**暨南大学本科实验报告专用纸**

课程名称 《数据库系统》 成绩评定

实验项目名称 指导教师雷小林 、魏林锋

实验项目编号 实验项目类型 综合 实验地点 N117

学生姓名 金鸿正 学号 2021102560

学院 信息科学技术学院 系 计算机 专业 软件工程

实验时间 年 月 日 午～ 月 日 午 温度 ℃湿度

1 数据库定义与操作语言实验

1.1 数据库定义语言

1.1.1 实验目的

理解和掌握数据库 DDL 语言，能够熟练地使用 SQL DDL 语句创建、修改和删除数据库、模式和基本表。

1.1.2 实验内容和要求

理解和掌握 SQL DDL 语句的语法，特别是各种参数的具体含义和使用方法；使用 SQL 语句创建、修改和删除数据库、模式和基本表。掌握 SQL 语句常见语法错误的调试方法。

1.1.3 实验重点和难点

实验重点：创建数据库、基本表。

实验难点：创建基本表时，为不同的列选择合适的数据类型，正确创建表级和列级完整性约束，如列值是否允许为空、主码和完整性限制等。注意：数据完整性约束可以在创建基本表时定义，也可以先创建表然后定义完整性约束。由于完整性约束的限制，被引用的表要先创建。

1.1.4 实验内容记录

1.1.4.1 实验内容简述

本实验建立对 employees 数据库的认识，分析各个表的关系和约束，分析表中各个列的语意。

1.1.4.2 数据库结构

数据库名：employees

departments 部门表

employees 雇员表

dept\_emp 雇员门表

dept\_manager 部门经理表

titles 职位表

salaries 工资

CREATE TABLE employees (

emp\_no INT NOT NULL,

birth\_date DATE NOT NULL,

first\_name VARCHAR(14) NOT NULL,

last\_name VARCHAR(16) NOT NULL,

gender ENUM ('M','F') NOT NULL,

hire\_date DATE NOT NULL,

PRIMARY KEY (emp\_no)

);

CREATE TABLE departments (

dept\_no CHAR(4) NOT NULL,

dept\_name VARCHAR(40) NOT NULL,

PRIMARY KEY (dept\_no),

UNIQUE KEY (dept\_name)

);

CREATE TABLE dept\_manager (

emp\_no INT NOT NULL,

dept\_no CHAR(4) NOT NULL,

from\_date DATE NOT NULL,

to\_date DATE NOT NULL,

FOREIGN KEY (emp\_no) REFERENCES employees (emp\_no) ON DELETE CASCADE,

FOREIGN KEY (dept\_no) REFERENCES departments (dept\_no) ON DELETE CASCADE,

PRIMARY KEY (emp\_no,dept\_no)

);

CREATE TABLE dept\_emp (

emp\_no INT NOT NULL,

dept\_no CHAR(4) NOT NULL,

from\_date DATE NOT NULL,

to\_date DATE NOT NULL,

FOREIGN KEY (emp\_no) REFERENCES employees (emp\_no) ON DELETE CASCADE,

FOREIGN KEY (dept\_no) REFERENCES departments (dept\_no) ON DELETE CASCADE,

PRIMARY KEY (emp\_no,dept\_no)

);

CREATE TABLE titles (

emp\_no INT NOT NULL,

title VARCHAR(50) NOT NULL,

from\_date DATE NOT NULL,

to\_date DATE,

FOREIGN KEY (emp\_no) REFERENCES employees (emp\_no) ON DELETE CASCADE,

PRIMARY KEY (emp\_no,title, from\_date)

);

CREATE TABLE salaries (

emp\_no INT NOT NULL,

salary INT NOT NULL,

from\_date DATE NOT NULL,

to\_date DATE NOT NULL,

FOREIGN KEY (emp\_no) REFERENCES employees (emp\_no) ON DELETE CASCADE,

PRIMARY KEY (emp\_no, from\_date)

);

1.1.5 思考

* SQL 语法规定，双引号括定的符号串为对象名称，单引号括定的符号串为常量字符串，那么什么情况下需要用双引号来界定对象名呢？请实验验证。
* 数据库对象的完整引用是“服务器名.数据库名.模式名.对象名”，但通常可以省略服务器名和数据库名，甚至模式名，直接用对象名访问对象即可。请设计相应的实验验证基本表及其列的访问方法。

1.2 数据基本查询实验

1.2.1 实验目的

掌握 SQL 程序设计基本规范，熟练运用 SQL 语言实现数据基本查询，包括单表查询、分组统计查询和连接查询.

