  ADD FOREIGN KEY (L\_ORDERKEY) references ORDERS;  图片包含 屏幕截图  描述已自动生成  COMMIT WORK;  ALTER TABLE LINEITEM  ADD FOREIGN KEY (L\_PARTKEY, L\_SUPPKEY) references PARTSUPP;  图片包含 屏幕截图  描述已自动生成  COMMIT WORK;  以上脚本执行顺序是否可以改变？为什么？  不可以。因为设置外键需要将先设置好引入外键的那个表的主键，才能设置本表的外键，例如“ADD FOREIGN KEY (L\_ORDERKEY) references ORDERS;”则是需要先设置好表orders的主键才能完成此操作。  3 执行基本查询  对数据库各个表进行查询测试，查看其中数据  图片包含 文字, 屏幕截图  描述已自动生成  图片包含 文字  描述已自动生成 | | | | |
| 实验总结：  1 TPC-H的ER图应该是什么样的？请还原出TPC-H的ER图    2 在TPCH的数据导入环节中，关系表数据的导入顺序是否可以改变？如果在建表时增加了完整性约束，数据导入的顺序是否可以改变？如果需要改变，应如何调整，为什么？  可以改变，因为没有完整性约束。如果增加完整性约束则不可以改变。如果需要改变应当使用事务，在所有数据导入后commit检查完整性约束。  3 实验中遇到的难点有哪些？  导入数据库时不知道该怎么做，经历了搜索尝试后才成功。  4 实验自我评价。  本次实验中， 由于中文路径问题出现过一些BUG，后来经过搜索引擎解决。 | | | | |

实验报告1.4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告1.4 | | | | |
| 题目：PostgreSQL数据库备份与恢复操作 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.5.10 |
| 1 PostgreSQL数据库备份  （1）在pgAdmin界面中左侧树状菜单中选择要备份的数据库  （2）右键点击数据库，出现菜单  （3）选择backup选项，填写相关信息  （4）点击“backup”完成备份  2 PostgreSQL数据库恢复  （1）在pgAdmin中创建要恢复的数据库  （2）在数据库右键菜单中选择“restore”，选择数据库备份文件进行恢复  3 每次实验课课后备份数据库，并复制文件，在下次实验课时恢复数据库并在恢复的数据库上进行实验操作。 | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  无  2 实验自我评价。  掌握了数据库的备份和还原操作。 | | | | |

# 实验2: SQL的数据操作实验

实验报告2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告2.1 | | | | |
| 题目：SQL基本查询 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.5.11 |
| 完成以下实验内容，其中标注“\*”的需要自己编写SQL查询语句并写在实验报告中。  1. 导入实验数据  使用SQL脚本导入Enrolled数据库各关系表的数据。  脚本文件包括create table.txt, insert data.txt  2. 单表查询，在Enrolled数据库上完成以下查询SQL语句的编写  （1）（单表查询-投影）求选修了课程的学生的学号  1）要求：不消除重复元组，写出其SQL语句  select Sno  from SC;    2）要求：消除重复元组，写出其SQL语句  select distinct Sno  from SC;    3）以下查询结果与上面哪个结果相同？  select Student.Sno from Student, SC where Student.Sno=SC.Sno  与1）相同。  （2）（单表查询-投影）求开设的课程号、课程名、学时和学分  要求：对查询结果的列重新命名为“课号”，“课名”，“学时”和“学分”  select Cno 课号, cname 课名, chour 学时, Ccredit学分  from Course  图片包含 屏幕截图  描述已自动生成  （3）（单表查询）求计算机系（‘CS’）和数学系（‘MA’）的学生学号、姓名和年龄  要求：两种以上SQL语句写法  Select Sno, Sname, Sage  From student  Where Sdept='CS' or Sdept='MA'    Select Sno, Sname, Sage  From student  Where Sdept in ('CS', 'MA');    （4）（单表查询）求不是数学系、计算机系的学生的情况  要求：两种以上SQL语句写法  Select \*  From student  Where sdept! ='CS' and Sdept! ='MA'    Select \*  From student  Where Sdept not in ('CS', 'MA');    （5）求全体学生的信息  要求：按年龄升序排列  Select \*  From student  Order by Sage    （6）求计算机系年龄在18~20岁之间的学生姓名和年龄  要求：两种以上SQL语句写法  1）  Select Sname,Sage  From student  Where Sdept='CS' and Sage between 18 and 20    2）  Select Sname, Sage  From student  Where Sdept= 'CS' and Sage<=20and Sage>=18  图片包含 屏幕截图  描述已自动生成  （7）求姓名是以”李”开头的计算机系学生  select Sname  from student  where Sname like '李%' and Sdept='CS'  图片包含 屏幕截图  描述已自动生成  3 完成多表查询语句  （8）求选修了数据库课程的学生的学号、姓名、成绩  select Student.Sno,Sname,SC.Grade  from Student, Course, SC  where Student.Sno=SC.Sno and SC.Cno=Course.Cno and Course.Cname='数据库原理'    （9）求学号为末尾是5的学生的学号和所选修的课程的学分  Select student.sno, course.ccredit  From student, course, sc  Where student.sno=sc.sno and sc.cno=course.cno and student.sno like '%5'    （10）求选修“数据库原理”课程且成绩为80分以上的学生的学号、姓名和成绩  Select student.sno, student. sname, sc. grade  From student, course, sc  Where student.sno=sc.sno and sc.cno=course.cno and course.cname='数据库原理' and grade>80  图片包含 屏幕截图  描述已自动生成  （11）求计算机系每个学生的姓名，选修的课程名和成绩  Select student. sname, course. cname, sc. grade  From student, sc, course  Where student. sdept= 'CS' and student.sno=sc.sno and sc.cno=course.cno    （12）求每一门课程的间接先行课号  Select c1.cno, c2. cpno  From course c1, course c2  Where c1. cpno=c2.cno  图片包含 屏幕截图  描述已自动生成  （13）查询没有选修任何课程的学生姓名\*  SELECT student.sname  FROM student  WHERE student.sno NOT IN (SELECT DISTINCT sc.sno FROM sc)  图片包含 屏幕截图  描述已自动生成  4 完成分组聚合查询语句  （14）求每个学生的学号及平均成绩  Select sno, avg(grade)  From sc  Group by sno    （15）求男学生（ssex=‘M’）每一年龄组中超过1人的年龄组及人数  Select sage, count(sno)  From student  Where student. ssex='M'  Group by sage  Having count(sno)>1    （16）查询每个系的学生中年龄最大的学生姓名、年龄  以下语句是否能够执行？如不能执行请说明原因，并修改为正确的语句。  Select sname, max(sage)  From student  Group by sdept  不能，Sname属性出现在secect子句中，它既没有被分组操作，也没有被聚集操作。  