

数据库系统课程上机实验手册

华为 OpenGauss 版

大连理工大学

Dalian University of Technology

目录

第一章 实验介绍.....	3
1.1 关于本实验.....	3
1.2 实验目的.....	3
1.3 内容描述.....	3
第二章 DDL.....	4
2.1 预备知识.....	4
2.2 实验任务.....	4
2.3 SQL 代码与对应结果.....	4
第三章 DML.....	6
3.1 预备知识.....	6
3.2 实验任务.....	6
第四章 数据查询.....	11
4.1 单表查询.....	11
4.2 聚合查询.....	13
4.3 多表查询.....	14
4.4 子查询（必须使用子查询实现）.....	17
4.5 集合查询.....	19
第五章 索引操作.....	22
5.1 预备知识.....	22
5.2 实验任务.....	23
第六章 事务的并发控制.....	25
6.1 预备知识.....	25
6.2 实验任务.....	25
第七章 自我总结.....	28

第一章 实验介绍

1.1 关于本实验

本实验共包含 5 个子实验，从数据准备开始，逐一介绍了 `scoot` 数据库中的 SQL 语法，包括数据查询、数据更新、数据定义和数据控制。

1.2 实验目的

学习 SQL 相关操作, 了解 OpenGauss 系统对索引和事务并发控制的实现。

1.3 内容描述

子实验 2 为数据定义实验，介绍了 DDL 的类型、语法格式和使用场景，帮助读者熟练掌握如何用数据定义语言定义或修改数据库中的对象。

子实验 3 为数据更新实验，通过对 DML 语言基本语法和使用, 帮助读者掌握如何对数据库表中数据进行更新操作, 包括数据插入、数据修改和数据删除。

子实验 4 为数据查询实验，通过基本的 DQL 语言使用，帮助读者掌握从一个或多个表查询数据的操作。

子实验 5 为索引操作，通过基本的索引使用，帮助读者掌握索引的创建和使用。

子实验 6 为事务并发控制，通过对事务并发场景的设计，帮助读者了解 OpenGauss 数据库关系对并发的支持情况。

第二章 DDL

2.1 预备知识

数据库模式定义语言 DDL (Data Definition Language)，是用于描述数据库中要存储的现实世界实体的语言，用来创建数据库中的各种对象——表、视图、索引、同义词、聚簇等

```
create table 表名(  
    字段名 1 类型[(宽度) 约束条件],  
    字段名 2 类型[(宽度) 约束条件],  
    字段名 3 类型[(宽度) 约束条件]  
);  
#类型：使用限制字段必须以什么样的数据类型传值  
#约束条件：约束条件是在类型之外添加一种额外的限制
```

2.2 实验任务

创建 DEPT、BONUS、SALGRADE、EMP 表，关系模式为：

```
DEPT (DEPTNO INT, DNAME VARCHAR(14), LOC VARCHAR(13));  
EMP (EMPNO INT, ENAME VARCHAR(10), JOB VARCHAR(9), MGR INT, HIREDATE DATE, SAL  
    FLOAT, COMM FLOAT, DEPTNO INT);  
BONUS (ENAME VARCHAR(10), JOB VARCHAR(9), SAL INT, COMM INT);  
SALGRADE ( GRADE INT, LOSAL INT, HISAL INT);
```

注意：EMP 表中的 DEPTNO 属性为外键，对应 DEPT 表中的 DEPTNO

2.3 SQL 代码与对应结果

请输入如下代码，验证执行结果是否与答案相符。

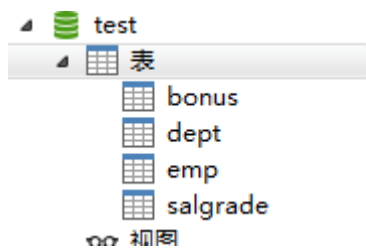
```
CREATE TABLE DEPT (  
    DEPTNO      INT,  
    DNAME       VARCHAR(14),  
    LOC         VARCHAR(13),  
    CONSTRAINT PK_DEPT PRIMARY KEY (DEPTNO)  
);
```

```
CREATE TABLE BONUS (  
  ENAME      VARCHAR(10),  
  JOB        VARCHAR(9),  
  SAL        INT,  
  COMM       INT  
);
```

```
CREATE TABLE SALGRADE (  
  GRADE      INT,  
  LOSAL      INT,  
  HISAL      INT  
);
```

```
CREATE TABLE EMP (  
  EMPNO      INT,  
  ENAME      VARCHAR(10),  
  JOB        VARCHAR(9),  
  MGR        INT,  
  HIREDATE   DATETIME,  
  SAL        FLOAT,  
  COMM       FLOAT,  
  DEPTNO     INT,  
  CONSTRAINT PK_EMP PRIMARY KEY (EMPNO),  
  CONSTRAINT FK_DEPTNO FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO)  
);
```

答案:



第三章 DML

3.1 预备知识

DML 是 Data Manipulation Language 的缩写，意思是数据操纵语言，是指在 SQL 语言中，负责对数据库对象运行数据访问工作的指令集，以 INSERT、UPDATE、DELETE 三种指令为核心，分别代表插入、更新与删除，是开发以数据为中心的应用程序必定会使用到的指令。

1 增

```
INSERT INTO tb_name (col_name1, col_name2) VALUES ('val1', 'val1'), ('val2', 'val2');
```

