ECE356 – Database Systems

Lab 3: Advanced MySQL Features

indexes / stored procedures / transaction management

Hua Fan (h27fan@uwaterloo.ca)
Lab Instructor
Fall 2016

Goals

Learn how to do the following:

1. Add indexes to a table.
2. Create and invoke a stored procedure.
3. Use MySQL transactions management service.

Notes:

1. You will use the MySQL command line interface (*mysql* command) or the NetBeans MySQL GUI to complete the tasks in the lab.
2. The exercises in this lab assume that your database instance contains the tables created in previous labs (Employee, Department, Assigned and Project) and the corresponding data.

Marking Scheme

This lab consists of three parts, and the weight on each part is as follows:

- Part 1: Creating indexes (scripts to create the required tables are provided) (30%)
- Part 2: Stored procedure (starting scripts are provided) (40 %)
- Part 3: Transaction management (30%)

Notes:

1. You and your lab partner must save your work and retain a digital copy until the end of the term.
2. You must demonstrate your solution to a TA at the end of the lab.

Part 1:
Creating indexes

Part 1: Creating indexes for a table

This part consists of three steps:

1. Create indexes.
2. List indexes currently in your database.
3. Examine the database query plan before and after creating the indexes for queries.

Step1: Create indexes (1/2)

- The MySQL InnoDB engine does not support HASH indexes, because indexes are expensive to rebuild when stored on disk, and InnoDB is disk-based. Any indexes defined as HASH will be created as BTREE indexes when using the MySQL InnoDB engine
- In this lab, we will use the MySQL "memory" database engine, which supports both B-tree and hash indexes.
- In this part of the lab you will:
  - Write SQL statements to create indexes.
  - Save these CREATE INDEX statements for query 1 and 2(detailed later) in a file called part1a.sql.
  - Execute these statements in the next step of the lab.

Notes:

- To show all supported engines use the command:
  - SHOW ENGINES;
- To switch between DB engines use the command:
  - SET default_storage_engine=<DB-Engine-Name>;

Step1: Create indexes (2/2)

- Assume that the database tables have been created without primary or foreign key constrains.
  - You can run the provided script, createTables.sql, to recreate the tables as expected for this exercise.
- For each query below, decide which attributes should be indexed and select the type of index (BTREE or HASH) that would be the most beneficial to the performance of the query.

  Queries:

  1. Find employees with salaries ranging between 30000 and 40000.
  2. List all the departments (deptID, deptName, location) and the employees that work for each department(empID, empName, job, salary).
- Write the create index statements for the selected attributes.
  - Hint:
    - CREATE INDEX …. ON …. USING ….;

Step 2: List indexes currently in your database

- For this exercise you will:
  - Add primary key and referential integrity to the database tables, as we did in lab 2.
  - Add 1000 rows to the Employee table to make the use of indexes advantageous to the database.
- You can run the provided script, createTablesWithConstrains.sql, to recreate the tables Employee and Department, used in this exercise. The addition of referential integrity and primary key constrains will cause the automatic creation of indexes for columns EmpID and DeptID in the Employee table.
- Check the indexes that exist for each table with the command "show index from table_name".
- Save the output to a file named part1b.txt.

Step 3: Examine the database query plan (1/2)

- The EXPLAIN statement in SQL helps you determine where you should add indexes to tables to improve the execution time of your SQL statements. In the exercise in the next slide you will run this statement before and after creating the indexes on your tables.

Notes:

- You can read more about query optimization with EXPLAIN here:
  - http://dev.mysql.com/doc/refman/5.7/en/using-explain.html

Step 3: Examine the database query plan (2/2)

A. Write the SQL statements for queries 1 and 2.
   1. Find employees with salaries ranging between 30000 and 40000.
   2. List all the departments (deptID, deptName, location) and the employees that work for each department(empID, empName, job, salary).

B. Display the database query plan for each query (with EXPLAIN).

   EXPLAIN <Query 1>;

   EXPLAIN <Query 2>;

C. Create the necessary indexes to improve the execution time of each query (from part 1.a)
   - Hint: You can delete the index created and try creating another one with different type.

D. Display the database query plan for each query, after creating the index (with EXPLAIN)

   EXPLAIN <Query 1>;

   EXPLAIN <Query 2>;

E. Explain the difference between the query plan before and after the creation of indexes.
   - You are expected to comment on the rows count, the indexes that were actually chosen , and the join order.

F. Save the output of B) and D), and your explanation of the difference between them, to a file named part1c.txt

Part 2:
Creating and invoking a stored procedure

Part 2: Creating and invoking a stored procedure

- Stored procedures allow developers to implement application functionality in a module that is stored in the database data dictionary.
- A procedure can be invoked using its name and required parameters.
- Some of the benefits of stored procedures:
  - It facilitates the reuse of program logic.
  - It simplifies the data access.
  - It provides access control (i.e., users may not be given access to database tables; instead users are given access to stored procedures that perform specific operations on these tables).
  - It (slightly) improves the performance (i.e., remove part of the compilation overhead).
- This part consists of four steps:
  1. Create a stored procedure.
  2. Invoke the stored procedure (using CALL statement).
  3. Add error checking to the stored procedure.
  4. Display the output of a stored procedure (given as an "out" parameter).

