|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验报告3.1 | | | | | | | | |
| 题目：数据库查询优化与调优 | 姓名 | | 杨世蛟 | | 班级 | | 1905 | |
| 学号 | | 20194652 | | 时间 | | 10.29 | |
| （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 tpch.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     1. 查看执行计划   执行：  EXPLAIN SELECT \*  FROM public. customer  where c\_name='Customer#000103500';  执行计划为：      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：       |  |  |  | | --- | --- | --- | | 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 | 9.223 | 0.308 | | 2 | 9.124 | 0.235 | | 3 | 8.887 | 0.318 | | 4 | 9.126 | 0.298 | | 5 | 9.230 | 0.297 | | 6 | 8.983 | 0.300 | | 7 | 8.750 | 0.256 | | 8 | 8.980 | 0.305 | | 9 | 8.999 | 0.314 | | 10 | 8.776 | 0.265 | | 平均值 | 8.950 | 0.285 |   （2）创建复合索引  1）执行在Part表上的查询  select \*  from tpch.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 tpch.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 | 11.360 | 0.262 | | 2 | 11.439 | 0.258 | | 3 | 11.239 | 0.249 | | 4 | 11.026 | 0.302 | | 5 | 11.039 | 0.259 | | 6 | 11.603 | 0.261 | | 7 | 12.506 | 0.260 | | 8 | 12.561 | 0.258 | | 9 | 11.750 | 0.245 | | 10 | 11.812 | 0.248 | | 平均值 | 11.8 | 0.255 |   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 tpch.part where p\_brand='Brand#53'    顺序扫描  （3）Hash索引  1）执行在Part表上的查询  select \*  from tpch.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 tpch.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 | 10.001 | 0.231 | | 2 | 10.703 | 0.213 | | 3 | 10.123 | 0.312 | | 4 | 10.456 | 0.321 | | 5 | 10.789 | 0.275 | | 6 | 10.123 | 0.304 | | 7 | 10.456 | 0.305 | | 8 | 10.789 | 0.276 | | 9 | 10.123 | 0.278 | | 10 | 10.456 | 0.303 | | 平均值 | 10.555 | 0.280 |   （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 tpch.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 tpch.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 | 10.123 | 0.297 | | 2 | 10.456 | 0.298 | | 3 | 10.789 | 0.299 | | 4 | 11.123 | 0.300 | | 5 | 11.456 | 0.301 | | 6 | 11.789 | 0.302 | | 7 | 11.995 | 0.303 | | 8 | 10.123 | 0.304 | | 9 | 10.456 | 0.305 | | 10 | 10.789 | 0.306 | | 平均值 | 10.700 | 0.301 |   **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 tpch.customer  where c\_name LIKE 'Customer#0001035%';    2）Q2  SELECT \*  FROM tpch.customer  where c\_name LIKE 'Customer%500';    3）Q3  SELECT \*  FROM tpch.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 | 10.390 | 顺序扫描 | 否 | | Q2 | 13.705 | 顺序扫描 | 否 | | Q3 | 10.861 | 顺序扫描 | 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 as c join  orders as o on c.c\_custkey = o.o\_custkey  join lineitem as l on l.l\_orderkey = o.o\_orderkey  where l\_partkey = '123008' and  o\_orderdate between to\_date('1995-11-01', 'YYYY-MM-DD') and  to\_date('1995-11-30', 'YYYY-MM-DD'); | | 执行时间 | 429ms | | 创建索引语句 | create index test1\_partkey\_index on lineitem(l\_partkey); | | 使用索引执行时间 | 0.657 |   （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 tpch.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 | 巨大 | 顺序扫描，物化 | 否 | | Q2 | 0.381 | 索引扫描  反连接 | 是 | | Q3 | 0.365 | 同样的反连接和索引扫描 | 是 | | | | | | | | | |
| 实验思考题：  1 PostgreSQL的索引还有哪些？那些索引如何使用？  bitmap index          2 实验自我评价。  还行 | | | | | | | | |
| 实验报告3.2 | | | | | | | | |
| 题目：数据库视图定义 | | 姓名 | | 杨世蛟 | | 班级 | | 1905 |
| 学号 | | 20194652 | | 时间 | | 10.29 |
| 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 | 18rows |     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';    结果都在上面   |  |  |  | | --- | --- | --- | | 查询 | 执行结果 | 原因分析 | | Q1.1 |  |  | | Q1.2 |  |  | | Q2.1 |  |  | | Q2.2 |  | 失败，因为不同部门 | | Q2.3 |  |  | | Q3.1 |  |  | | Q3.2 |  |  | | Q4.1 |  |  | | Q4.2 |  |  | | Q4.3 |  |  | | Q5.1 |  | groupby不能自动更新  需要触发器来处理相关操作 |   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: | 有依赖，删除失败 | | Cascade | 级联删除 |   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： | 按视图的逻辑进行计算 | | Q2： | 顺序扫描 | | | | | | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  无  2 实验自我评价。  还行 | | | | | | | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验报告3.3 | | | | | | | | |
| 题目：数据库安全性实验 | 姓名 | | 杨世蛟 | | 班级 | | 1905 | |
| 学号 | | 20194652 | | 时间 | | 10.30 | |
| 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';  2 授权  1）授予user2在表全部权限  GRANT ALL PRIVILEGES course上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';    执行结果在上面   |  |  |  | | --- | --- | --- | | 查询 | 是否正常执行 | 原因分析 | | Q3.1 |  |  | | Q3.2 |  |  | | Q3.3 |  | 无权限 | | Q3.4 |  | 无权限 | | Q3.5 |  | 无权限 | | Q4.1 |  |  | | Q4.2 |  | 无权限 | | Q5.1 |  | 无权限 | | Q5.2 |  | 无权限 | | Q6.3 |  |  | | Q6.4 |  |  | | Q7.1 |  | 无权限 | | Q7.2 |  |  | | Q7.3 |  |  | | Q8.1 |  | 无权限 | | | | | | | | | |
| 实验总结：  1 实验中遇到的难点有哪些？  无  2 实验自我评价。  还行 | | | | | | | | |
| 实验报告3.4 | | | | | | | | |
| 题目：数据库完整性 | | 姓名 | | 杨世蛟 | | 班级 | | 1905 |
| 学号 | | 20194652 | | 时间 | | 10.30 |
| 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 | 用address和dept做主键  南湖没有计算机系 |   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 ONUPDATE 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？  否    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    级联删除  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    更新的时候变成了null     |  |  |  | | --- | --- | --- | | 查询 | 执行结果情况 | 原因分析 | | Q1 | 结果在上面 |  | | Q2 |  |  | | Q3 |  |  | | Q4 |  |  |   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 Department2values('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');    不能为空  Q4:Insert into Department2 values('SE', '计算机系', '浑南校区');    主键  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 |  |  | | Q2 |  |  | | Q3 |  |  | | Q4 |  |  | | Q5 |  |  | | Q6 | 没结果 | 因为第五个插入的是department，而不是2 |   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 Department3values('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 |  |  | | Q2 | 失败 | 属性不满足要求 |   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 | 违反约束 | | 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。  还行 | | | | | | | | |