2 删

```
DELETE FROM table_name [WHERE];
```

3 改

```
UPDATE tb_name SET col_name1 = 'new_val1', col_name2 = 'new_val2' [WHERE];
```

3.2 实验任务

3.2.1 实践题目 1

在 DEPT 表中插入数据 10, 'ACCOUNTING', 'NEW YORK'。

请写出满足查询结果的查询代码。

结果：

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK

代码：INSERT INTO dept VALUES (10, 'ACCOUNTING', 'NEW YORK')

3.2.2 实践题目 2

在 DEPT 表中根据 DEPTNO 修改 LOC 为 BEIJING。

请写出满足查询结果的查询代码。

结果：

DEPTNO	DNAME	LOC
10	ACCOUNTING	BEIJING

代码:

```
UPDATE DEPT
SET LOC = 'BEIJING'
WHERE DEPTNO = 10;
SELECT *
FROM dept
WHERE DEPTNO = 10;
```

3.2.3 实践题目 3

在 DEPT 表中删除 DEPTNO 为 10 的数据,。

请写出满足查询结果的查询代码。

结果:

DEPTNO	DNAME	LOC
(Null)	(Null)	(Null)

代码:

```
DELETE FROM DEPT
WHERE DEPTNO = 10;
SELECT *
FROM dept
WHERE DEPTNO = 10;
```

3.2.4 实践题目 4

在 DEPT 表中插入两条数据 10, 'ACCOUNTING', 'NEW YORK' 和 20, 'RESEARCH', 'DALLAS'。

请写出满足查询结果的查询代码。

结果:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS

代码:

```
INSERT INTO dept VALUES (10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO dept VALUES (20, 'RESEARCH', 'DALLAS');
SELECT *
FROM dept
WHERE DEPTNO = 10 OR DEPTNO =20;
```

3.2.5 实践题目 5

删除 DEPT 表中所有数据。

请写出满足查询结果的查询代码。

结果:

DEPTNO	DNAME	LOC
(Null)	(Null)	(Null)

代码:

```
DELETE FROM DEPT WHERE 1=1;
SELECT * FROM DEPT;
```

3.2.6 实践题目 6

执行如下代码, 并验证代码执行结果是否与图片给出的结果相同。

代码:

```
Delete from DEPT;
INSERT INTO DEPT VALUES
(10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO DEPT VALUES
(20, 'RESEARCH', 'DALLAS');
INSERT INTO DEPT VALUES
(30, 'SALES', 'CHICAGO');
INSERT INTO DEPT VALUES
(40, 'OPERATIONS', 'BOSTON');
```

```
Delete from EMP;
INSERT INTO EMP VALUES
(7369, 'SMITH', 'CLERK', 7566, '1980-12-17', 800, NULL, 20);
INSERT INTO EMP VALUES
(7499, 'ALLEN', 'SALESMAN', 7698, '1981-02-20', 1600, 300, 30);
INSERT INTO EMP VALUES
```



```

(7521, 'WARD', 'SALESMAN', 7698, '1981-02-22', 1250, 500, 30);
INSERT INTO EMP VALUES
(7566, 'JONES', 'MANAGER', 7839, '1981-04-02', 2975, NULL, 20);
INSERT INTO EMP VALUES
(7654, 'MARTIN', 'SALESMAN', 7698, '1981-09-28', 1250, 1400, 30);
INSERT INTO EMP VALUES
(7698, 'BLAKE', 'MANAGER', 7839, '1981-05-01', 2850, NULL, 30);
INSERT INTO EMP VALUES
(7782, 'CLARK', 'MANAGER', 7839, '1981-06-09', 2450, NULL, 10);
INSERT INTO EMP VALUES
(7788, 'SCOTT', 'ANALYST', 7566, '1987-06-13', 3000, NULL, 20);
INSERT INTO EMP VALUES
(7839, 'KING', 'PRESIDENT', NULL, '1981-11-17', 5000, NULL, 10);
INSERT INTO EMP VALUES
(7844, 'TURN...', 'SALESMAN', 7698, '1981-09-08', 1500, 0, 30);
INSERT INTO EMP VALUES
(7876, 'ADAMS', 'CLERK', 7788, '1987-06-13', 1100, NULL, 20);
INSERT INTO EMP VALUES
(7900, 'JAMES', 'CLERK', 7698, '1981-12-03', 950, NULL, 30);
INSERT INTO EMP VALUES
(7934, 'MILLER', 'CLERK', 7782, '1982-01-23', 1300, NULL, 10);

```

```

Delete from SALGRADE;
INSERT INTO SALGRADE VALUES
(1, 700, 1200);
INSERT INTO SALGRADE VALUES
(2, 1201, 1400);
INSERT INTO SALGRADE VALUES
(3, 1401, 2000);
INSERT INTO SALGRADE VALUES
(4, 2001, 3000);
INSERT INTO SALGRADE VALUES
(5, 3001, 9999);

```

结果:

DEPT表:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

EMP表:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7566	1980-12-17 00:00:00.000	800	NULL	20
2	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
3	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
4	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
5	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
6	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30
7	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
8	7788	SCOTT	ANALYST	7566	1987-06-13 00:00:00.000	3000	NULL	20
9	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
10	7844	TURN...	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
11	7876	ADAMS	CLERK	7788	1987-06-13 00:00:00.000	1100	NULL	20
12	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
13	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10

SALGRADE表:

