Institut für Informationssysteme
Technische Universität Braunschweig

Technische Universität Braunschweig
Institut für Informationssysteme
http://www.ifis.cs.tu-bs.de

Wolf-Tilo Balke, Christoph Lofi, and Joachim Selke

SQL Lab: Assignment 4

(due to 28.1.2010)

General Information

In this assignment you will learn how detailed information about database objects can be retrieved through the database catalog, how to use existing tables as templates for new ones, how to increase query performance through indexes, how to create materialized views, and how to formulate recursive SQL queries.

Task

1. Using the Database Catalog

In 1985, Edgar F. Codd, the inventor of the relational database model, proposed twelve rules to define what is required from a DBMS in order to be considered as being *relational*. In particular, rule 4 requires that a relational database is self-describing. In other words, the database must contain certain system tables that describe the structure of the database itself. These tables containing the structural information of the other tables are usually called the database catalog.

An overview of DB2's database catalog can be found on the following page of the DB2 documentation: <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.sql.ref.doc/doc/r0011297.html>.
DB2 also provides the DESCRIBE TABLE statement, a convenient mechanism to retrieve table definitions:
<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.cmd.doc/doc/r0002019.html>.
To use this statement in Netbeans IDE, it has to be embedded into the ADMIN_CMD procedure, as described here:
<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.sql.rtn.doc/doc/r0023570.html>.

In the schema IMDB (*not* IMDBRAW) of the DBLAB database you will find a cleaned-up version of the IMDb database. Use SQL queries on the system catalog to obtain meta-data of the table IMDB.CAST_AND_CREW. That is, retrieve all information about column definitions (name, data type, length, constraints, check conditions), primary keys, foreign keys (linking to which table and column), unique keys, table constraints, and indexes. Include the statements you used and their respective results in your answer sheet and explain briefly the meaning of every constraint.

Using this information, build a CREATE TABLE statement for creating an equivalent table CAST_AND_CREW with similar keys, constraints, indexes etc. in your own schema. Finally, after creating the table (and checking that the table indeed showed similar meta-data), drop it again.

2. Using Existing Tables as Templates

Use several CREATE TABLE LIKE statements to import tables and data from the IMDB schema into your own schema. Please copy the table structure of the tables TITLE, CAST_AND_CREW, and NAME. Also, copy all data within these tables related to any cinema movie (type = 'film') or TV movie (type = 'TV movie') released between 2007 and 2009. Also, ensure that primary and foreign key constraints are also accordingly created. Please see <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.sql.ref.doc/doc/r0000927.html> for reference on the CREATE TABLE statement.

3. Creating Indexes

Write an SQL query on the tables created in the last exercise for counting all females involved in any cinema movie in 2009. What indexes should be created to speed up query processing (also think of the case when the table grows and the number of rows gets larger)? Provide corresponding SQL DDL statements.

4. Creating Materialized Views

Create a *materialized* view (called materialized query table in DB2) that contains all co-actorships, i.e., which actor played together with which other actor, for all actors of cinema movies in 2008 (use your tables created in exercise 2). The view should contain three columns. Here are some example rows:

('Ledger, Heath', 'Freeman, Morgan (I)', 'The Dark Knight (2008)'),
('Ledger, Heath', 'Bale, Christian', 'The Dark Knight (2008)'), …

Supplement the view with according indexes to speed up query processing.

5. Formulating Recursive SQL Queries

In this task, you will compute a list containing the "Six degrees of Heath Ledger in 2008" with the according Ledger-2008 number using your previously created materialized view. Look up "Six Degrees of Kevin Bacon" on Wikipedia.

First, formulate a (non-recursive) SQL query that lists all actors and actresses having a Ledger-2008 number of 3 or less, along with their respective Ledger-2008 number. Here are some example result rows: ('Ledger, Heath', 0); ('Freeman, Morgan', 1); ('Jolie, Angelina', 2); ('Jolie-Pitt, Shiloh', 3). Optimize your query for efficient processing (e.g. by ensuring that each co-actorship link is only used once during query processing)!

Technische Universität Braunschweig
Institut für Informationssysteme
http://www.ifis.cs.tu-bs.de

Wolf-Tilo Balke, Christoph Lofi, and Joachim Selke

Next, formulate a *recursive SQL query* for computing the same result as the previous query (try to formulate a query that is as short as possible). Also you might want to look up recursive SQL queries or just take a look on the detour in lecture 8. Since DB2 imposes some restrictions on what SQL operations are permitted in recursive queries, computing the Ledger-2008 number of every actor cannot be done efficiently (without applying complicated workarounds). Therefore, you only are required to compute the Ledger-2008 numbers up to (and including) 3.

As always, send all relevant SQL queries, results, and answers to your tutor via **e-mail.** In addition, also **print out** this information on paper and bring them to the next lab meeting.