select s1.sname,s1.sage  From Student s1,  (select sdept,max(sage)as age from student group by sdept)s2  Where s1.sdept=s2.sdept and s1.sage = s2.age  图片包含 屏幕截图  描述已自动生成  （17）查询计算机系（CS）选课人数超过5人的课程的课程名\*  select cname  from course,(select distinct Cno as cc,count(t1.s)  from SC,(select distinct sc.sno as s from SC,student  where sdept = 'cs' and student.sno = sc.sno)t1  where sc.sno = t1.s  group by Cno  having count(t1.s)>5)t2    5 使用嵌套查询语句完成以下查询  （18）查询没有选修“数据库原理”课程的学生\*  select \* from student  where sno not in  (select sno from sc where cno in (select cno from course where cname = '数据库原理'))  图片包含 屏幕截图  描述已自动生成  （19）求选修了全部课程的学生的学号  Select sno  From student  Where not exists (select\*  From course  Where not exists (select\*  From sc  Where sno=student.sno and sc. cno=course.cno))  图片包含 屏幕截图  描述已自动生成  （20）求选修课程记录中，男生里成绩最好和最差的学生的姓名，性别和成绩，以及女生里成绩最好和最差的学生的姓名，性别和成绩 \*  Select student.ssex, max(grade)  From student, SC  Where student.sno=SC.sno  Group by student.ssex  Union  Select student.ssex, min(grade)  From student, SC  Where student.sno=SC.sno  Group by student.ssex  图片包含 屏幕截图  描述已自动生成 | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  按照具体的要求来写SQL语句，有一些是自己不熟悉的。  2 实验自我评价。  经过查询课本和PPT后掌握了之前不熟悉的SQL语句。 | | | | |

实验报告2.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告2.2 | | | | |
| 题目：SQL查询与查询执行计划分析 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.5.11 |
| 基于TPC-H数据库模式执行或编写查询语句：  （1）连接查询语句  查询供应价格大于零售价格的零件名、制造商名、零售价格和供应价格  1)SQL语句1  Select P.P\_NAME, P.P\_mfgr, P.P\_RETAILPRICE, PS.PS\_SUPPLYCOST  From part P, partsupp PS  Where P.P\_RETAILPRICE>PS.PS\_SUPPLYCOST  Limit 200  2) SQL语句2  Select P.P\_NAME, P.P\_mfgr, P.P\_RETAILPRICE, PS.PS\_SUPPLYCOST  From part P, partsupp PS  Where P.P\_RETAILPRICE>PS.PS\_SUPPLYCOST  AND P.P\_PARTKEY =PS.PS\_SUPPKEY  Limit 200  比较以上两个查询的执行计划，并说明哪个查询符合要求  第二个符合要求，第一个查询出来的供应和零售有一些是不对应的。  （2）三表连接的查询计划  查询顾客“Customer#000000000”订购的订单编号、总价及其订购的零件编号、数量和明细价格，编写SQL查询语句，并查看执行计划（截图）。其中“??”为自己的学号后两位。（我学号后两位为00，没有记录，这里使用的是01）  Select O\_ORDERKEY, O\_TOTALPRICE, L\_PARTKEY, L\_EXTENDEDPRICE  From ORDERS O, LINEITEM L  Where O.CUSTKEY=000000025 AND O.ORDERKEY=L.ORDERKEY    （3）不相关子查询  查询供应商名“Supplier#000000000”所供应的零件信息。其中“??”为自己的学号后两位。（我学号后两位为00，没有记录，这里使用的是01）  1）使用IN的不相关子查询实现  图片包含 文字, 屏幕截图  描述已自动生成  2）使用连接操作实现  图片包含 文字  描述已自动生成  3）对比以上两个SQL语句，查看系统生成的查询执行计划是否相同。  不相同。  （4）简单相关子查询  查询没有购买过“Brand#13”品牌产品的顾客ID和姓名，编写带有EXISTS的SQL语句，并查看执行计划。  SELECT C\_CUSTKEY, C\_NAME  FROM CUSTOMERC  WHERE NOT EXISTS (SELECT O.O\_CUSTKEY  FROM Orders O, Lineitem L, PartSupp PS, Part P  WHERE C.C\_CUSTKEY=O.O\_CUSTKEY AND  O.O\_ORDERKEY=L.L\_ORDERKEY AND  L.L\_PARTKEY=PS.PS\_PARTKEY AND  L.L\_SUPPKEY= PS.PS\_SUPPKEY AND  PS.PS\_PARTKEY=P.P\_PARTKEY AND  P.P\_BRAND='Brand#13')  Limit 200;    （5）双层嵌套相关子查询  查询至少购买过顾客“Customer#0000000??”所购买过的全部零件的顾客姓名，其中“??”为自己的学号后两位。编写SQL语句，运行并查看执行计划。  Select distinct(c\_name)  From Customer C1  Where not exists (select \* from orders O1, lineitem I1, Customer C2  Where O1.o\_orderkey=I1.l\_orderkey  and C2.c\_name='Customer#000000000'  and O1.o\_custkey=C2.c\_custkey  and not exists (select \* from orders O2, lineitem I2  where O2.o\_orderkey=I2.l\_orderkey  and O2.o\_custkey=C1.c\_custkey  and I2.l\_partkey= I1.l\_partkey)  )  order by c\_name limit 100;    （6）子查询派生表  查询订单平均额度超过10000的法国（FRANCE）顾客信息。要求使用FROM子句中的派生表实现。编写SQL语句，并查看执行计划  SELECT \*  FROM CUSTOMER C  WHERE 10000 < ( SELECT AVG(O\_TOTALPRICE)  FROM ORDERS O  WHERE O.O\_CUSTKEY=C.C\_CUSTKEY) | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  有一部分查询方式不是很熟悉。  2 实验自我评价。  通过实际训练对查询SQL更加熟悉了。 | | | | |

实验报告2.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告2.3 | | | | |
| 题目： SQL数据更新实验 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.5.25 |
| （1）INSERT语句  1）完整数据列  插入一条学生记录  insert into Student values('20150020', '苏三', 'M', 20,'CS');    2）部分数据列  插入一条学生记录  insert into Student(Sno, Sname, Ssex) values('20150021', '陈东', 'M');    3）重复执行插入语句  编写SQL语句将本寝室同学信息插入数据库  insert into Student(Sno, Sname, Ssex)  SELECT 20161111,'zh'  UNION ALL  SELECT 20162222,'li'  UNION ALL  SELECT 20163333,'lv' ;  UNION ALL  SELECT 20164444,'yang' ;  UNION ALL  （2）插入批量数据  创建一个计算机系学生表，把所有计算机系学生插入到新表中  CREATE TABLE CS\_S (  Sno VARCHAR (20),  Sname VARCHAR (30),  Ssex VARCHAR (2),  Sage INTEGER,  Sdept VARCHAR (30),  PRIMARY KEY (Sno))  或者  CREATE TABLE CS\_S AS SELECT \* FROM Student WITH NO DATA;    插入计算机系学生数据  insert into CS\_S  SELECT \*  FROM Student  WHERE Sdept='CS';    （3）修改记录  1）修改部分记录  编写SQL语句，修改C4课程的成绩，将成绩不足60分的修改为60分，记录返回结果  UPDATE SC  SET Grade=60  WHERE 'C4'=(SELECT Cname FROM Course WHERE Cno=SC.Cno) AND Grade<60  2）修改部分记录-子查询  编写SQL语句，修改信息系学生的“C3”课程成绩，将成绩加10分。  （4）删除记录  编写SQL语句，删除姓名为“李东”的学生信息，及其选修的课程信息。  1）先删除选课记录  DELETE FROM SC  WHERE Sno=(SELECT Sno FROM Student WHERE Sname='李东')  2）再删除学生记录  DELETE FROM Student  Where Sname='李东' | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  无。  2 实验自我评价。  通过实际训练更加熟悉SQL语句操作。 | | | | |