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

BONUS表: 无数据

第四章 数据查询

对于每个查询需求，给出对应的查询语句，并验证是否与给出的结果相同。将查询语句写在“代码：”下面，具体行数可以根据语句行数自行调节。

4.1 单表查询

4.1.1 预备知识

单表查询是最简单的查询方式。所有要查询的信息，都集中在一张表中。也就是说，SQL 语句中的 FROM 子句中只有一个表。我们可以通过以下的几道实践题目来巩固这个知识点。

4.1.2 实践题目 1

查看 EMP 表中部门号为 10 的员工的姓名，职位，参加工作时间，工资。

结果：

	ename	job	hiredate	sal
1	CLARK	MANAGER	1981-06-09 00:00:00.000	2450
2	KING	PRESIDENT	1981-11-17 00:00:00.000	5000
3	MILLER	CLERK	1982-01-23 00:00:00.000	1300

代码：

```
SELECT ENAME, JOB, HIREDATE, SAL
FROM emp
WHERE DEPTNO = 10
```

4.1.3 实践题目 2

查询每个员工每个月拿到的总金额(emp.sal 为工资, emp.comm 为补助)。(提示: gaussdb 中, nvl (ex1, ex2) 表示如果 ex1 为空则返回 ex2)

结果：

	ename	total
1	SMITH	800
2	ALLEN	1900
3	WARD	1750
4	JONES	2975
5	MARTIN	2650
6	BLAKE	2850
7	CLARK	2450
8	SCOTT	3000
9	KING	5000
10	TURNER	1500
11	ADAMS	1100
12	JAMES	950
13	MILLER	1300

代码：

```
SELECT ENAME, NVL (SAL, 0) + NVL (COMM, 0) AS TOTAL
FROM EMP
```

4.1.4 实践题目 3

显示第 3 个字符为大写 O 的所有员工的姓名及工资。

结果：

	ename	sal
1	SCOTT	3000

代码：

```
SELECT ENAME, SAL
FROM emp
WHERE ENAME LIKE '___O%'
```

4.1.5 实践题目 4

显示有补助的员工的姓名，工资，补助。

结果：

	ename	sal	comm
1	ALLEN	1600	300
2	WARD	1250	500
3	MARTIN	1250	1400
4	TURNER	1500	0

代码：

```
SELECT ENAME, SAL, COMM
FROM emp
WHERE COMM IS NOT NULL
```

4.1.6 实践题目 5

显示员工的最高工资和最低工资。
结果：

	最高工资	最低工资
1	5000	800

代码：

```
SELECT MAX(SAL), MIN(SAL)
FROM EMP
```

4.2 聚合查询

4.2.1 预备知识

在查询中，我们经常会遇到这样的问题：求平均值、求最值等等。我们需要使用一些函数如 AVG(), MAX() 等进行计算，也需要通过 GROUP BY 子句来聚合属性。

4.2.2 实践题目 1

显示每种职业的平均工资。
结果：

	job	average
1	ANALYST	3000
2	CLERK	1037.5
3	MANAGER	2758.3333333333
4	PRESIDENT	5000
5	SALESMAN	1400

代码：

```
SELECT JOB, AVG(SAL)
FROM emp
GROUP BY JOB
```

4.2.3 实践题目 2

显示每个部门每种岗位的平均工资和最高工资。

结果：

	deptno	job	average	max
1	20	ANALYST	3000	3000
2	10	CLERK	1300	1300
3	20	CLERK	950	1100
4	30	CLERK	950	950
5	10	MANAGER	2450	2450
6	20	MANAGER	2975	2975
7	30	MANAGER	2850	2850
8	10	PRESIDENT	5000	5000
9	30	SALESMAN	1400	1600

代码：

```
SELECT DEPTNO, JOB, AVG (SAL) , MAX (SAL)
FROM emp
GROUP BY DEPTNO, JOB;
```

4.2.4 实践题目 3

显示平均工资低于 2500 的部门号，平均工资及最高工资。

结果：

	deptno	average	max
1	20	1968.75	3000
2	30	1566.6666666666667	2850

代码：

```
SELECT DEPTNO, AVG (SAL) , MAX (SAL)
FROM EMP
GROUP BY DEPTNO
HAVING AVG (SAL) <2500
```

4.3 多表查询

4.3.1 预备知识

在大部分情况下，我们所需要的信息并不仅仅包含在一张表中。我们首先需要使用 join 连接多个表，然后再进行查询

4.3.2 实践题目 1

显示工资高于 2500 或岗位为 MANAGER 的所有员工的姓名，工资，职位，和部门号。
结果：

	ename	sal	job	deptno
1	JONES	2975	MANAGER	20
2	BLAKE	2850	MANAGER	30
3	CLARK	2450	MANAGER	10
4	SCOTT	3000	ANALYST	20
5	KING	5000	PRESIDENT	10

代码：

```
SELECT ENAME,SAL,JOB,EMP.DEPTNO
FROM emp JOIN DEPT ON EMP.deptno = DEPT.deptno
WHERE SAL>2500 OR JOB ='MANAGER'
```

4.3.3 实践题目 2

排序显示所有员工的姓名，部门号，工资（以部门号升序，工资降序，雇用日期升序显示）。

结果：

	ename	deptno	sal
1	KING	10	5000
2	CLARK	10	2450
3	MILLER	10	1300
4	SCOTT	20	3000
5	JONES	20	2975
6	ADAMS	20	1100
7	SMITH	20	800
8	BLAKE	30	2850
9	ALLEN	30	1600
10	TURNER	30	1500
11	WARD	30	1250
12	MARTIN	30	1250
13	JAMES	30	950

