January 12, 2016
Arnab Chakrabarti, Rihan Hai, Sandra Geisler

Implementation of Databases (WS 16/17)

Exercise 6

Due until January 24, 2017, 2pm.
Please submit your solution *in a single PDF file* before the deadline to the L^{2}P system!
Please submit solutions in groups of three students.

Exercise 6.1 (Serializability and Recoverability) (12 pts)

Consider the following two transactions running concurrently:

$$T1 = r_1(A)w_1(A)r_1(B)w_1(B)$$
$$T2 = r_2(B)r_2(A)w_2(A)w_2(B)$$

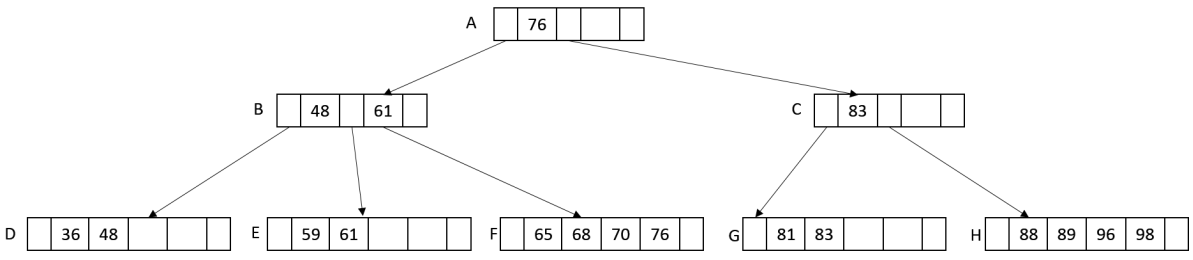
Assume that only exclusive lock and unlock actions are inserted by the scheduler, resulting in the following annotated transactions:

$$T1 = wl_1(A)r_1(A)w_1(A)wl_1(B)r_1(B)w_1(B)c_1wu_1(A)wu_1(B)$$
$$T2 = wl_2(B)r_2(B)wl_2(A)r_2(A)w_2(A)w_2(B)c_2wu_2(A)wu_2(B)$$

The two transactions are executed concurrently.

1. Is conflict serializability guaranteed? Why or why not? **(2 pts)**
2. Is cascading rollback possible? If not, explain why not. **(2 pts)**
3. Is deadlock possible? **(3 pts)**
4. Explain with example what are "Dirty Reads" and "Phantom Reads" ? **(5 pts)**

Exercise 6.2 (B+-tree Locking) (12 pts)



Given the above B+-tree of degrees k=1 and k*=2, describe the steps involved in executing each of the following operations according to the simple tree locking algorithm. Redraw the result B+-tree if it is updated. Be specific about the kind of lock obtained and answer each question independently of the others, always starting with the tree above. Note that we would like to unlock a node as early as possible to maximize concurrency. We also would like to maximize throughput; i.e., releasing a higher-level node has priority over releasing a lower level node. Use the notation *rl(node), wl(node), ru(node), wu(node), r(node), w(node)* to indicate shared locking, exclusive locking, shared unlocking, exclusive unlocking, reading and writing a node respectively. Use *delete(node)* to indicate the deletion of a node. Use *merge(child1,child2,resultNode)* (includes writing the resultNode) to indicate merging of two siblings. List the actions in the order they occur, and add short explanations if necessary.

1. Search key = 76 **(2 pts)**
2. Insert 82 **(4 pts)**
3. Delete 61 **(6 pts)**

Exercise 6.3 (Recovery) **(6 pts)**

Consider the recovery scenario described in the following. There are three transactions T1, T2, T3 updating pages A, B, C. At the time of the crash, the log contains the following entries:

LSN	LAST_LSN	TRAN_ID	TYPE	PAGE_ID
1	0	T3	update	B
2	0	T2	update	A
3	0	T1	update	C
4	2	T2	abort	
5	begin CKPT			
6	end CKPT			
7	1	T3	update	A
8	3	T1	update	C
9	8	T1	commit	

The transaction table and dirty page table for the checkpoint are:

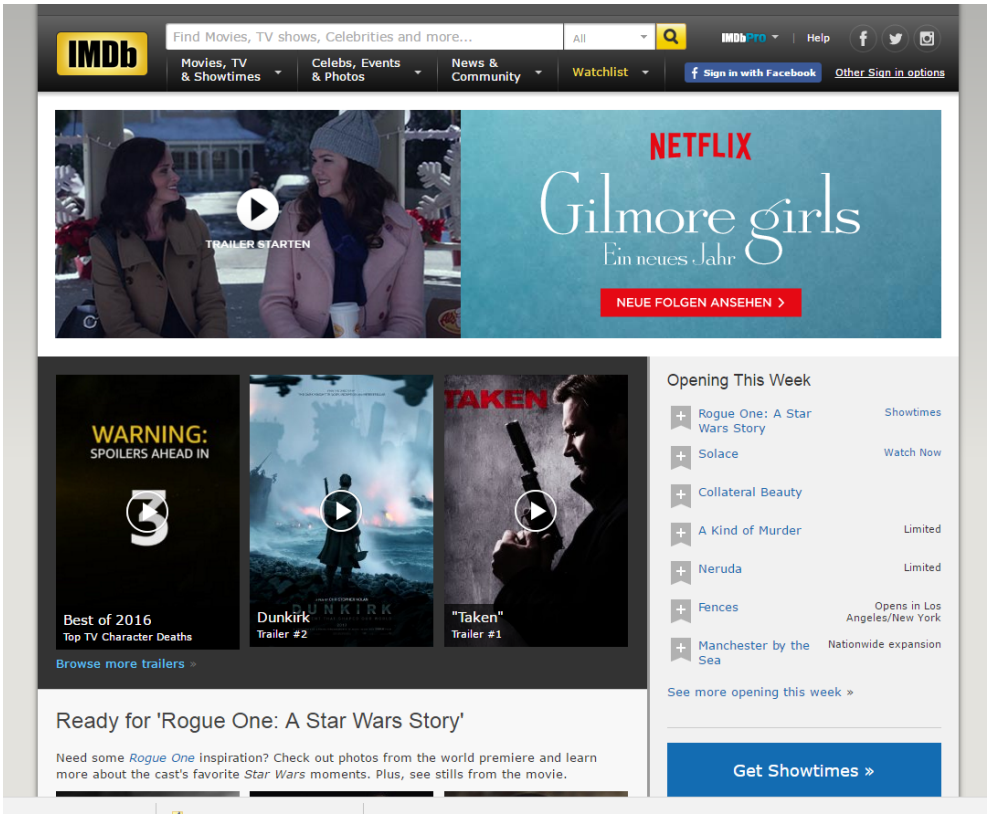
TRANSACTION_ID	LAST_LSN	STATUS
T1	3	active
T2	2	abort
T3	1	active

PAGE_ID	LSN
A	2
B	1
C	3

1. What is done during Analysis after the restart? Be precise about the points at which Analysis begins and ends and describe the contents of any tables constructed in this phase.
2. What is done during Redo? Be precise about the modifications to any tables due to prcoessing of log records.
3. What is done during Undo? Be precise about the modifications to the log and any tables due to processing of log records.

Exercise 6.4 (Bonus task & Challenge: Database Tuning Challenge) **(15 pts)**

In this task we use a real data set, Internet Movie Data Base (IMDB)[1]. It is an online movie related database which also covers information regarding actors, film companies, etc.



We provide a snapshot of IMDB (data from 05/2013) as csv files (around 3.6GB). You can download the files via `https://gigamove.rz.rwth-aachen.de/d/id/Tfjb6Dx6qSzMvF`. Use PostgreSQL to store the data and fulfill the following tasks. To begin with, below are some information for you to set up your database (the mentioned files are provided as attachment to this task in L2P).

- Create PostgreSQL database, name it as "imdbload":
  createdb imdbload
- We organized the real data from IMDB into 21 tables. In order to create all the tables, you can directly use the given file *schematext.sql*. It contains DDL statements *CREATE TABLE* with primary keys specified.
- Import the data from the given csv files into the corresponding tables in your database .
- Please use the below settings for PostgreSQL.You need to modify these setting in the file *postgresql.conf*, and restart the server after changing the settings. (The recommended setting is to boost the performance. If your computer doesn't support such setting, please write down the actual value of each parameter in your report).

[1] http://www.imdb.com/

Table 1: PostgreSQL Settings

Parameter Name	*Usage*	Value
work_mem	memory limit per operator	2 GB
shared_buffers	buffer pool size	4GB
effective_cache_size	size of the operating systems buffer cache	32GB
geqo_threshold	Force PostgreSQL to always use dynamic programming instead of fallingback to a heuristic for queries with more than 12 joins.	18

Consider the below query (the query file is also provided in L2P):

```
SELECT
  MIN(k.keyword) AS movie_keyword,
  MIN(n.name) AS actor_name,
  MIN(t.title) AS marvel_movie
FROM cast_info AS ci,
     keyword AS k,
     movie_keyword AS mk,
     name AS n,
     title AS t
WHERE k.keyword = 'marvel cinematic universe'
AND n.name LIKE '%Downey%Robert%'
AND t.production_year > 2010
AND k.id = mk.keyword_id
AND t.id = mk.movie_id
AND t.id = ci.movie_id
AND ci.movie_id = mk.movie_id
AND n.id = ci.person_id;
```