实验报告2.4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告2.4 | | | | |
| 题目：基于TPC-H查询语句的PLAP测试 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.5.25 |
| （1）Q1价格统计报告查询  Q1语句是查询lineItems的一个定价总结报告。在单个表lineitem上查询某个时间段内，对已经付款的、已经运送的等各类商品进行统计，包括业务量的计费、发货、折扣、税、平均价格等信息。  Q1语句的特点是：带有分组、排序、聚集操作并存的单表查询操作。这个查询会导致表上的数据有95%到97%行被读取到。  select  l\_returnflag,  l\_linestatus,  sum(l\_quantity) as sum\_qty,  sum(l\_extendedprice) as sum\_base\_price,  sum (l\_extendedprice \* (1 - l\_discount)) as sum\_disc\_price,  sum (l\_extendedprice \* (1 - l\_discount) \* (1 + l\_tax)) as sum\_charge,  avg(l\_quantity) as avg\_qty,  avg(l\_extendedprice) as avg\_price,  avg(l\_discount) as avg\_disc,  count(\*) as count\_order  from  lineitem  where  l\_shipdate <= date'1998-06-01' - interval '90' day  group by  l\_returnflag,  l\_linestatus  order by  l\_returnflag,  l\_linestatus;  执行以上查询，并记录执行时间和执行计划。  19sec and 45msec  QQ截图20190602231142  （2）Q2: 最小代价供货商查询  Q2语句查询获得最小代价的供货商。得到给定的区域内，对于指定的零件（某一类型和大小的零件），哪个供应者能以最低的价格供应它，就可以选择哪个供应者来订货。  Q2语句的特点是：带有排序、聚集操作、子查询并存的多表查询操作。查询语句没有从语法上限制返回多少条元组，但是TPC-H标准规定，查询结果只返回前100行（通常依赖于应用程序实现）。  Q2的查询语句如下：  select  s\_acctbal, s\_name, n\_name, p\_partkey, p\_mfgr, s\_address, s\_phone, s\_comment  from  part, supplier, partsupp, nation, region  where  p\_partkey = ps\_partkey  and s\_suppkey = ps\_suppkey  and p\_size = 20  and p\_type like '%[TYPE]'  and s\_nationkey = n\_nationkey  and n\_regionkey = r\_regionkey  and r\_name = '[REGION]'  and ps\_supplycost = (  select  min(ps\_supplycost)  from  partsupp, supplier, nation, region  where  p\_partkey = ps\_partkey  and s\_suppkey = ps\_suppkey  and s\_nationkey = n\_nationkey  and n\_regionkey = r\_regionkey  and r\_name = '[REGION]'  )  order by  s\_acctbal desc,  n\_name,  s\_name,  p\_partkey;  修改并执行以上查询，并记录执行时间和执行计划。  2 secs and 53 msec  explain_plan_1559489241412_wps图片  （3）Q5: 某地区供货商为公司带来的收入查询  Q5语句查询得到通过某个地区零件供货商而获得的收入（收入按sum(l\_extendedprice \* (1 -l\_discount))计算）统计信息。可用于决定在给定的区域是否需要建立一个当地分配中心。  Q5语句的特点是：带有分组、排序、聚集操作、子查询并存的多表连接查询操作。  Q5的查询语句如下：  select  n\_name,  sum(l\_extendedprice \* (1 - l\_discount)) as revenue //聚集操作  from  customer,orders,lineitem,supplier,nation,region //六表连接  where  c\_custkey = o\_custkey  and l\_orderkey = o\_orderkey  and l\_suppkey = s\_suppkey  and c\_nationkey = s\_nationkey  and s\_nationkey = n\_nationkey  and n\_regionkey = r\_regionkey  and r\_name = '[REGION]' //指定地区，在TPC-H标准指定的范围内随机选择  and o\_orderdate >= date '[DATE]' //DATE是从1993年到1997年中随机选择的一年的1月1日  and o\_orderdate < date '[DATE]' + interval '1' year  group by n\_name //按名字分组  order by revenue desc; //按收入降序排序，注意分组和排序子句不同  43secs and 41msec  （4）Q14: 促销效果查询  Q14语句查询获得某一个月的收入中有多大的百分比是来自促销零件。用以监视促销带来的市场反应。  Q14语句的特点是：带有分组、排序、聚集、子查询、左外连接操作并存的查询操作。  Q14的查询语句如下：  select  100.00 \* sum (case  when p\_type like 'PROMO%' //促销零件  then l\_extendedprice\*(1-l\_discount) //某一特定时间的收入  else 0  end) / sum (l\_extendedprice \* (1 - l\_discount)) as promo\_revenue  from  lineitem, part  where  l\_partkey = p\_partkey  and l\_shipdate >= date '[DATE]' // DATE是从1993年到1997年中任一年的任一月的一号  and l\_shipdate < date '[DATE]' + interval '1' month;  14secs and 24 msec  （5）Q18: 大订单顾客查询  Q18语句查询获得比指定供货量大的供货商信息。可用于决定在订单量大，任务紧急时，验证否有充足的供货商。  Q18语句的特点是：带有分组、排序、聚集、IN子查询操作并存的三表连接操作。查询语句没有从语法上限制返回多少条元组，但是TPC-H标准规定，查询结果只返回前100行（通常依赖于应用程序实现）。  Q18的查询语句如下：  select  c\_name, c\_custkey, o\_orderkey, o\_orderdate, o\_totalprice, //基本信息  sum(l\_quantity) //订货总数  from  customer, orders, lineitem  where  o\_orderkey in ( //带有分组操作的IN子查询  select  l\_orderkey  from  lineitem  group by  l\_orderkey having  sum(l\_quantity) > [QUANTITY] // QUANTITY是位于312到315之间的任意值  )  and c\_custkey = o\_custkey  and o\_orderkey = l\_orderkey  group by  c\_name,  c\_custkey,  o\_orderkey,  o\_orderdate,  o\_totalprice  order by  o\_totalprice desc,  o\_orderdate  limit 100;  8 secs and 24msec  （6）Q21: 不能按时交货供货商查询  Q21语句查询获得不能及时交货的供货商。  Q21语句的特点是：带有分组、排序、聚集、EXISTS子查询、NOT EXISTS子查询操作并存的四表连接操作。查询语句没有从语法上限制返回多少条元组，但是TPC-H标准规定，查询结果只返回前100行（通常依赖于应用程序实现）。  Q21的查询语句如下：  select  s\_name, count (\*) as numwait  from  supplier, lineitem l1, orders, nation  where  s\_suppkey = l1.l\_suppkey  and o\_orderkey = l1.l\_orderkey  and o\_orderstatus = 'F'  and l1.l\_receiptdate > l1.l\_commitdate  and exists ( //EXISTS子查询  select  \*  from  lineitem l2  where  l2.l\_orderkey = l1.l\_orderkey  and l2.l\_suppkey <> l1.l\_suppkey  )  and not exists ( //NOT EXISTS子查询  select  \*  from  lineitem l3  where  l3.l\_orderkey = l1.l\_orderkey  and l3.l\_suppkey <> l1.l\_suppkey  and l3.l\_receiptdate > l3.l\_commitdate  )  and s\_nationkey = n\_nationkey  and n\_name = '[NATION]' //TPC-H标准定义的任意值  group by  s\_name  order by  numwait desc,  s\_name  limit 100;  5 secs and 21 msec  对于以上6个查询，完成下表，其中优化部分为选做   |  |  |  |  | | --- | --- | --- | --- | | 查询 | 执行时间ms | 优化方法\* | 优化后执行时间\*ms | | Q1 | 19045 |  |  | | Q2 | 2053 |  |  | | Q5 | 43041 |  |  | | Q14 | 14024 |  |  | | Q18 | 8024 |  |  | | Q21 | 5021 |  |  | | 执行查询的机器配置：  CPU: Ryzen 2500U  内存: 8GB  PostgreSQL版本：PostgreSQL 10 | | | | | | | | |