代码：

```
SELECT ENAME,DEPTNO,SAL
FROM emp
ORDER BY DEPTNO ASC,SAL DESC, HIREDATE ASC
```

4.3.4 实践题目 3

采用自然连接的原理显示部门名以及相应的员工姓名。（Sql server 不支持 NATURAL

JOIN 语法。)

结果:

	dname	ename
1	RESEARCH	SMITH
2	SALES	ALLEN
3	SALES	WARD
4	RESEARCH	JONES
5	SALES	MARTIN
6	SALES	BLAKE
7	ACCOUNTING	CLARK
8	RESEARCH	SCOTT
9	ACCOUNTING	KING
10	SALES	TURNER
11	RESEARCH	ADAMS
12	SALES	JAMES
13	ACCOUNTING	MILLER

代码:

```
SELECT DNAME,ENAME  
FROM EMP NATURAL JOIN dept
```

4.3.5 实践题目 4

查询 SCOTT 的上级领导的姓名。

结果:

	ename
1	JONES

代码:

```
SELECT M.ENAME  
FROM emp,EMP AS M  
WHERE EMP.MGR = M.EMPNO AND EMP.ENAME = 'SCOTT'
```

4.3.6 实践题目 5

显示部门的部门名称, 员工名即使部门没有员工也显示部门名称。

结果:

	dname	ename
1	ACCOUNTING	CLARK
2	ACCOUNTING	KING
3	ACCOUNTING	MILLER
4	RESEARCH	SMITH
5	RESEARCH	JONES
6	RESEARCH	SCOTT
7	RESEARCH	ADAMS
8	SALES	ALLEN
9	SALES	WARD
10	SALES	MARTIN
11	SALES	BLAKE
12	SALES	TURNER
13	SALES	JAMES
14	OPERATIONS	NULL

代码：

```
SELECT DNAME, ENAME
FROM DEPT LEFT JOIN emp ON DEPT.DEPTNO = emp.DEPTNO
```

4.4 子查询（必须使用子查询实现）

4.4.1 预备知识

子查询就是指的一个完整的查询语句之中，嵌套若干个不同功能的小查询，从而一起完成复杂查询的一种编写形式。

4.4.2 实践题目 1

显示所有员工的名称、工资以及工资级别。

结果：

	ename	sal	grade
1	SMITH	800	1
2	ADAMS	1100	1
3	JAMES	950	1
4	WARD	1250	2
5	MARTIN	1250	2
6	MILLER	1300	2
7	ALLEN	1600	3
8	TURNER	1500	3
9	JONES	2975	4
10	BLAKE	2850	4
11	CLARK	2450	4
12	SCOTT	3000	4
13	KING	5000	5

代码:

```
SELECT ENAME, SAL, (SELECT GRADE FROM SALGRADE WHERE SAL >= LOSAL AND  
SAL<=HISAL)  
FROM EMP
```

4.4.3 实践题目 2

显示所有工资等级为 2 的员工的姓名, 工资。(分别给出相关子查询和不相关子查询的查询语句)

结果:

	ENAME	SAL
1	WARD	1250
2	MARTIN	1250
3	MILLER	1300

代码:

不相关子查询:

```
SELECT ENAME, SAL  
FROM  
(SELECT ENAME, SAL, (SELECT GRADE FROM SALGRADE WHERE SAL >= LOSAL AND  
SAL<=HISAL)  
FROM EMP )  
WHERE GRADE = 2
```

相关子查询:

```
SELECT ENAME, SAL  
FROM emp  
WHERE (SELECT GRADE FROM SALGRADE WHERE SAL>LOSAL AND SAL<HISAL) = 2
```

4.4.4 实践题目 3

显示职位属于 10 号部门所提供职位范围的员工的姓名, 职位, 工资, 部门号。

结果:

	ename	job	sal	deptno
1	SMITH	CLERK	800	20
2	JONES	MANAGER	2975	20
3	BLAKE	MANAGER	2850	30
4	CLARK	MANAGER	2450	10
5	KING	PRESIDENT	5000	10
6	ADAMS	CLERK	1100	20
7	JAMES	CLERK	950	30
8	MILLER	CLERK	1300	10

代码:

```
SELECT ENAME, JOB, SAL, DEPTNO
```

```
FROM emp
WHERE EMP.JOB IN (SELECT JOB FROM EMP WHERE EMP.DEPTNO = 10)
```

4.4.5 实践题目 4

显示工资比 30 号部门中所有员工的工资都高的员工的姓名，工资和部门号。
结果：

	ename	sal	deptno
1	JONES	2975	20
2	SCOTT	3000	20
3	KING	5000	10

代码：

```
SELECT ENAME, SAL, DEPTNO
FROM emp
WHERE SAL > ALL (SELECT SAL FROM EMP WHERE DEPTNO = 30)
```

4.4.6 实践题目 5

显示包含工资等级为 4 的员工部门信息，具体包括部门名称，部门所在地（使用双层嵌套子查询实现）

结果：

	DNAME	LOC
1	SALES	CHICAGO
2	ACCOUNTING	NEW YORK
3	RESEARCH	DALLAS

代码：

```
SELECT DNAME, LOC
FROM dept
WHERE DEPT.DEPTNO IN (SELECT DEPTNO FROM EMP WHERE (SELECT GRADE FROM
SALGRADE WHERE SAL > LOSAL AND SAL < HISAL ) = 4)
```