Step 1: Create a stored procedure

- Write a script to create a stored procedure, name this script part2a.sql (starter code is provided).
- The script will use the CREATE PROCEDURE statement to create a module that raises the salary of an employee by a given percentage. The name of the procedure is increase_salary_proc. The parameters for this procedure are the following:

Parameters	IN/OUT	Data type
inEmpID	IN	INT
inPercentage	IN	DOUBLE(4,2)

- Using the *mysql* command, run this script and create the stored procedure.

Step 2: Invoke the stored procedure

- Using the *mysql* command
  - Invoke the stored procedure with parameters, inEmpID = 45 and inPercentage = 12.50.
  - Issue a SELECT statement on the Employee table to verify that the update has been successful.
  - Save the output of your CALL statement and subsequent SELECT in a file called part2b.out.

Step 3: Add error checking to the stored procedure

- Add a parameter to return an error code indicating one of two possible error conditions (percentage out of range and employee not found).
- Save the modified script in file part2c.sql.
- The parameters for the procedure will now be:

Parameters	IN/OUT	Data type	Value
inEmpID	IN	INT	
inPercentage	IN	DOUBLE(5,2)	
errorCode	OUT	INT	-1: percentage < 0 OR percentage > 100 -2: Employee not in table 0: Success

Notes:
In order to complete this task you will need to become familiar with SQL statements used to

- Declare variables
- Control flow of execution (see commands IF, LEAVE)
- The ROW_COUNT() function

Step 4: Display the output of a stored procedure (given as an "out" parameter)

- Using the *mysql* command, execute the stored procedure, with the following parameters:
  - Percentage equal to 120.00 (out of range)
  - EmpID = 1200 (not in the table)
  - EmpID = 68, percentage = 20.00 (success)
- Display the value of the errorCode parameter after the execution of all three cases. Save your mysql commands and output in a file called partd.out.

Part 3:
Using MySQL transaction management

Part 3: Using MySQL transaction management

This part consists of two steps:

1. Understand the effect of ROLLBACK/COMMIT.
2. Write a stored procedure with a transaction comprising multiple statements.

Notes:
You will be using MySQL InnoDB database engine for this part since this engine provides transaction management and locking services.

Step 1: Understand the effect of ROLLBACK/COMMIT (1/2)

- To execute a group of statements as a transaction, you must set the commit mode of the database appropriately.
  - By default, MySQL runs in auto-commit mode: Changes produced by individual statements are committed to the database immediately, as soon as they execute. In this step we will disable auto-commit by setting the AUTOCOMMIT attribute to 0.
  - While auto-commit is disabled, you begin a transaction with BEGIN and end it with COMMIT.
- If an error occurs during the transaction, you can roll back the affect of all its constituent statements by issuing a ROLLBACK statement.

Notes:
If the autocommit attribute is set to 0 (disabled)
use the statements BEGIN and COMMIT/ROLLBACK to indicate the start and end of a transaction.
If autocommit is set to 1 (enabled),
each SQL statement acts as an individual transaction (it is committed automatically). The documentation for the START TRANSACTION, COMMIT, and ROLLBACK syntax can be found on http://dev.mysql.com/doc/refman/5.7/en/commit.html

Step 1: Understand the effect of ROLLBACK/COMMIT (2/2)

- Write the commands to start a transaction that executes the UPDATE statement given below:
  - UPDATE Employee SET salary = salary + 100000 where empID = 66;
- Check the data in the Employee table (with SELECT statement),
  - Case 1: after you end this transaction with "ROLLBACK",
  - Case 2: after you end this transaction with "COMMIT".
  - Notice the effect of the UPDATE in both cases.
- Save the commands you issued in a file called part3a.sql.

Step 2: A stored procedure with a transaction comprising multiple statements

- Write a stored procedure, named transfer_funds, which transfers funds from one project to another in a single transaction. The parameters for this procedure are shown in the table below.

Parameters	IN/OUT	Data type	value
fromProject	IN	INT	
toProject	IN	INT	
amount	IN	INT	
errorCode	OUT	INT	Set as appropriate.

- The transaction boundaries are declared in the stored procedure
- Execute the procedure and verify that funds are transferred correctly.
- Save the statements needed to create this stored procedure in a file named part3c.sql

Thank You ☺

Acknowledgement

- Special thanks for Tiuley Alguindigue, Mohammad Hamdaqa, Prof. Wojciech Golab and Prof. Paul Ward for their help creating these lab exercises.