| 实验总结：  1 能想到何种方法提高查询的效率？  建立索引，合理规划SQL语句尤其是查询，增强查询计划的优化等等。 | | | | |

# 实验3:数据库的调优、安全性和完整性

实验报告3.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告3.1 | | | | |
| 题目：数据库查询优化与调优 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.6.5 |
| （1）创建唯一索引  1）执行在Customer表上的查询，查询顾客“Customer#000103500”的信息，SQL语句如下：  SELECT \*  FROM public. customer  where c\_name='Customer#000103500';  执行10次，记录查询执行时间和平均值到后面表格  2）查看执行计划  执行：  EXPLAIN SELECT \*  FROM public. customer  where c\_name='Customer#000103500';  执行计划为：  Gather (cost=1000.00..5366.35 rows=1 width=159)  Workers Planned: 2  -> Parallel Seq Scan on customer (cost=0.00..4366.25 rows=1 width=159)  Filter: ((c\_name)::text = 'Customer#000103500'::text)  3）在c\_name属性上创建唯一索引  CREATE UNIQUE INDEX idx\_custname ON customer (c\_name);    4）再次执行查询顾客“Customer#000103500”的SQL语句  执行10次，记录查询执行时间和平均值到后面表格  5）再次查看执行计划  执行计划为：  Index Scan using idx\_custname on customer (cost=0.42..8.44 rows=1 width=159)  Index Cond: ((c\_name)::text = 'Customer#000103500'::text)     |  |  |  | | --- | --- | --- | | 执行次数 | 创建索引前执行时间ms | 创建索引后执行时间ms | | 1 | 302 | 84 | | 2 | 298 | 75 | | 3 | 295 | 78 | | 4 | 301 | 79 | | 5 | 300 | 76 | | 6 | 295 | 78 | | 7 | 300 | 77 | | 8 | 303 | 75 | | 9 | 301 | 79 | | 10 | 308 | 78 | | 平均值 | 300.3 | 77.9 |   （2）创建复合索引  1）执行在Part表上的查询  select \*  from part  where p\_mfgr='Manufacturer#5' and p\_brand='Brand#53'  查看以上查询执行计划    执行10次，记录查询执行时间和平均值到后面表格  2）创建复合索引  CREATE INDEX idx\_mfgr\_brand ON part (p\_mfgr, p\_brand)    3）执行查询  重新查看以上查询执行计划    执行10次，记录查询执行时间和平均值到后面表格   |  |  |  | | --- | --- | --- | | 执行次数 | 创建索引前执行时间ms | 创建索引后执行时间ms | | 1 | 286 | 78 | | 2 | 287 | 79 | | 3 | 289 | 77 | | 4 | 295 | 78 | | 5 | 284 | 76 | | 6 | 286 | 79 | | 7 | 288 | 80 | | 8 | 289 | 78 | | 9 | 292 | 77 | | 10 | 291 | 78 | | 平均值 | 288.7 | 78.0 |   4）分析查询  查看以下查询的执行计划，看是否使用了索引  select \* from part where p\_brand='Brand#53'  查询计划为：  Seq Scan on part (cost=0.00..6597.00 rows=8147 width=130)  Filter: (p\_brand = 'Brand#53'::bpchar)  可见，没有使用索引。  （3）Hash索引  1）执行在Part表上的查询  select \*  from part  where p\_name='bisque tan cyan sky drab'  查看以上查询执行计划    执行10次，记录查询执行时间和平均值到后面表格  2）创建Hash索引  CREATE INDEX idx\_partname\_hash ON part USING hash (p\_name)    3）重新执行查询  查看以上查询执行计划  执行10次，记录查询执行时间和平均值到后面表格   |  |  |  | | --- | --- | --- | | 执行次数 | 创建索引前执行时间ms | 创建索引后执行时间ms | | 1 | 286 | 65 | | 2 | 285 | 64 | | 3 | 283 | 67 | | 4 | 289 | 82 | | 5 | 288 | 72 | | 6 | 284 | 64 | | 7 | 286 | 65 | | 8 | 287 | 67 | | 9 | 288 | 64 | | 10 | 285 | 62 | | 平均值 | 286.1 | 67.2 |   （4）聚簇索引  在PostgreSQL中并没有“create cluster index”语句用以创建聚簇索引，而是用“create index”创建索引后再对关系表用“cluster indexname on tbname”创建聚簇存储。  1）执行在Partsupp表上的查询  select \*  from partsupp  where ps\_suppkey= 6114;  查看以上查询执行计划    执行10次，记录查询执行时间和平均值到后面表格  2）创建聚簇索引  首先创建属性ps\_suppkey上的索引  CREATE INDEX idx\_ps\_suppkey ON Partsupp (ps\_suppkey);  对关系表进行聚簇存储  CLUSTER idx\_ps\_suppkey ON Partsupp;  3）重新执行查询  查看以上查询执行计划    执行10次，记录查询执行时间和平均值到后面表格   |  |  |  | | --- | --- | --- | | 执行次数 | 创建索引前执行时间ms | 创建索引后执行时间ms | | 1 | 379 | 74 | | 2 | 298 | 67 | | 3 | 302 | 62 | | 4 | 288 | 64 | | 5 | 305 | 63 | | 6 | 323 | 60 | | 7 | 314 | 61 | | 8 | 325 | 59 | | 9 | 311 | 64 | | 10 | 286 | 66 | | 平均值 | 313.1 | 64.0 |   **注意**：  该表实际上按索引顺序拷贝到了一个临时表中，然后重新改成原名．因此，在建簇时所有赋予的权限和其它索引都将丢失．  如果你只是随机的访问表中的行，那么在堆表中的数据的实际存储顺序是无关紧要的．但是，如果你对某些数据的访问多于其他数据，而且有一个索引将这些数据分组，那你就将从CLUSTER中获益．  （5）B+Tree索引有效性  B+Tree索引可以用于等值和范围查询，对于LIKE匹配操作符的查询，仅在匹配模式存在一个常量，且常量位于模式的起始位置时才有效。  执行以下三个查询，记录查询执行时间和执行计划。  