4.5 集合查询

4.5.1 预备知识

当两张表的属性相同时，我们可以对它们做一些集合运算，如并集、交集等。

4.5.2 实践题目 1

显示工资高于 2500 或职位为 MANAGER 的员工的姓名，工资和职位（采用 UNION 语法实现）。

结果：

	ename	sal	job
1	BLAKE	2850	MANAGER
2	CLARK	2450	MANAGER
3	JONES	2975	MANAGER
4	KING	5000	PRESIDENT
5	SCOTT	3000	ANALYST

代码：

```
(SELECT ENAME, SAL, JOB
FROM EMP
WHERE SAL > 2500)
UNION
(SELECT ENAME, SAL, JOB
FROM emp
WHERE JOB='MANAGER')
```

4.5.3 实践题目 2

显示工资高于 2500 且职位为 MANAGER 的员工的姓名，工资和职位（采用 INTERSECT 语法实现）。

结果：

	ename	sal	job
1	BLAKE	2850	MANAGER
2	JONES	2975	MANAGER

代码：

```
(SELECT ENAME, SAL, JOB
FROM EMP
WHERE SAL > 2500)
intersect
(SELECT ENAME, SAL, JOB
FROM emp
WHERE JOB='MANAGER')
```

4.5.4 实践题目 3

显示工资高于 2500 但职位不是 MANAGER 的员工的姓名，工资和职位（采用 MINUS 语法

实现)。

结果:

	ename	sal	job
1	KING	5000	PRESIDENT
2	SCOTT	3000	ANALYST

代码:

```
(SELECT ENAME, SAL, JOB
FROM EMP
WHERE SAL > 2500)
minus
(SELECT ENAME, SAL, JOB
FROM emp
WHERE JOB='MANAGER')
```

4.5.5 实践题目 4

显示提供了工资在 2000~5000 之间的所有职位的部门名称

结果:

DNAME
1 RESEARCH

代码:

```
SELECT DNAME
FROM dept
WHERE NOT EXISTS ((SELECT DISTINCT JOB
FROM EMP
WHERE SAL>2000 AND SAL<5000) --找到所有的工资在2000-5000
范围内的员工所从事的工作
EXCEPT
(SELECT DISTINCT JOB --找到主查询中部门号所对提供的
全部工作, 如果这些工作和
--所有的工资在2000-5000范围内的
员工所从事的工作
--完全相同, 那么他们的差集是空集,
不存在差集是空集就说明他们玩去拿相同
FROM EMP E
WHERE DEPT.DEPTNO = E.DEPTNO))
```

第五章 索引操作

5.1 预备知识

索引可以提高数据的访问速度，但同时也增加了插入、更新和删除表的处理时间。所以是否要为表增加索引，索引建立在哪些字段上，是创建索引前必须要考虑的问题。需要分析应用程序的业务处理、数据使用、经常被用作查询条件或者被要求排序的字段来确定是否建立索引。索引相关的 DDL 包括创建索引、删除索引属性和删除索引。

利用索引检索，需要在大数据量的情况下才能够体现出效率差距。因此，本实验需要利用 PLSQL（过程化 SQL）知识自动构建一个含有大量数据的表作为实验对象。PLSQL 相关知识请参考“HCIP-GaussDB-OLTP V1.0 培训教材”第 236 页



- PLSQL是一种高级数据库程序设计语言。
- 全称：
 - 过程化SQL语言 (Procedure Language & Structured Query Language) 。
- 定义：
 - 一组为了完成特定功能的SQL语句的集合。

第236页 版权所有 © 2019 华为技术有限公司



下面给出实验中可能用到的 PLSQL 代码示例，该代码用于向某个表循环插入 100 个数据。其中 count 是一个变量

```
declare
count int;
begin
count := 1;
while count<=100 loop
insert into table_name(id, name, password)
values (count, 'some_name', 'some_name' || count);
count := count + 1;
end loop;
end;
```

5.2 实验任务

创建一个表，然后向该表中循环插入大量数据（1000000 条以上）。然后，对于该表中某个属性建立索引，并进行对比实验，对比利用索引和不利用索引的情况下，完成相同查询任务花费的时间，验证利用索引查询是否真的可以提升查询效率。

注：可以参考 5.4 节的视频。给出每个步骤的 SQL 语句，并对语句执行结果进行截图说明，尤其是其查询时间的区别截图。

1. 创建一个名为 INDEX_TAB 的表

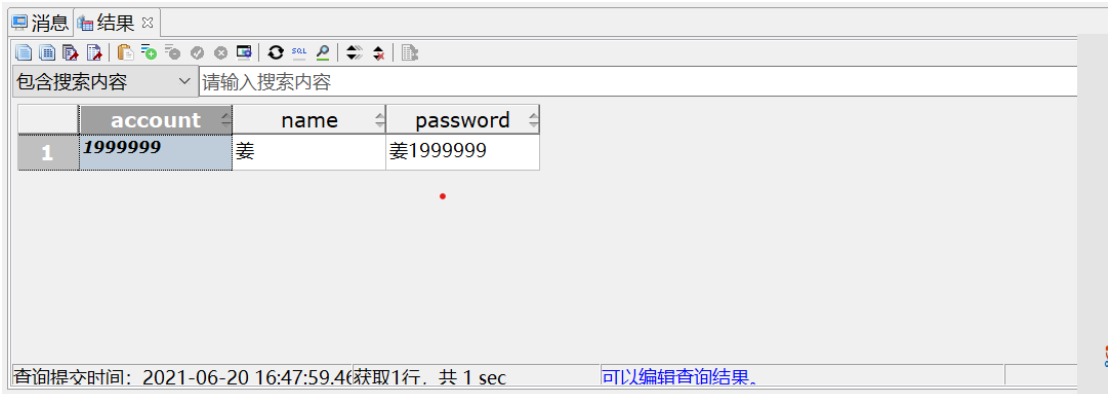
```
create table INDEX_TAB
(account int,
name varchar(15),
password varchar(15)
);
```