1.2.2 实验内容和要求

针对 TPC-H 数据库设计各种表单查询 SQL 语句、分组统计查询语句；设计单个表针对自身的连接查询，设计多个表的连接查询。理解和掌握 SQL 查询语句各个字句的特点和作用，按照 SQL 程序设计规范写出具体的 SQL 查询语句，并调试通过。

说明：简单地说，SQL 程序设计规范包含 SQL 关键字大写、表名、属性名、存储过程名等标识符大小写混合、SQL 程序书写缩紧排列等变成规范。

1.2.3 实验重点和难点

实验重点：分组统计查询、单表自身连接查询、多表连接查询。

实验难点：区分元组过滤条件和分组过滤条件；确定连接属性，正确设计连接条件。

1.2.4 实验内容记录

1.2.4.1 单表查询（查询）

查询部门信息。

SELECT dept\_no, dept\_name

FROM departments;

1.2.4.2 单表查询（选择）

查询雇佣日期为 1990-01-01 的所有男性员工信息。

SELECT \*

FROM employees

WHERE hire\_date='1990-01-01'

AND gender='M';

1.2.4.3 不带分组过滤条件的分组统计查询

查询所有员工的工资总额。

SELECT emp\_no, SUM(salary)

FROM salaries

GROUP BY emp\_no;

1.2.4.4 带分组过滤条件的分组统计查询

查询工资总额不低于 100 万的所有员工的工资总额。

SELECT emp\_no, SUM(salary)

FROM salaries

GROUP BY emp\_no

HAVING SUM(salary) >= 1000000;

1.2.4.5 两表连接查询（普通连接）

查询所有 1990 年入职的员工职位。

SELECT DISTINCT e.emp\_no, first\_name, last\_name, hire\_date, title

FROM employees e, titles t

WHERE e.emp\_no=t.emp\_no

AND hire\_date

BETWEEN '1990-01-01'

AND '1990-12-31';

1.2.5 思考

不在 GROUP BY 子句中出现的属性，是否可以出现在 SELECT 子句中？请举例并上机验证。

析：

GROUP BY 用于按相同属性分组，用来细化聚集函数的作用对象。使用 GROUP BY 子句后，查询结果集中每个分组只有一个元组，因此对于没有出现在 GROUP BY 子句中的属性，在 SELECT 子句中出现会出现问题。举例如：

查询所有员工的总工资

SELECT emp\_no, SUM(salary), salary

FROM salaries

GROUP BY emp\_no;

+--------+-------------+--------+

| emp\_no | SUM(salary) | salary |

+--------+-------------+--------+

| 10001 | 1281612 | 60117 |

| 10002 | 413127 | 65828 |

| 10003 | 301212 | 40006 |

| 10004 | 904196 | 40054 |

| 10005 | 1134585 | 78228 |

| 10006 | 606179 | 40000 |

对于没有出现在 GROUP　BY 中的属性，MYSQL 选取了分组中的第一个元组值（尽管没有实际意义）。

请举例说明分组统计查询中 WHERE 和 HAVING 有何区别？

两者区别在于作用对象不同，WHERE 作用于基本表或视图，HAVING 作用于组。举例如：

SELECT emp\_no, SUM(salary)

FROM salaries

GROUP BY emp\_no

HAVING SUM(salary)>1000000;

+--------+-------------+

| emp\_no | SUM(salary) |

+--------+-------------+

| 10001 | 1281612 |

| 10005 | 1134585 |

| 10009 | 1409122 |

| 10018 | 1098241 |

| 10021 | 1029743 |

| 10050 | 1067848 |

SELECT emp\_no, SUM(salary)

FROM salaries

WHERE SUM(salary)>1000000

GROUP BY emp\_no;