1）Q1  SELECT \*  FROM public. customer  where c\_name LIKE 'Customer#0001035%';  2）Q2  SELECT \*  FROM public. customer  where c\_name LIKE 'Customer%500';  3）Q3  SELECT \*  FROM public. customer  where c\_name LIKE '%500';   |  |  |  |  | | --- | --- | --- | --- | | 查询 | 执行时间ms | 查询执行计划 | 是否使用索引 | | Q1 | 125 | Seq Scan on customer (cost=0.00..5460.00 rows=1515 width=159)  Filter: ((c\_name)::text ~~ 'Customer#0001035%'::text) | 是 | | Q2 | 328 | Gather (cost=1000.00..5367.75 rows=15 width=159)  Workers Planned: 2  -> Parallel Seq Scan on customer (cost=0.00..4366.25 rows=6 width=159)  Filter: ((c\_name)::text ~~ 'Customer%500'::text) | 否 | | Q3 | 319 | Gather (cost=1000.00..5367.75 rows=15 width=159)  Workers Planned: 2  -> Parallel Seq Scan on customer (cost=0.00..4366.25 rows=6 width=159)  Filter: ((c\_name)::text ~~ '%500'::text) | 否 |   （6）索引设计\*  1）编写SQL语句，查询在“1995年11月”购买了零件编号为“123008”的顾客姓名。  记录查询时间  2）创建索引  设计并创建索引  3）重新执行查询，并记录查询执行时间   |  |  | | --- | --- | | SQL语句 | select c\_name  from customer, orders, lineitem  where o\_orderdate between '1995-11-01' and '1995-11-30'  and l\_orderkey=o\_orderkey  and l\_partkey=123008  and c\_custkey=o\_custkey | | 执行时间 | 397ms | | 创建索引语句 | create index idx\_o\_orderdate on orders (o\_orderdate);  create index idx\_o\_custkey on orders (o\_custkey);  create index idx\_l\_orderkey on lineitem (l\_orderkey);  create index idx\_l\_partkey on lineitem (l\_partkey) | | 使用索引执行时间 | 89ms |   （7）SQL语句优化\*\*  查询没有供应过零件的供应商名称  1）Q1使用Not IN语句  Select S.S\_name  From supplier S  Where S.S\_suppkey not in (SELECT PS\_suppkey  From partsupp PS);  2）Q2使用not exist语句实现  Select S.S\_name  From supplier S  Where not exists (select 1 from partsupp PS where PS.PS\_suppkey= S.S\_suppkey);  3）Q3使用外链接  Select S.S\_name  From supplier S Left outer join partsupp PS on PS.PS\_suppkey= S.S\_suppkey  Where PS.PS\_suppkey is null;  对比以上三个查询语句的执行时间和执行计划。   |  |  |  |  | | --- | --- | --- | --- | | 查询 | 执行时间ms | 查询执行计划 | 是否使用索引 | | Q1 | 600000 |  | 否 | | Q2 | 308 |  | 否 | | Q3 | 123 |  | 是 | | | | | |
| 实验思考题：  1 PostgreSQL的索引还有哪些？那些索引如何使用？  （1）b-tree适合所有的数据类型，支持排序，支持大于、小于、等于、大于或等于、小于或等于的搜索。索引与递归查询结合，还能实现快速的稀疏检索。  （2）hash索引存储的是被索引字段VALUE的哈希值，只支持等值查询。hash索引特别适用于字段VALUE非常长（不适合b-tree索引，因为b-tree一个PAGE至少要存储3个ENTRY，所以不支持特别长的VALUE）的场景，例如很长的字符串，并且用户只需要等值搜索，建议使用hash index。  （3）gin是倒排索引，存储被索引字段的VALUE或VALUE的元素，以及行号的list或tree。应用场景：当需要搜索多值类型内的VALUE时，适合多值类型，例如数组、全文检索、TOKEN。（根据不同的类型，支持相交、包含、大于、在左边、在右边等搜索）；当用户的数据比较稀疏时，如果要搜索某个VALUE的值，可以适应btree\_gin支持普通btree支持的类型。（支持btree的操作符）；当用户需要按任意列进行搜索时，gin支持多列展开单独建立索引域，同时支持内部多域索引的bitmapAnd, bitmapOr合并，快速的返回按任意列搜索请求的数据。  （4）GiST是一个通用的索引接口，可以使用GiST实现b-tree, r-tree等索引结构。不同的类型，支持的索引检索也各不一样。例如：1、几何类型，支持位置搜索（包含、相交、在上下左右等），按距离排序。2、范围类型，支持位置搜索（包含、相交、在左右等）。3、IP类型，支持位置搜索（包含、相交、在左右等）。4、空间类型（PostGIS），支持位置搜索（包含、相交、在上下左右等），按距离排序。5、标量类型，支持按距离排序。  （5）SP-GiST类似GiST，是一个通用的索引接口，但是SP-GIST使用了空间分区的方法，使得SP-GiST可以更好的支持非平衡数据结构，例如quad-trees, k-d tree, radis tree。应用场景：1、几何类型，支持位置搜索（包含、相交、在上下左右等），按距离排序。2、范围类型，支持位置搜索（包含、相交、在左右等）。3、IP类型，支持位置搜索（包含、相交、在左右等）。  （7）BRIN 索引是块级索引，有别于B-TREE等索引，BRIN记录并不是以行号为单位记录索引明细，而是记录每个数据块或者每段连续的数据块的统计信息。因此BRIN索引空间占用特别的小，对数据写入、更新、删除的影响也很小。  （8）rum 是一个索引插件，由Postgrespro开源，适合全文检索，属于GIN的增强版本。1、在RUM索引中，存储了lexem的位置信息，所以在计算ranking时，不需要回表查询（而GIN需要回表查询）。2、RUM支持phrase搜索，而GIN无法支持。3、在一个RUM索引中，允许用户在posting tree中存储除ctid（行号）以外的字段VALUE，例如时间戳。  （9）bloom索引接口是PostgreSQL基于bloom filter构造的一个索引接口，属于lossy索引，可以收敛结果集(排除绝对不满足条件的结果，剩余的结果里再挑选满足条件的结果)，因此需要二次check，bloom支持任意列组合的等值查询。  （10）zombodb是PostgreSQL与ElasticSearch结合的一个索引接口，可以直接读写ES。  （11）bitmap索引是Greenplum的索引接口，类似GIN倒排，只是bitmap的KEY是列的值，VALUE是BIT（每个BIT对应一行），而不是行号list或tree。当某个字段的唯一值个数在100到10万之间(超出这个范围，不建议使用bitmap)时，如果表的记录数特别多，而且变更不频繁（或者是AO表），那么很适合BITMAP索引，bitmap索引可以实现快速的多个或单个VALUE的搜索。因为只需要对行号的BITMAP进行BIT与或运算，得到最终的BITMAP，从最终的BITMAP映射到行进行提取。  （12）表达式索引也是PostgreSQL特有的特性，例如用户的数据需要转换后查询，例如某些设备上传的地理坐标的坐标系不符合国标，需要转换为国内的空间坐标来查询。  2 实验自我评价。  在本次实验中了解到了很多的数据库索引知识及其相关操作方式方法。 | | | | |

实验报告3.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告3.2 | | | | |
| 题目：数据库视图定义 | 姓名 | 姓名 | 戚子强 | 班级 |
| 学号 | 学号 | 20164625 | 时间 |
| 1 创建视图  创建计算机系学生信息视图v\_CS\_Students。  create view v\_CS\_Students  as  select \*  from Student  where Sdept='CS';  创建后执行查询：  Select \*  From v\_CS\_Students  查看查询结果数量   |  |  | | --- | --- | | 执行结果元组数量 | 17 |   2 创建聚合查询视图  创建学生平均成绩视图v\_avg\_grade，表示学生的平均成绩，其中包括Sno, name，avgGrade。  create view v\_avg\_grade (Sno, Name, avgGrade)  as  select SC. Sno, sname, avg (Grade)  from SC, Student  where SC. Sno=Student.Sno  group by SC. Sno, sname  创建后执行查询，查看前5个记录：  Select \*  From v\_avg\_grade limit 5;  执行结果：   |  |  | | --- | --- | | 执行结果 |  |   3 创建使用with check opition子句的视图  创建数学系学生信息视图v\_MA\_Students，其中包括学号、姓名、年龄和性别。  