2. 向这个表中插入 2000000 条数据

```
DECLARE
COUNT INT;
BEGIN
COUNT := 1;
WHILE COUNT<2000000 LOOP
INSERT INTO index_tab(account,name,password)
VALUES (COUNT,'姜','姜'||COUNT);
COUNT := COUNT+1;
END LOOP;
END;
```

3. 查询其中的某条数据

```
SELECT *
FROM index_tab
WHERE password = '姜1999999'
```



The screenshot shows a database query result window. At the top, there are tabs for '消息' (Messages) and '结果' (Results). Below the tabs is a toolbar with various icons. A search bar is present with the text '包含搜索内容' and a placeholder '请输入搜索内容'. The main area displays a table with the following data:

	account	name	password
1	1999999	姜	姜1999999

At the bottom of the window, there is a status bar that reads: '查询提交时间: 2021-06-20 16:47:59.4(获取1行, 共 1 sec)'. To the right of the status bar is a link that says '可以编辑查询结果.'.

4. 加上索引后，查询其中的某条数据

```
create index idx_on_index_tab
on index_tab(account);
```

消息 结果			
包含搜索内容 请输入搜索内容			
	account	name	password
1	1999999	姜	姜1999999
查询提交时间: 2021-06-20 16:52:12.5 获取1行, 共971 ms 可以编辑查询结果。			

结论: 加了索引好像快了那么 29ms, 但没有快太多

第六章 事务的并发控制

6.1 预备知识

事务并发可能带来的 3 大类问题：

- 1) dirty read(脏读)，一个事务读取了另外一个事务未提交的数据。
- 2) non-repeatable read(不可重复读)，同一事务中，前后读取的数据不一致。
- 3) phantom read(幻读)，类似不可重复读，但针对插入/删除操作。

美国国家标准协会在 SQL 标准中，对于满足事务的隔离性的程度划分为以下四个级别。未提交读、已提交读、可重复读和序列化。不同的隔离级别能够解决的上述问题也不同（参考课程第一节）

6.2 实验任务

通过并发实验设计，验证 OpenGauss 是否存在上述三类问题。根据验证结果给出 OpenGauss 系统默认的事务隔离级别。

注：实验设计可以模仿 10.6 节中的操作，设计两个事务的并发操作场景，判断是否存在对应问题。需要详细的实验设计过程描述和实验结果的截图。

- (1) 验证脏读：先在窗口一中开启事务并修改数据，再在窗口一读取数据；在窗口二中开启事务并读取数据，观察到两个窗口读取到的数据是不一样的。在窗口一提交数据，再在窗口二读数据，观察到两个窗口读取到的数据是一样的，则可验证 gauss 数据库不支持脏读。

窗口一：

```
opengauss=# start transaction;
START TRANSACTION
opengauss=# select * from emp;
   ename | sal
-----+-----
xia     | 2000
wang    | 3000
li      | 4000
(3 rows)

opengauss=# select * from emp;
   ename | sal
-----+-----
xia     | 2000
wang    | 3000
li      | 100
(3 rows)
```

窗口二:

```
opengauss=# start transaction;
START TRANSACTION
opengauss=# update empatest set sal=100 where ename='li';
UPDATE 1
opengauss=# select * from empatest;
  ename | sal
-----+-----
  xia   | 2000
  wang  | 3000
  li    |  100
(3 rows)

opengauss=# commit;
COMMIT
```

- (2) 验证不可重复读: 先在窗口一中开启事务并修改一行数据, 再在窗口二中开启事务并修改同一行数据发现窗口二中该行数据暂时修改不了; 再在窗口一提交事务, 发现窗口二同一行数据可以进行修改了; 再在窗口一查看表中的数据, 为窗口一修改后的数据, 在窗口二查看表中的数据, 为窗口二修改后的数据; 将窗口二提交事务, 发现窗口一的数据变为窗口二修改后的数据了, 说明两个事务同时对同一行数据进行修改时, 会出现阻塞状态 (因为事务会默认对一行数据添加行锁, 一个事物提交数据后, 另一个事务才可对他进行修改), 则可验证 gauss 数据库不支持不可重复读。

窗口一:

```
opengauss=# start transaction;
START TRANSACTION
opengauss=# update empatest set sal=600 where ename='li';
UPDATE 1
opengauss=# commit;
COMMIT
opengauss=# select * from empatest;
  ename | sal
-----+-----
  xia   | 2000
  wang  | 3000
  li    |  600
(3 rows)

opengauss=# select * from empatest;
  ename | sal
-----+-----
  xia   | 2000
  wang  | 3000
  li    |  700
(3 rows)
```