ERROR 1111 (HY000): Invalid use of group function

WHERE 子句中不能使用聚集函数作为条件表达式。

连接查询速度是影响关系数据库性能的关键因素。请讨论如何提高连接查询速度，并进行实验验证。

连接操作中最慢的是嵌套循环算法，如果有序，或者有索引，则可以进行优化。

如果连接条件只有等值比较，则容易优化（有序双指针比较）；

如果连接条件能够建立索引，且事先已建立索引，则容易优化；

举例：

查询在 1992-01-01 入职的员工及其所在部门信息。

SELECT COUNT(\*) FROM (

SELECT e.emp\_no, d.dept\_name, e.hire\_date

FROM employees e, dept\_emp de, departments d

WHERE e.emp\_no=de.emp\_no

AND de.dept\_no=d.dept\_no

AND e.hire\_date='1992-01-01'

) AS C;

+----------+

| COUNT(\*) |

+----------+

| 278 |

+----------+

1 row in set (0.26 sec)

SELECT COUNT(\*) FROM (

SELECT e.emp\_no, d.dept\_name, e.hire\_date

FROM employees e, dept\_emp de, departments d

WHERE e.emp\_no=de.emp\_no

AND de.dept\_no=d.dept\_no

AND d.dept\_name='Finance'

) AS C;

+----------+

| COUNT(\*) |

+----------+

| 17346 |

+----------+

1 row in set (0.02 sec)

可以看出，对于存在 UNIQUE 完整性限制的属性，连接速度提升非常大。

1.3 数据高级查询实验

1.3.1 实验目的

掌握 SQL 嵌套查询和集合查询等各种高级查询的设计方法等。

1.3.2 实验内容和要求

针对 TPC-H 数据库，正确分析用户查询要求，设计各种嵌套查询和集合查询。

1.3.3 实验重点和难点

实验重点：嵌套查询。

实验难点：相关子查询、多层 EXIST 嵌套查询。

1.3.4 实验内容记录

1.3.4.1 IN 嵌套查询

查询财务部门经理的工资记录。

SELECT s.emp\_no, first\_name, last\_name, salary,

s.from\_date, s.to\_date

FROM salaries s, employees e

WHERE s.emp\_no=e.emp\_no

AND s.emp\_no IN (

SELECT dm.emp\_no

FROM dept\_manager dm, departments d

WHERE d.dept\_name='Finance'

AND d.dept\_no=dm.dept\_no

);

1.3.4.2 单层 EXISTS 嵌套查询

注：带 EXISTS 谓词的子查询不一定能被其它形式的子查询等价替换。

统计（曾经）在财务部门的员工数量。

使用 EXISTS 谓词可以理解为找出这样的员工，该员工存在部门为财务的元组。

SELECT COUNT(DISTINCT de.emp\_no)

FROM dept\_emp de

WHERE EXISTS (

SELECT \*

FROM departments d

WHERE d.dept\_name='Finance'

AND d.dept\_no=de.dept\_no

);

如果用两表连接查询，可以实现相似的语义（语义存在细微区别，前者是存在语义，后者是计数语义）。

SELECT COUNT(DISTINCT de.emp\_no)

FROM dept\_emp de, departments d

WHERE de.dept\_no=d.dept\_no

AND d.dept\_name='Finance';

1.3.4.3 两层 EXISTS 嵌套查询

注意：SQL 查询的结果是一个集合，实际上就是一层全称谓词。如果给出的条件中再出现全称谓词，则可以转换为存在谓词进行查询。

注意：嵌套查询如果进行了相关查询，则实际上就是一个外层和内层进行笛卡尔积然后对结果集合进行筛选（嵌套内层进行筛选）的过程。

查询（曾经）在所有部门待过的员工。

转换为，不存在这样的部门，员工没有待过。

SELECT emp\_no

FROM employees e

WHERE NOT EXISTS (

SELECT \*

FROM departments d

WHERE NOT EXISTS (

SELECT \*

FROM dept\_emp de

WHERE d.dept\_no=de.dept\_no

AND e.emp\_no=de.emp\_no

)

);