create view v\_MA\_Students  as  select \*  from Student  where Sdept='MA'  with check option;  创建后执行查询：  Select \*  From v\_MA\_Students    查看查询结果数量   |  |  | | --- | --- | | 执行结果元组数量 | 9 |   4 视图的更新操作  执行以下操作，记录结果，并分析原因，填入后续表格   1. Q1.1：向视图v\_CS\_Students插入记录   insert into v\_CS\_Students values ('96001', '叶修','M',19,'CS')  Q1.2：执行查询，查看记录是否成功插入  Select \* From v\_CS\_Students where sno='96001'   1. Q2.1：向视图v\_CS\_Students插入记录   insert into v\_CS\_Students values ('96002', '刘皓','M',20,'IS')  Q2.2：执行查询，查看记录是否成功插入  Select \* From v\_CS\_Students where sno='96002'  Q2.3：执行查询，查看记录是否成功插入  Select \* From Student where sno='96002'   1. Q3.1：向视图v\_MA\_Students插入记录   insert into v\_MA\_Students values ('96003', '唐柔','W',20,'MA')  Q3.2：执行查询，查看记录是否成功插入  Select \* From v\_MA\_Students where sno='96003'   1. Q4.1：向视图v\_MA\_Students插入记录   insert into v\_MA\_Students values ('96004', '魏琛','M',19,'MA')  Q4.2：查询视图，查看记录是否成功插入  Select \* From v\_MA\_Students where sno='96004'  Q4.3：查询基本表，查看记录是否成功插入  Select \* From Student where sno='96004'   1. Q5.1：更新视图v\_avg\_grade记录   Update v\_avg\_grade set avgGrade=90 where sno='95528';   |  |  |  | | --- | --- | --- | | 查询 | 执行结果 | 原因分析 | | Q1.1 | INSERT 0 1 Query returned successfully in 68 msec. | 成功插入，因为没有重复。 | | Q1.2 | Successfully run. Total query runtime: 73 msec. 1 rows affected. | 查询成功，由Q1.1插入而来。 | | Q2.1 | INSERT 0 1 Query returned successfully in 66 msec. | 插入成功，但不是CS系。 | | Q2.2 | Successfully run. Total query runtime: 72 msec. 0 rows affected. | 查询失败，因为Q2.1插入的不是CS系的。 | | Q2.3 | Successfully run. Total query runtime: 84 msec. 1 rows affected. | 查询成功，因为这里没有做系别限制。 | | Q3.1 | INSERT 0 1 Query returned successfully in 70 msec. | 成功插入到MA系别。 | | Q3.2 | Successfully run. Total query runtime: 152 msec. 0 rows affected. | 读取到Q3.1中插入的结果。 | | Q4.1 | INSERT 0 1 Query returned successfully in 66 msec. | 成功插入。 | | Q4.2 | Successfully run. Total query runtime: 74 msec. 1 rows affected. | 查询到上述插入的操作结果。 | | Q4.3 | Successfully run. Total query runtime: 71 msec. 1 rows affected. | 查到一个结果，因为这个sno是唯一的 | | Q5.1 | 发生错误 | 无法更新视图"v\_avg\_grade" DETAIL，因为包含 GROUP BY 的视图列不能自动更新.；可以启用对视图的更新操作, 需要提供INSTEAD OF UPDATE触发器或者一个无条件的 ON UPDATE DO INSTEAD 规则。 |   5 视图删除  1）创建视图v\_C4\_SC  创建选修了课程号‘c4’的学生号，姓名和成绩  create view v\_C4\_SC (sno, name, grade)  as  select SC.sno, sname, grade  from Student S, SC  where S.sno=SC.sno and cno='c4'  2）创建视图v\_C4\_HS  在视图v\_C4\_SC基础上创建选修了课程号‘c4’且成绩在90分以上的学生的学生号，姓名和成绩  create view v\_C4\_HS  as  select \*  from v\_C4\_SC  where grade>90  3）删除视图  v\_C4\_SC视图受限删除  Drop View v\_C4\_SC Restrict;  v\_C4\_SC视图级联删除  Drop View v\_C4\_SC Cascade;  执行结果和原因分析填入下表   |  |  |  | | --- | --- | --- | | Restrict: | ERROR: 错误: 无法删除 视图 v\_c4\_sc 因为有其它对象倚赖它  DETAIL: 视图 v\_c4\_hs 倚赖于 视图 v\_c4\_sc  HINT: 使用 DROP .. CASCADE 把倚赖对象一并删除. |  | | Cascade | 注意: 递归删除 视图 v\_c4\_hs DROP VIEW |  |   6 物化视图\*\*  物化视图则是将视图映射出的数据存成一张物理表。  1）创建物化视图  CREATE MATERIALIZED VIEW v\_avg\_grade\_m (Sno, Name, avgGrade)  as  select SC. Sno, sname, avg (Grade)  from SC, Student  where SC. Sno=Student.Sno  group by SC. Sno, sname;  2）比较普通视图和雾化视图的执行计划  Q1：EXPLAIN select \* from v\_avg\_grade;  Q2：EXPLAIN select \* from v\_avg\_grade\_m;  将执行计划填入下表   |  |  | | --- | --- | | Q1： |  | | Q2： |  | | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  无。  2 实验自我评价。  过程比较顺利。 | | | | |

实验报告3.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告3.3 | | | | |
| 题目：数据库安全性实验 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.6.5 |
| 1 创建用户  以数据库Enrolled的创建者user1登录数据库并创建用户：  CREATE USER user2 WITH PASSWORD '123456';  CREATE USER user3 WITH PASSWORD '123456';  CREATE USER avguser WITH PASSWORD '123456';  CREATE ROLE Query returned successfully in 70 msec.  2 授权  1）授予user2在表course上全部权限  GRANT ALL PRIVILEGES ON course TO user2;  2）授予user2在表student上的查询权限  GRANT SELECT ON student TO user2 with grant option;  3）授予user2在表SC上的修改成绩权限  GRANT UPDATE (grade) ON SC TO user2;  4）授予avguser在视图v\_avg\_grade上的查询权限  GRANT SELECT ON v\_avg\_grade TO avguser;  3 验证user2权限  使用user2登录数据库Enrolled  1）Q3.1查询course表  Select \* from course  2）Q3.2修改课程号‘c4’课程的学分为4  Update course Set Credit=4 where Cno='c4';  3）Q3.3查询计算机系的学生信息  Select \* from student where sdept='CS';  4）Q3.4更新学生的年龄，增加1岁  Update student Set sage=sage+1;  5）Q3.5更新SC表，对课程号‘c3’的学生成绩加2分  Update SC Set grade= grade +1 where cno='c3';  6）级联授权，将查询student表的权限授权给user3  GRANT SELECT ON student TO user3;  4 验证avguser权限  使用avguser登录数据库Enrolled  1）Q4.1查询视图v\_avg\_grade上的内容  Select \* from v\_avg\_grade;  2）Q4.2查询SC表上的内容  Select \* from SC；  5 验证级联授权  使用user3登录数据库Enrolled  1）Q5.1查询数学系的学生信息  Select \* from student where sdept='MA';  2）Q5.2 查询课程信息  Select \* from course  6 回收权限  使用user1登录数据库  1）回收user2在表SC上的修改成绩权限  REVOKE UPDATE (grade) ON SC FROM user2;  2）回收所有用户对course表的修改权限  