窗口二:

```

opengauss=# start transaction;
START TRANSACTION
opengauss=# update empset set sal=700 where ename='li';
UPDATE 1
opengauss=# select * from empset;
  ename | sal
-----+-----
 xia   | 2000
 wang  | 3000
  li   |  700
(3 rows)

opengauss=# commit;
COMMIT

```

- (3) 验证幻读：先在窗口一中开启事务并修改一行数据，再在窗口二中开启事务并修改另一行数据，发现此时并没有出现阻塞状态，说明 `gauss` 数据库不是对表加锁，而是对行加锁；再在窗口一和二读取数据，发现两窗口没法读取到对方未提交的数据的；在两窗口中提交事务，再在窗口一和二读取数据，发现读取到的数据是一样的（说明了当读取两个行数据时是可以修改成功的，但是读取不到对方的未提交的数据），则可验证 `gauss` 数据库支持幻读。

窗口一：

```

opengauss=# start transaction;
START TRANSACTION
opengauss=# update empset set sal=1000 where ename='xia';
UPDATE 1
opengauss=# select * from empset;
  ename | sal
-----+-----
 wang  | 3000
  li   |  700
 xia   | 1000
(3 rows)

opengauss=# commit;
COMMIT
opengauss=# select * from empset;
  ename | sal
-----+-----
  li   |  700
 xia   | 1000
 wang  | 1500
(3 rows)

```

窗口二：

```

opengauss=# start transaction;
START TRANSACTION
opengauss=# update empset set sal=1000 where ename='xia';
UPDATE 1
opengauss=# select * from empset;
   ename | sal
-----+-----
 wang   | 3000
  li    |  700
  xia    | 1000
(3 rows)

opengauss=# commit;
COMMIT
opengauss=# select * from empset;
   ename | sal
-----+-----
  li     |  700
  xia     | 1000
 wang    | 1500
(3 rows)

```

第七章 用户权限设置及回收

使用 GRANT 命令进行用户授权包括以下三种场景：

- 将系统权限授权给角色或用户。
- 将数据库对象授权给角色或用户。
- 将角色或用户的权限授权给其他角色或用户。

7.1 将系统权限授权给用户或者角色

任务：用自己的姓名全拼代替“joe”完成下述操作，并将操作过程截屏截图：

```

[omm@jiang ~]$ gsql -d postgres -p 26000
gsql ((openGauss 1.1.0 build 392c0438) compiled at 2020-12-31 20:08:06 commit 0 last mr )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

postgres=# grant all privileges to JiangPeifeng;
ALTER ROLE
postgres=#

```

步骤 1，启动服务器，再使用 gsql 客户端以管理员用户身份连接 postgres 数据库，假设端口号为 26000。

```
gsql -d postgres -p 26000 -r
```

步骤 2，创建名为 joe 的用户，并将 sysadmin 权限授权给 joe。

```
postgres=# CREATE USER joe PASSWORD 'Bigdata@123';
```

```
CREATE ROLE
```

```
postgres=# GRANT ALL PRIVILEGES TO joe;
```

```
ALTER ROLE
```

7.2 将数据库对象授权给角色或用户

任务：用自己的姓名全拼代替“joe”完成下述操作，并将操作过程截屏截图：

```
postgres=# REVOKE ALL PRIVILEGES FROM JiangPeifeng;
ALTER ROLE
postgres=# CREATE SCHEMA tpcds;
ERROR: schema "tpcds" already exists
postgres=# CREATE TABLE tpcds.reason
(
    r_reason_sk          INTEGER          NOT NULL,
    r_reason_id          CHAR(16)         NOT NULL,
    r_reason_desc        VARCHAR(20)
);
postgres=# postgres=# postgres=# postgres=# postgres=# ERROR: relation "reason" already exists
postgres=# GRANT USAGE ON SCHEMA tpcds TO JiangPeifeng;
GRANT
postgres=# GRANT ALL PRIVILEGES ON tpcds.reason TO JiangPeifeng;
GRANT
postgres=# GRANT select (r_reason_sk,r_reason_id,r_reason_desc),update (r_reason_desc) ON tpcds.reason TO JiangPeifeng;
GRANT
postgres=# GRANT create,connect on database postgres TO JiangPeifeng WITH GRANT OPTION;
GRANT
postgres=# CREATE ROLE tpcds_manager PASSWORD 'Bigdata@123';
ERROR: role "tpcds_manager" already exists
postgres=# GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
GRANT
postgres=#
```

步骤 1，撤销 joe 用户的 sysadmin 权限，然后创建 tpcds 模式，并给 tpcds 模式下创建一张 reason 表。

```
postgres=# REVOKE ALL PRIVILEGES FROM joe;
```

```
ALTER ROLE
```

```
postgres=# CREATE SCHEMA tpcds;
```

```
CREATE SCHEMA
```

```
postgres=# CREATE TABLE tpcds.reason
```

```
(
```

```
    r_reason_sk          INTEGER          NOT NULL,
```

```
    r_reason_id          CHAR(16)         NOT NULL,
```

```
    r_reason_desc        VARCHAR(20)
```

```
);
```

```
CREATE TABLE
```

步骤 2，将模式 tpcds 的使用权限和表 tpcds.reason 的所有权限授权给用户 joe。

```
postgres=# GRANT USAGE ON SCHEMA tpcds TO joe;
```

```
GRANT
```

```
postgres=# GRANT ALL PRIVILEGES ON tpcds.reason TO joe;
```

```
GRANT
```