1.3.4.4 FROM 子句中的嵌套查询

FROM 子句中进行嵌套是将嵌套块生成临时表进行查询。

统计所有员工的平均总工资。

首先需要一张员工的总工资表，然后根据总工资表进行平均数计算。

SELECT SUM(total\_salary)/COUNT(DISTINCT total\_emp\_no) avg

FROM (

SELECT emp\_no total\_emp\_no, SUM(salary) total\_salary

FROM salaries

GROUP BY emp\_no

)

AS total;

+-------------+

| avg |

+-------------+

| 604887.4671 |

+-------------+

1.3.4.5 集合查询（交、并、差）

与集合运算的要求一致，集合查询时进行集合运算的集合必须具有相同的列数以及对应有相同的数据类型。

统计财务（Finance）部门和销售（Sales）部门的历史员工数。

SELECT COUNT(DISTINCT emp\_no)

FROM dept\_emp de, departments d

WHERE de.dept\_no=d.dept\_no

AND d.dept\_name IN ('Finance')

UNION

SELECT COUNT(DISTINCT emp\_no)

FROM dept\_emp de, departments d

WHERE de.dept\_no=d.dept\_no

AND d.dept\_name IN ('Sales');

+------------------------+

| COUNT(DISTINCT emp\_no) |

+------------------------+

| 17346 |

| 52245 |

+------------------------+

查询在财务（Finance）部门和销售（Sales）部门都工作过的员工。

注：MYSQL 目前不支持交运算。

SELECT DISTINCT de.emp\_no

FROM dept\_emp de, departments d

WHERE de.dept\_no=d.dept\_no

AND d.dept\_name='Finance'

INTERSECT

SELECT DISTINCT de.emp\_no

FROM dept\_emp de, departments d

WHERE de.dept\_no=d.dept\_no

AND d.dept\_name='Sales';

1.3.5 思考

试分析什么类型的查询可以用连接查询实现，什么类型的查询只能用嵌套查询实现？

连接查询在算法实现上，一定可以通过嵌套循环实现，因此连接查询一定能被嵌套查询等价替换。由于嵌套查询在嵌套条件上提供了一些语义，因此嵌套查询不一定能被转换为连接查询。

从语义上看，连接查询首先是对两个表进行笛卡尔积运算（不带条件的），然后对得到的元组集合进行条件筛选，因此连接查询适用于需要将多个表的属性关联起来的查询需求。嵌套查询则是外层表和内层表进行嵌套罗列，嵌套时可以使用 IN、ANY、ALL、EXISTS 谓词。这些谓词的使用使得嵌套查询语义不一定能被连接查询实现。

连接查询在算法实现上不一定需要通过嵌套循环实现，因此效率往往高于嵌套查询。如：

统计在财务（Finance）部门的员工人数。

SELECT COUNT(e.emp\_no)

FROM employees e, dept\_emp de, departments d

WHERE e.emp\_no=de.emp\_no

AND de.dept\_no=d.dept\_no

AND d.dept\_name='Finance';

+-----------------+

| COUNT(e.emp\_no) |

+-----------------+

| 17346 |

+-----------------+

1 row in set (0.02 sec)

1 row in set (0.02 sec)

SELECT COUNT(e.emp\_no)

FROM employees e

WHERE EXISTS (

SELECT \*

FROM dept\_emp de

WHERE EXISTS (

SELECT \*

FROM departments d

WHERE e.emp\_no=de.emp\_no

AND de.dept\_no=d.dept\_no

AND d.dept\_name='Finance'

)

);

+-----------------+

| COUNT(e.emp\_no) |

+-----------------+

| 17346 |

+-----------------+

1 row in set (0.40 sec)

可以看出来，三个表的连接查询相比三个表的嵌套查询在速度上要快很多。

试分析不相关子查询和相关子查询的区别。

两者区别在于子查询是否引用外层查询的属性。若不引用，则两个查询完全隔离，相当于是两个独立的查询。

从实现上看，相关子查询由于存在相关引用，因此子查询被执行多次，而不相关子查询由于是一个独立的查询，因此只执行一次。

1.4 数据更新实验

1.4.1 实验目的

熟悉数据库的数据更新操作，能够使用 SQL 语句对数据库进行数据的插入、修改、删除操作。

1.4.2 实验内容和要求

针对 employees 数据库设计单元组插入、批量修改插入、修改数据和删除数据等 SQL 语句。理解和掌握 INSERT、UPDATE 和 DELETE 语法结构的各个组成成分，结合嵌套 SQL 子查询，分别设计几种不同形式的插入、修改和删除的语句，并调试成功。

