Assignment 4
100 points

Professor Yanlei Diao

Question 1: OLTP versus OLAP Applications

Please indicate for each of the following applications, whether it is more suitable to use an OLTP database or an OLAP database. Briefly explain your answer if needed.

(a) Banking
(b) Ticketing
(c) Telecommunications: e.g., phone location updates and lookups
(d) E-commerce: searching and buying books
(e) Retailing: user profiling, promotion strategy analysis
(f) Financial services: claims analysis, risk analysis, credit card fraud detection
(g) Telecommunications: call pattern analysis, fraud detection
(h) Utilities: power usage analysis

Answer: the first four: OLTP; the last four: OLAP

Question 2: OLAP Operations

Consider the instance of Sales relation shown below.

pid	***timeid***	***locid***	***sales***
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
13	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26
12	2	2	45
12	3	2	20
13	1	2	20
13	2	2	40
13	3	2	5

(a) Show the result of pivoting the relation on *pid* and *timeid*.
(b) Write a collection of SQL queries to contain the same result as in Part (a).
(c) Show the result of pivoting the relation on *pid* and *locid*.

Answer:
(a)

	Timeid =1	Timeid =2	Timeid = 3	Total
Pid = 11	25+35=60	8+22=30	15+10=25	115
Pid = 12	30+26=56	20+45=65	50+20=70	191
Pid = 13	8+20=28	10+40=50	10+5=15	93
Total	144	145	110	399

(b)
```
    Select sum(sales)
    From SALES
    Group by pid, timeid ;

    Select sum(sales)
    From SALES
    Group by pid;

    Select sum(sales)
    From SALES
    Group by timeid;

    Select sum(sales)
    From SALES
```

(c).

	Locid =1	Locid =2	Total
Pid = 11	25+8+15=48	35+22+10=67	115
Pid = 12	30+20+50=100	26+45+20=91	191
Pid = 13	8+10+10=28	20+40+5=65	93
Total	176	223	399

Question 3: Implementation of the Cube Operator

Consider a relation SALES consisting of 19 dimension attributes, A_1, A_2, A_3, … and 1 numerical measure, Sales.
    SALES(A_1, A_2, A_3, …, A_{19}, sales)

Assume that all attributes are of the equal length.

We would like to run the following cube operator:
```
    Select A1, A2, A3, sales
    From SALES
    Group by Cube(A1, A2, A3) ;
```

Assume that the relation contains N pages and we have B buffer pages to run this query, where sqrt(N) < B < N. Please describe the most efficient algorithm to run this query and analyze its I/O cost.

Hint: please first think how to run: Group SALES By A_1, A_2, A_3. Then add other Group By operations covered by the cube.

Answer:
The baseline algorithm assumes that all tuples have distinct values in A_1, A_2, A_3.

(a) Group SALES By A_1, A_2, A_3. If sqrt(N) < B < N/10, both a single relation sorting or hashing algorithm works for this group by operation in two passes. The I/O cost is 4N, including the output S_{ABC}.

Size of S_{ABC} is estimated as follows: if 19 dimension attributes and 1 sales measure cover N pages, then 3 dimension attributes and 1 sales measure cover N*4/20 pages.

(b) Then, give the hierarchy, we next compute Group by AB, AC, BC from S1. Name the output S_{AB}, S_{AC}, S_{BC}.

I/O cost is: 4* N*4/20 * 3 = 12N/5.

The output of each covers N*3/20 pages.

(c) Then, we next compute Group by A, B, C from two of S_{AB}, S_{AC}, S_{BC}. Name the output S_A, S_B, S_C.

I/O cost is: 4* N*3/20 * 3 = 9N/5.

The output of each covers N*2/20 pages.

(4) Finally, we compute the global attributes.

I/O cost is N/10.

Total = 4N + 12N/5 + 9N/5 + N/10 = (8+3 /10)*N.

More efficient algorithms exist. So we can take any answer that is less that the above one.

Question 4: Query Rewriting

(1) Define a schema S, a database instance D, a set of views V, and a query Q such that Q(D) under the closed-world assumption given the extensions of the views in V is different from Q(D) under the open-world assumption given the extensions of the views in V.

(2) (a) Find a query Q, a query language L and a set of views V such that there is no maximally contained rewriting of Q using V with respect to L.
(b) For te same Q and V, find another language L' for which there exists a maximally contained rewriting of Q using V with respect to L'.

Question 5: Windows and Materialized Views in PostgreSQL

In this problem set, we continue to use the DBLP publication database containing information of over 3 million papers published in computer science conferences and journals (this data was derived from the DBLP system, maintained by Michael Ley at http://www.informatik.uni-trier.de/_ley/db/).

* **Schema**. This database consists of five tables: (1) an authors table, containing the names of authors, (2) the venue table, containing information about conferences or journals where papers are published, (3) the papers table, describing the papers themselves, (4) the paperauths table which indicates which authors wrote which papers, and (5) the papertypes . The schema is the following:

```
authors (id: INTEGER NOT NULL, name: VARCHAR(200))

venue (id: INTEGER NOT NULL, name: VARCHAR(200) NOT NULL, year: INTEGER NOT NULL, school:
VARCHAR (200), volume: VARCHAR(50), number: VARCHAR(50), type: INTEGER NOT NULL)

papers (id: INTEGER, name: VARCHAR NOT NULL, venue: INTEGER REFERENCES VENUE(id),
pages: VARCHAR(50), url: VARCHAR);

paperauths (paperid: INTEGER REFERENCES PAPERS(id), authid: INTEGER REFERENCES AUTHORS(id))

papertypes (id: INTEGER NOT NULL, type: VARCHAR(20) NOT NULL)
```

* **Indexes.** Your database has **at least** the following indexes:

- All primary key indexes,
- A B+ tree on the <name> attribute of the **authors** table, and
- A B+ tree on the <paperid, authid> attributes of the **paperauths** table.