REVOKE update ON course FROM public;  3）Q6.3回收user2在表student上的查询权限，受限回收  REVOKE SELECT ON student FROM user2 RESTRICT;  4）Q6.4回收user2在表student上的查询权限，级联回收  REVOKE SELECT ON student FROM user2 CASCADE;  7 验证user2权限  使用user2登录数据库Enrolled  1）Q7.1更新学生的年龄，增加1岁  Update student Set sage=sage+1;  2）Q7.2修改课程号‘c4’课程的学分为5  Update course Set Credit=5 where Cno='c4';  3）Q7.3 查询课程信息  Select \* from course  8 验证user3权限  使用user3登录数据库Enrolled  1）Q8.1查询数学系的学生信息  Select \* from student where sdept='MA';   |  |  |  | | --- | --- | --- | | 查询 | 是否正常执行 | 原因分析 | | Q3.1 | 正常 | user2在表course上全部权限 | | Q3.2 | 正常 | user2在表course上全部权限 | | Q3.3 | 正常 | user2在表student上的查询权限 | | Q3.4 | 失败 | user2没有Student的修改权限 | | Q3.5 | 正常 | user2在表SC上的修改成绩权限 | | Q4.1 | 正常 | avguser在视图v\_avg\_grade上的查询权限 | | Q4.2 | 失败 | 没有相关权限 | | Q5.1 | 正常 | user2将查询student表的权限授权给user3 | | Q5.2 | 失败 | 没有相关权限 | | Q6.3 | 失败 | user2的授权是级联授权 | | Q6.4 | 正常 | user2的授权是级联授权 | | Q7.1 | 失败 | 没有相关权限 | | Q7.2 | 正常 | 修改权限依然存在 | | Q7.3 | 失败 | 查询权限已经被收回 | | Q8.1 | 失败 | user3的权限是由user2授权，user2的权限已经被级联收回 | | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  无。  2 实验自我评价。  在本次实验中，通过实际操作深入了解了上课所学的知识。 | | | | |

实验报告3.4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告3.4 | | | | |
| 题目：数据库完整性 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.6.5 |
| 1 不带实体完整性定义的关系表  编写 SQL语句，创建关系表Department  CREATE TABLE Department (  Dept varchar (30),  Name varchar (30),  Address varchar (50))  插入记录  Insert into Department values('MA','数学系','南湖校区');  Insert into Department values('CS','计算机系','浑南校区');  Insert into Department values('IS','信息系','浑南校区');  Insert into Department values('SE','软件工程系','浑南校区');  Insert into Department values('EE','电气工程系','南湖校区');  Insert into Department values('CS','计算机系','南湖校区');  2 定义实体完整性  在关系表Department上增加实体完整性，定义主键Dept：  ALTER TABLE Department ADD CONSTRAINT pk\_department PRIMARY KEY(Dept);  以上语句是否能够执行？如果不能执行，请给出应对策略，使得以上语句能够执行。   |  |  | | --- | --- | | 是否能够执行 | 否 | | 处理策略 | 设置联合主键：ALTER TABLE Department ADD CONSTRAINT pk\_department PRIMARY KEY(Dept,Name,Address); |   3 创建表后定义参照完整性  在关系表Student上增加参照完整性，Sdept参照Department的Dept：  ALTER TABLE Student  ADD CONSTRAINT fk\_sdepartment FOREIGN KEY (sdept) REFERENCES Department (Dept) ON DELETE CASCADE ON UPDATE SET NULL;  在关系表Student上插入记录：  insert into Student values('96010', '杜明', 'M', 19,'SE');  insert into Student values('96011', '孙翔', 'M', 18,'SE');  insert into Student values('96012', '郑璇', 'W', 19,'EE');  4 参照完整性的违约处理  1）Q1在student表上执行删除操作  删除学号为‘95010’的学生  Delete FROM Student WHERE sno= '95010';  是否执行成功？  2）删除‘EE’系  Delete FROM Department WHERE dept='EE';  查看电气工程系相关学生信息Q2  SELECT \*  From Student  Where sdept='EE';  查看结果如何，并说明原因  3）更新软件工程系信息  将软件工程专业（'SE'）更新为物联网专业（'IOT','物联网'）  UPDATE Department SET dept='IOT',name='物联网' WHERE dept='SE'  查询软件工程系学生Q3  SELECT \*  From Student  Where sdept='SE';  查询原软件工程系学生的专业状态Q4  SELECT \*  From Student  Where sno=' 96010';  查看结果如何，并说明原因   |  |  |  | | --- | --- | --- | | 查询 | 执行结果情况 | 原因分析 | | Q1 | 成功 | 没有相关的完整性约束 | | Q2 | 失败 | 已经被同步删除 | | Q3 | 失败 | 专业已同步更新，再查询之前的就名字查不到了 | | Q4 | 成功 | 专业已同步更新 |   5 用户自定义完整性  创建关系表Department2  CREATE TABLE Department2 (  Dept varchar (30) Primary Key,  Name varchar (30) NOT NULL UNIQUE,  Address varchar(50) DEFAULT '南湖校区'  )  按一下SQL语句顺序执行插入记录，结果如何说明原因  Q1:Insert into Department2 values('MA', '数学系', '南湖校区');  Q2:Insert into Department2 values('CS', '计算机系', '浑南校区');  Q3:Insert into Department2 values('IS',NULL, '浑南校区');  Q4:Insert into Department2 values('SE', '计算机系', '浑南校区');  Q5:Insert into Department(dept,name) values('EE','电气工程系');  Q6: select \* from Department2 where dept='EE';  各语句执行结果，和原因分析填入下表：   |  |  |  | | --- | --- | --- | | 查询 | 执行结果情况 | 原因分析 | | Q1 | 成功 | 符合完整性约束 | | Q2 | 成功 | 符合完整性约束 | | Q3 | 失败 | Name不能是空 | | Q4 | 失败 | 重复键违反唯一约束"department2\_name\_key" | | Q5 | 失败 | 在字段 "address" 中空值违反了非空约束 | | Q6 | 查询为空 | 0 rows affected. |   6用户自定义完整性检查约束（CHECK）  创建关系表Department3  CREATE TABLE Department3 (  Dept varchar (30) Primary Key,  Name varchar (30) NOT NULL UNIQUE,  Address varchar(50) CHECK(Address IN ('南湖校区', '浑南校区'))  )  插入记录，结果如何说明原因  Q1:Insert into Department3 values('MA', '数学系', '南湖校区');  Q2:Insert into Department3 values('CS', '计算机系', '沈河校区');  各语句执行结果，和原因分析填入下表：   |  |  |  | | --- | --- | --- | | 查询 | 执行结果情况 | 原因分析 | | Q1 | 成功 | 符合自定义完整性约束 | | Q2 | 失败 | 关系 "department3" 的新列违反了检查约束 "department3\_address\_check" |   7添检查约束和删除检查约束\*  1）向Department2中增加检查约束  ALTER TABLE Department2 ADD CONSTRAINT dept\_add CHECK (Address IN ('南湖校区', '浑南校区'));  执行插入记录语句  Q1:Insert into Department3 values('BA', '工商管理', '沈河校区');  2）删除检查约束  ALTER TABLE department2 DROP CONSTRAINT dept\_add;  执行插入记录语句  Q2:Insert into Department3 values('BA', '工商管理', '沈河校区');  各语句执行结果，和原因分析填入下表：   |  |  |  | | --- | --- | --- | | 查询 | 执行结果情况 | 原因分析 | | Q1 | 失败 | 关系 "department3" 的新列违反了检查约束 "department3\_address\_check"，行包含(BA, 工商管理, 沈河校区). | | Q2 | 失败 | 关系 "department3" 的新列违反了检查约束 "department3\_address\_check"，行包含(BA, 工商管理, 沈河校区). | | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  暂无。  2 实验自我评价。  在这次实验中，实验比较简单，很顺利地完成了实验。 | | | | |