1.4.3 实验重点和难点

实验重点：插入、修改和删除数据的 SQL。

实验难点：与嵌套 SQL 子查询相结合的插入、修改和删除数据的 SQL 语句；利用一个表的数据来插入、修改和删除另外一个表的数据。

1.4.4 实验内容记录

1.4.4.1 INSERT 基本语句

插入一条雇员记录。

INSERT INTO employees

VALUES(500000,'1990-05-06','San','Zhang','M','2010-09-05');

INSERT INTO employees(emp\_no,birth\_date,first\_name,last\_name,gender,hire\_date)

VALUES(500000,'1990-05-06','San','Zhang','M','2010-09-05');

1.4.4.2 批量数据 INSERT 语句

创建一个男员工表和女员工表，并将所有男员工插入到男员工表，将所有女员工插入到女员工表。

先创建两张表。

CREATE TABLE employees\_female

AS SELECT \*

FROM employees

WHERE 1=2;

CREATE TABLE employees\_male

AS SELECT \*

FROM employees

WHERE 1=2;

注：这里的条件永不满足，用来复制表模式，而不复制任何数据。

然后将数据录入两张表。

INSERT INTO employees\_female

SELECT e.\*

FROM employees e

WHERE e.gender='F';

INSERT INTO employees\_male

SELECT e.\*

FROM employees e

WHERE e.gender='M';

创建一个部门员工人数统计表，并统计所有部门的历史员工人数。

先创建表。

CREATE TABLE dept\_emp\_count(

dept\_no CHAR(4),

emp\_count INT);

然后批量录入数据。

INSERT INTO dept\_emp\_count

SELECT de.dept\_no, COUNT(DISTINCT emp\_no)

FROM dept\_emp de

GROUP BY de.dept\_no;

1.4.4.3 UPDATE 语句（修改部分记录的部分列值）

将所有 1985 年及以前入职，职位为 Assistant Engineer 的员工的职位修改为 Engineer 。

UPDATE titles t

SET t.title='Engineer'

WHERE t.emp\_no IN (

SELECT e.emp\_no

FROM employees e

WHERE e.hire\_date < '1986-01-01'

)

AND t.title='Assistant Engineer';

1.4.4.4 UPDATE 语句（利用一个表的数据更新另一个表的数据）

更新部门员工人数统计表（假设表已经存在）。

UPDATE dept\_emp\_count decount

SET decount.emp\_count = (

SELECT COUNT(DISTINCT de.emp\_no)

FROM dept\_emp de

WHERE de.dept\_no=decount.dept\_no

);

1.4.4.5 DELETE 基本语句（删除给定条件的所有记录）

删除 2000 年以前的工资记录。

DELETE FROM salaries

WHERE from\_date < '2000-01-01'

AND to\_date < '2000-01-01';

1.4.5 思考

请分析数据库模式更新和数据更新 SQL 语句的异同。

两者的关键字不同，更新模式使用 ALTER TABLE ，更新数据使用 UPDATE 。

请分析数据库系统除了 INSERT、UPDATE 和 DELETE 等基本的数据更新语句之外，还有哪些可以用来更新数据库基本表数据的 SQL 语句？

如 truncate 。

1.5 视图实验

1.5.1 实验目的

熟悉 SQL 语言有关视图的操作，能够熟练使用 SQL 语句来创建需要的视图，定义数据库外模式，并能使用所创建的视图实现数据管理。

1.5.2 实验内容和要求

针对给定的数据库模式，以及相应的应用需求，创建视图和带 WITH CHECK OPTION 的视图，并验证 WITH CHECK OPTION 选项的有效性。理解和掌握视图消解执行原理，掌握可更新视图和不可更新视图的区别。