If you have created other indexes, please drop them using the `drop index 'index-name'` command.

* **Useful commands:**

mysql: **`SHOW TABLES`**
postgresql: **`\d`**

mysql: **`SHOW INDEXES`**
postgresql: **`\di`**

mysql: **`SHOW COLUMNS`**
postgresql: **`\d` table**

mysql: **`DESCRIBE TABLE`**
postgresql: **`\d+` table**

* **Materialized Views.** PostgreSQL supports materialized views using the following command:
http://www.postgresql.org/docs/current/static/sql-creatematerializedview.html

* **Window Functions.** PostgreSQL supports the window functions as defined below:
http://www.postgresql.org/docs/current/static/tutorial-window.html

(a) Query writing using Views:

Define a single view that allows to rewrite the following type queries:

- find the name of the n authors having the maximum number of publications (n integer) ;
- find the n years when a maximum number of author have published ;
- find the average on the number of year on how many authors published exactly once in that year.

Try to minimize the size (number of rows and of columns) of the defined view.

(b) Query writing using Views:

Let us consider the following query:

> Find the venue where the author with id 17060 has published most.

Can you answer that query using only the above defined view? If yes, express it. Otherwise, prove this is not possible.

(c) Window Functions

(1) Compare the number of pages of each paper of each author with the average number of pages of all the papers of that author. Your resulting table should include 5 columns (names of these are in parentheses): ID of the author (id), name of the author (author), name of the paper (title), number of pages of the paper (pages), average number of pages of all papers of that author (avg). Finally, order your result set by author id (ascending order) and limit results to only the first 10 rows.

Answer:

```
SELECT id, author, title, avg
FROM (
      SELECT a.id, a.name as author, p.name as title, p.pages, avg(p.pages)
            OVER (PARTITION BY a.id) AS avg
      FROM authors a, papers p, paperauths x
      WHERE a.id = x.authid
      AND p.id = x.paperid ) AS page_rank
ORDER BY id ASC
LIMIT 10;
```

(2) Find each author's longest paper (paper with the most pages). In case of ties (if an author's longest paper has the same number of pages as other of his or her papers), order papers by alphabetical order.
Your resulting table should include 4 columns (names of these are in parentheses): ID of the author (id), name of the author (author), name of the paper (title), and number of pages of the paper (pages).
Finally, order your result set by author id (ascending order) and limit results to only the first 10 rows.

Answer:

```
SELECT id, author, title, pages FROM (
   SELECT a.id, a.name as author, p.name as title, p.pages, rank() OVER (
      PARTITION BY a.id
      ORDER BY pages DESC, p.name ASC
   ) AS page_rank
   FROM authors a, papers p, paperauths x
   WHERE a.id = x.authid
   AND p.id = x.paperid ) AS ranked
WHERE page_rank = 1
ORDER BY id
LIMIT 10;
```

(3) First, find the number of papers each author wrote or co-wrote each year. Now, using these results, store them in a temporary table R, and for each year an author has published something, find the minimum and maximum

number of papers the author has published in a span of 3 years (note that the years are not necessarily contiguous: for every year, look at the previous row and also at the next row in table R).

Your resulting table should include 5 columns (names of these are in parentheses): ID of the author (id), name of the author (author), year of publication (year), minimum number of papers published in a span of 3 publications (min), maximum number of papers published in a span of 3 publications (max).
Finally, order your result set by author id (ascending order) and year (ascending order) and limit results to only the first 10 rows.

Example:
Let's suppose author X published 4 papers in 2000, 5 papers in 2001, 3 papers in 2003, and 1 paper in 2005.
Your first result set (table R) would look something like this (depending on the names you give to your columns, of course).

```
 id | author | year | count
----+--------+------+-------
  1 |    X   | 2000 |   4
  1 |    X   | 2001 |   5
  1 |    X   | 2003 |   3
  1 |    X   | 2005 |   1
```

And this would be your final result set:

```
 id | author | year | min | max
----+--------+------+-----+-----
  1 |    X   | 2000 |  4  |  5
  1 |    X   | 2001 |  3  |  5
  1 |    X   | 2003 |  1  |  5
  1 |    X   | 2005 |  1  |  3
```

Answer:

```
SELECT id, author, year,
FIRST_VALUE(count) OVER (PARTITION BY id ROWS 1 PRECEDING) AS min,
FIRST_VALUE(count) OVER (PARTITION BY id ROWS 1 FOLLOWING) AS max
FROM (
   SELECT a.id, a.name as author, v.year, count(p.id)
   FROM authors a, venue v, papers p, paperauths x
   WHERE a.id = x.authid AND p.id = x.paperid AND p.venue = v. id
   GROUP BY a.id, v.year
   ORDER BY a.id, v.year
) AS year_count
ORDER BY id, year
LIMIT 10;

Rows between 1 pre and 1 fow
```