授权成功后，joe 用户就拥有了 tpcds.reason 表的所有权限，包括增删改查等权限。

步骤 3，将 tpcds.reason 表中 r_reason_sk、r_reason_id、r_reason_desc 列的查询权限，r_reason_desc 的更新权限授权给 joe。

```
postgres=# GRANT select (r_reason_sk,r_reason_id,r_reason_desc),update (r_reason_desc)
ON tpcds.reason TO joe;
```

```
GRANT
```

步骤 4，将数据库 postgres 的连接权限授权给用户 joe，并给予其在 postgres 中创建 schema 的权限，而且允许 joe 将此权限授权给其他用户。

```
postgres=# GRANT create,connect on database postgres TO joe WITH GRANT OPTION;
```

```
GRANT
```

步骤 5, 创建角色 `tpcds_manager`, 将模式 `tpcds` 的访问权限授权给角色 `tpcds_manager`, 并授予该角色在 `tpcds` 下创建对象的权限, 不允许该角色中的用户将权限授权给其人。

```
postgres=# CREATE ROLE tpcds_manager PASSWORD 'Bigdata@123';
CREATE ROLE
postgres=# GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
GRANT
```

7.3 将用户或者角色的权限授权给其他用户或角色

任务: 用自己的姓名全拼代替“joe”完成下述操作, 并将操作过程截屏截图:

```
postgres=# CREATE ROLE manager PASSWORD 'Bigdata@123';
CREATE ROLE
postgres=# GRANT joe TO manager WITH ADMIN OPTION;
GRANT ROLE
postgres=# CREATE ROLE senior_manager PASSWORD 'Bigdata@123';
CREATE ROLE
postgres=# GRANT manager TO senior_manager;
GRANT ROLE
postgres=#
```

步骤 1, 创建角色 `manager`, 将 `joe` 的权限授权给 `manager`, 并允许该角色将权限授权给其他人。

```
postgres=# CREATE ROLE manager PASSWORD 'Bigdata@123';
CREATE ROLE
postgres=# GRANT joe TO manager WITH ADMIN OPTION;
GRANT ROLE
```

步骤 2, 创建用户 `senior_manager`, 将用户 `manager` 的权限授权给该用户。

```
postgres=# CREATE ROLE senior_manager PASSWORD 'Bigdata@123';
CREATE ROLE
postgres=# GRANT manager TO senior_manager;
GRANT ROLE
```

7.4 权限回收

任务: 用自己的姓名全拼代替“joe”完成下述操作, 并将操作过程截屏截图:

```

DROP ROLE
postgres=# REVOKE JiangPeifeng FROM manager;
WARNING: role "manager" is not a member of role "jiangpeifeng"
REVOKE ROLE
postgres=# REVOKE manager FROM senior_manager;
REVOKE ROLE
postgres=# DROP USER manager;
DROP ROLE
postgres=# REVOKE ALL PRIVILEGES ON tpceds.reason FROM JiangPeifeng;
REVOKE
postgres=# REVOKE ALL PRIVILEGES ON SCHEMA tpceds FROM JiangPeifeng;
REVOKE
postgres=# REVOKE USAGE,CREATE ON SCHEMA tpceds FROM tpceds_manager;
REVOKE
postgres=# DROP ROLE tpceds_manager;
DROP ROLE
postgres=# DROP ROLE senior_manager;
DROP ROLE
postgres=# DROP USER JiangPeifeng CASCADE;
DROP ROLE
postgres=# █

```

步骤 1，撤销权限，并清理用户。

```

postgres=# REVOKE joe FROM manager;
REVOKE ROLE
postgres=# REVOKE manager FROM senior_manager;
REVOKE ROLE
postgres=# DROP USER manager;
DROP ROLE
postgres=# REVOKE ALL PRIVILEGES ON tpceds.reason FROM joe;
REVOKE
postgres=# REVOKE ALL PRIVILEGES ON SCHEMA tpceds FROM joe;
REVOKE
postgres=# REVOKE USAGE,CREATE ON SCHEMA tpceds FROM tpceds_manager;
REVOKE
postgres=# DROP ROLE tpceds_manager;
DROP ROLE
postgres=# DROP ROLE senior_manager;
DROP ROLE
postgres=# DROP USER joe CASCADE;
DROP ROLE

```

注意：实验完成后请尽量清理本实验的对象，以免影响与其它实验产生冲突。

第八章 自我总结

第一个亮点：真的是很有意思，现在知道了删库跑路是需要权限的，并且模拟了一遍作为 DBA 是如何授予权限和剥夺权限的。以后如果从事数据库管理的工作的话知道该怎么授予和剥夺员工的权限了。

第二个亮点：我困惑了好久关于为什么数据库 SQL 语言里不能使用循环。现在我知道其实循环也是可以的，而且还可以像实验六那样声明出一个临时变量来。这样的话可以使用 while 循环来快速创建表。

第三个亮点：第三个亮点是关于索引的。索引好像也每快出多少。可能是因为我设的元素个数是 2000000，而查找的恰好是最后一个的原因。但是别的同学的实验结果显示还是要快出来不少的。

最后，感谢老师安排这次实验，让我能够巩固一下有关 DDL 和 DCL 的语句。现在我已经经历了 DDL、DML、DCL 的语句练习，感觉力量涌上来了。