# 实验4:数据库的事务管理与并发控制

实验报告4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告4.1 | | | | |
| 题目：数据库的事务声明和执行 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.6.5 |
| 1. 登录后使用Enrolled数据库  2. 使用事务向course表中插入记录  插入课程（“c7”，“生产实习”，96，“3”，“c4”）  BEGIN TRANSACTION;  INSERT INTO course VALUES('c7', '生产实习',96,'3','c4');  COMMIT;  执行查询  SELECT \* FROM course WHERE cno='c7';  查看执行结果：   |  |  | | --- | --- | | 执行结果 | 图片包含 屏幕截图  描述已自动生成 | | 分析 | 成功插入 |   3. 使用事务修改课程‘c7’的先行课  更新课程“c7”的先修课为“c5”，执行修改后不提交事务，而是执行回滚  BEGIN TRANSACTION;  UPDATE course SET cpno='c5' WHERE cno='c7';  ROLLBACK;  执行查询  SELECT \* FROM course WHERE cno='c7';  查看执行结果：   |  |  | | --- | --- | | 执行结果 | 图片包含 屏幕截图  描述已自动生成 | | 分析 | 成功回滚 |   4. 设置savepoint的回滚  1）关闭自动提交  使用命令\set AUTOCOMMIT off #关闭自动提交  或通过pgAdmin用户界面设置  05ab567f7822dd3ea8396200976ca26  2）执行带有savepoint的语句  BEGIN；  INSERT INTO course VALUES('c8', '数据库系统实现技术',40,'2.5','c4')；  SAVEPOINT sp1;  INSERT INTO course VALUES('c9', '数据库系统实践',40,'2.5','c4')；  SAVEPOINT sp2;  INSERT INTO course VALUES('c10', '分布式数据库',48,'3','c8')；  SELECT \* FROM course；   |  |  |  | | --- | --- | --- | | 执行结果 |  |  | | 分析 | 插入成功 |  |   ROLLBACK to SAVEPOINT sp2；  SELECT \* FROM course；   |  |  |  | | --- | --- | --- | | 执行结果 |  |  | | 分析 | 回滚至检查点2 |  |   ROLLBACK to SAVEPOINT sp1；  SELECT \* FROM course；   |  |  |  | | --- | --- | --- | | 执行结果 |  |  | | 分析 | 回滚至检查点1 |  |   INSERT INTO course VALUES('c11', '数据库专题',48,'3','c4')；  COMMIT；  SELECT \* FROM course；   |  |  |  | | --- | --- | --- | | 执行结果 |  |  | | 分析 | 回滚至检查点1，插入成功 |  |   5事务的完整性检验  1）打开自动提交  \set AUTOCOMMIT on  或通过pgAdmin用户界面设置  05ab567f7822dd3ea8396200976ca26  插入记录到course表  INSERT INTO course VALUES('c21', '数据挖掘',40,'2.5','c4');  INSERT INTO course VALUES('c22', '人工智能 ',40,'2.5','c21');  2）执行更新操作  DELETE FROM course WHERE cno='c21';  DELETE FROM course WHERE cno='c22';   |  |  |  |  | | --- | --- | --- | --- | | 执行结果 |  |  |  | | 分析 | 插入成功，删除成功 |  |  |   3）声明事务执行  BEGIN TRANSACTION;  DELETE FROM course WHERE cno='c21';  DELETE FROM course WHERE cno='c22';  COMMIT;   |  |  |  | | --- | --- | --- | | 执行结果 |  |  | | 分析 | 删除失败 |  | | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  无。  2 实验自我评价。  本次实验主要对数据库的事务管理与并发控制做了实践操作。 | | | | |

实验报告4.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验报告4.2 | | | | |
| 题目：多事务并发控制隔离性实验 | 姓名 | 戚子强 | 班级 | 计算机1606 |
| 学号 | 20164625 | 时间 | 2019.6.6 |
| 1. 多事务并发  开启两个postgresql的终端，分别用不同的用户进行登录  分别执行两个事务  事务A  BEGIN TRANSACTION  SELECT \* FROM COURSE；  SELECT \* FROM COURSE；  SELECT \* FROM COURSE；  COMMIT;  事务B  BEGIN TRANSACTION  INSERT INTO course VALUES('c15', '生产实习',96,'3','c4')  COMMIT;  两个事务操作按照以下顺序执行，并查看事务A中查询执行结果，填入下表：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 事务A | 事务B | 执行结果 | | 1 | BEGIN TRANSACTION |  | 成功 | | 2 | SELECT \* FROM COURSE |  | 成功 | | 3 |  | BEGIN TRANSACTION | 成功 | | 4 |  | INSERT INTO course VALUES ('c15', '生产实习',96,'3','c4'); | 插入成功 | | 5 | SELECT \* FROM COURSE |  | 未查询到新数据 | | 6 |  | COMMIT; | 提交成功 | | 7 | SELECT \* FROM COURSE; |  | 查询到新数据 | | 8 | COMMIT; |  | 提交成功 |   2. 带有排他锁的读操作的事务  两个事务操作按照以下顺序执行，并查看事务执行结果，填入下表：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 事务A | 事务B | 执行结果 | | 1 | BEGIN TRANSACTION |  | 执行成功 | | 2 | SELECT \* FROM COURSE FOR UPDATE; |  | 执行成功 | | 3 |  | BEGIN TRANSACTION | 执行成功 | | 4 |  | UPDATE course SET credit=4 WHERE cno='c4'; | 等待 | | 5 |  | COMMIT; | 因第4步等待，无法执行 | | 6 | COMMIT; |  | 因第4步等待，无法执行 |   两个事务操作按照以下顺序执行，并查看事务执行结果，填入下表：   |  |  |  |  | | --- | --- | --- | --- | | 序号 | 事务A | 事务B | 执行结果 | | 1 |  | BEGIN TRANSACTION | 执行成功 | | 2 |  | UPDATE course SET credit=4 WHERE cno='c4'; | 成功 | | 3 | BEGIN TRANSACTION |  | 执行成功 | | 4 | SELECT \* FROM COURSE FOR UPDATE; |  | 等待 | | 5 |  | COMMIT; | 提交成功，事务A查询成功 | | 6 | COMMIT; |  | 提交成功 | | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  无。  2 实验自我评价。  通过本次数据库实验，把上课所学的理论知识同实际实践相结合，验证了所学知识的同时更加印象深刻。 | | | | |