1.5.3 实验重点和难点

实验重点：创建视图。

实验难点：可更新的视图和不可更新的视图之区别，WITH CHECK OPTION 的验证。

1.5.4 实验内容记录

1.5.4.1 创建视图（省略视图列名）

创建男性员工视图。

CREATE VIEW v\_emploree\_male AS

SELECT \*

FROM employees e

WHERE e.gender='M';

1.5.4.2 创建视图（不能省略列名的情况）

如果目标列不是单纯的属性名（如聚集函数或表达式），则应该给出列名。

创建部门历史员工总数的视图。

CREATE VIEW v\_dept\_emp\_num(dept\_no, emp\_count) AS

SELECT dept\_no, COUNT(DISTINCT emp\_no)

FROM dept\_emp

GROUP BY dept\_no;

1.5.4.3 创建视图（WITH CHECK OPTION）

WITH CHECK OPTION 使得在对视图进行操作时，会验证操作是否符合视图的条件。

例如，对男性员工视图进行插入操作，插入一个女员工信息：

INSERT INTO

v\_emploree\_male

VALUES(500000,'1990-05-06','San','Zhang','F','2010-09-05');

由男性员工视图没有 WITH CHECK OPTION 语句，因此插入成功，这使得视图的封装性被破坏。

加入选项：

CREATE VIEW v\_emploree\_male\_check AS

SELECT \*

FROM employees e

WHERE e.gender='M'

WITH CHECK OPTION;

再执行插入：

INSERT INTO

v\_emploree\_male\_check

VALUES(500000,'1990-05-06','San','Zhang','F','2010-09-05');

会得到错误：

ERROR 1369 (HY000): CHECK OPTION failed 'employees.v\_emploree\_male\_check'

1.5.4.4 不可更新视图

如果为行列子集视图，则一定是可以更新的（如男性员工视图）。

如果不是，则不一定可以更新，如属性为聚集函数或表达式（如部门员工总数视图）。

尝试修改财务（Finance）部门员工总数为 0 并观察报错。

UPDATE v\_dept\_emp\_num vden

SET emp\_count=0

WHERE vden.dept\_no=(

SELECT d.dept\_no

FROM departments

WHERE d.dept\_name='Finance'

);

得到报错：

ERROR 1288 (HY000): The target table vden of the UPDATE is not updatable

1.5.4.5 删除视图

删除未被引用的视图，直接删除即可。

删除部门员工总数视图和男性员工视图。

DROP VIEW v\_dept\_emp\_num;

DROP VIEW v\_emploree\_male;

删除视图时 CASCADE 语句的作用：

先建立两个视图：

CREATE VIEW v\_e\_1 AS

SELECT emp\_no,birth\_date,gender

FROM employees;

CREATE VIEW v\_e\_2 AS

SELECT emp\_no,gender

FROM v\_e\_1;

直接删除 v\_e\_1，

DROP VIEW v\_e\_1;

对于 MYSQL ，删除成功，但是会引起 v\_e\_2 查询报错。

ERROR 1356 (HY000): View 'employees.v\_e\_2' references invalid table(s) or column(s) or function(s) or definer/invoker of view lack rights to use them

不论使用 RESTRICT 短语还是 CASCADE 短语结果都是一样。

1.5.5 思考

* 请分析视图和基本表在使用方面有哪些异同，并设计相应的例子加以验证。

视图的数据来源于最原始的基本表，也即视图本身不存放数据。

视图不一定都是可以修改的。

* 请具体分析修改基本表的结构对相应的视图会产生何种影响？

修改基本表后，可能导致视图不能正常工作。

先创建一个测试表：

CREATE TABLE d1

SELECT \* FROM departments;

并创建基于测试表的一个视图：

CREATE VIEW v\_d1 AS

SELECT \* FROM d1;

然后调整测试表的结构，这里先尝试调整列的数据类型：

ALTER TABLE d1

MODIFY COLUMN dept\_no INT;

调整会触发警告，数据变为 0 。相应的视图也会同步改变。

如果删除一列：

ALTER TABLE d1

DROP COLUMN dept\_name;