1. Draw the estimated execution plan for the above query. Which one is the most expensive step?
   (Tip: an operator tree of algebra expressions and segment/index scans as it is provided by using the EXPLAIN statement in PostgreSQL). **(2 pts)**

2. Fill in Table. 2 with the estimated cardinality, true cardinality, and q-error of each base table of the given query. Let estimated cardinality be c. Let true cardinality be c_t. The q-error e_q can be calculated by the below formula.

$$e_q = \begin{cases} \frac{c}{c_t}, \ if \ c > c_t \\ 1, \ if \ c = c_t \\ \frac{c_t}{c}, \ if \ c < c_t \end{cases}$$

   For instance, if true cardinality is 50, whether the estimate is 500 or 5, the q-error is 10. You can also understand it as $e_q = max(\frac{c}{c_t}, \frac{c_t}{c})$. If the table name in Table.2 has a '$\star$', please attach the screenshots how you get the results of the estimated cardinality and true cardinality of this table from PostgreSQL.
   (Tip: The true cardinality can be retrieved via executing queries with *SELECT COUNT(*)*) **(3 pts)**

Table 2: Cardinality comparison for base tables

	estimated cardinality	true cardinality	q-error
$\star$ *cast_info*			
name			
$\star$ *movie_keyword*			
keyword			
title			

3. Fill in Table. 3 with the estimated cardinality c, true cardinality c_t, and q-error e_q of each intermediate steps (mainly joins) of the execution plan. **(3 pts)**

Table 3: Cardinality comparison for intermediate steps

	c	c_t	e_q
$\sigma_{n.nameLIKE`\%Downey\%Robert\%'}(name)$			
$\sigma_{n.nameLIKE`\%Downey\%Robert\%'}(name) \bowtie cast\_info$			
$\sigma_{keyword='marvelcinematicuniverse'}(keyword)$			
$\sigma_{keyword='marvelcinematicuniverse'}(keyword) \bowtie movie_keyword$			
$(\sigma_{n.nameLIKE`\%Downey\%Robert\%'}(name) \bowtie cast\_info)$ $\bowtie$ $(\sigma_{keyword='marvelcinematicuniverse'}(keyword) \bowtie movie\_keyword)$			
$(\sigma_{n.nameLIKE`\%Downey\%Robert\%'}(name) \bowtie cast\_info)$ $\bowtie$ $(\sigma_{keyword='marvelcinematicuniverse'}(keyword) \bowtie movie\_keyword)$ $\bowtie$ $title$			

4. The database is heavily used with the query above and some additional queries. This generates a huge workload and the performance is going down. Your task as a database administrator is to increase the performance for the given workload. Here is additional information:

   - You may change the configuration of your server, such as the parameters in Table. 1(change the file postgresql.conf); if you do so, explain your changes briefly in the PDF document, and include the modified postgresql.conf in the submitted ZIP file.
   - The workload is generated by the stored procedure $genWorkload(N, Q)$(which you can find in the file *genWorkload.sql*). N is the number of statements which will be generated, and Q is the percentage of queries in the workload. In this task we assume about 100% queries. The genWorkload function returns an integer which represents the runtime of the workload in milliseconds. **The goal is to minimize the result for the query `SELECT genWorkload(1000,100).`** We recommend to start with the tiny database and smaller values for N, because without any optimizations, the workload will take hours or days.
   - You may not change or remove existing data nor make any assumptions about specific characteristics of the data. However, you may change the schema of the database: add columns, add key and foreign key constraints, change the type of columns, create indexes, create additional tables including materialized views, etc. Note that materialized views are not refreshed automatically in PostgreSQL. Whatever changes you do, include the SQL statements in a SQL script that can be executed to optimize the database from 'scratch', i.e., the original state of the database after running pgrestore. The SQL script should be part of your submission.
   - **UPDATE:** The queries can be changed, too. However, modified queries may use only the base relations and must not be rewritten to explicitly use views or other tables. It is not intended to optimize the database by denormalizing the schema to avoid join operations. If you change the schema, please note that you also have to update the

statements in the genWorkload function. If you change the genWorkload function, include the update in the SQL script of your submission. Make sure that the modified query really returns the same result as the original query.

- The team with the best solution will win a small prize and get the fame and honor of the whole class! **(7 pts)** .