此时查询视图会报告错误。

1.6 索引实验

1.6.1 实验目的

掌握索引设计原则和技巧，能够创建合适的索引以提高数据库查询、统计分析效率。

1.6.2 实验内容和要求

针对给定的数据库模式和具体应用需求，创建唯一索引、函数索引、复合索引等；修改索引；删除索引。设计相应的 SQL 查询验证索引有效性。学习利用 EXPLAIN 命令分析 SQL 查询是否使用了所创建的索引，并能够分析其原因，执行 SQL 查询并估算索引提高查询效率的百分比。要求实验数据集达到 10 万条记录以上的数据量，以便验证索引效果。

1.6.3 实验重点和难点

实验重点：创建索引。

实验难点：设计 SQL 查询验证索引有效性。

1.6.4 实验内容记录

1.6.4.1 创建唯一索引

对员工表的员工号码建立唯一索引

CREATE UNIQUE INDEX idx\_emp\_no

ON employees (emp\_no);

注：主码列在创建表时会自动创建相应的索引。

注：在存在数据时，建立索引需要一定的时间。

1.6.4.2 创建复合索引

对员工表的名字建立复合索引

CREATE INDEX idx\_name

ON employees (first\_name, last\_name);

1.6.4.3 创建长度函数索引

MYSQL 函数索引暂不录入。

1.6.4.4 创建聚簇索引

MYSQL Innodb 不支持单独建立聚簇索引。在此不列出。

1.6.4.5 创建 HASH 索引

对员工的雇佣日期创建 HASH 索引

CREATE INDEX idx\_hash\_hiredate

USING HASH

ON employees (hire\_date);

注：MYSQL 创建 HASH 索引的语法和书中给出的语法稍有不同。

1.6.4.6 修改索引名称

注：修改索引名称通过先删除再添加实现。

1.6.4.7 查询表上的已有索引

查询员工表上的已有索引

SHOW INDEX FROM employees;

1.6.4.8 分析某个 SQL 语句执行时是否使用了索引

分析从员工表上查询某个日期雇佣的员工是否使用了索引。

EXPLAIN SELECT \*

FROM employees

WHERE hire\_date='1990-01-01';

1 row in set (0.00 sec)

1.6.4.9 验证索引效率

首先验证复合索引，在有索引的情况下，统计指定名字的员工数目。

SELECT COUNT(\*)

FROM employees

WHERE first\_name='Georgi'

AND last\_name='Facello';

得到结果为：

+----------+

| COUNT(\*) |

+----------+

| 2 |

+----------+

1 row in set (0.00 sec)

然后去掉复合索引，再次执行，结果为：

+----------+

| COUNT(\*) |

+----------+

| 2 |

+----------+

1 row in set (0.06 sec)

结果显示复合索引提高了查询效率。

然后验证哈希索引，在有索引的情况下，统计指定雇佣日期的员工数目。

SELECT COUNT(\*)

FROM employees

WHERE hire\_date='1990-01-01';

结果为：

+----------+

| COUNT(\*) |

+----------+

| 65 |

+----------+

1 row in set (0.00 sec)

去掉索引，然后在查询，结果为：

+----------+

| COUNT(\*) |

+----------+

| 65 |

+----------+

1 row in set (0.06 sec)

结果显示哈希索引提高了查询效率。

1.6.5 思考

在一个表的多个字段上创建的复合索引，与在相应的每个字段上创建的多个简单索引有何异同？请设计相应的例子加以验证。

可以直接创建两个在名字上的简单索引然后测试即可。

CREATE INDEX idx\_first\_name

ON employees(first\_name);

CREATE INDEX idx\_last\_name

ON employees(last\_name);

EXPLAIN

SELECT COUNT(\*)

FROM employees

WHERE first\_name='Georgi'

AND last\_name='Facello';

1 row in set (0.00 sec)

也即 MYSQL 内部和执行了对多个单列索引合并为一个复合索引的优化，此时效率不受影响。

对于已定义的复合索引，从最左侧开始的列是可用的

这里在已定义了复合索引的情况下只查询第二个条件，发现索引没有被采用，而是进行了全表扫描。

**暨南大学本科实验报告专用纸(附页)**