# Assignment 3
120 points

Professor Yanlei Diao

## Question 1 [12 points] Sorting

Suppose that you have a file with 1,000,000 pages and you have 21 buffer pages. Answer the following questions assuming that the external sorting algorithm is used.

**(1)** How many runs will you produce in the first pass?

**(2)** How many passes will it take to sort the file completely?

**(3)** What is the total I/O cost of sorting the file?

**(4)** How many buffer pages do you need to sort the file completely in just two passes?

## Question 2 [20 points] Evaluation of selections

Consider a relation with this schema:

Employees(<u>*eid*: integer</u>, *ename*: string, *sal*: integer, *title*: string, *age*: integer)

Suppose that the following indexes, all using Alternative (2) for data entries, exist:

a) a hash index on *eid*,

b) a B+ tree index on *sal*,

c) a hash index on *age*,

d) a clustered B+ tree index on <*age*, *sal* >.

Each Employees record is 100 bytes long, and you can assume that each index data entry is 20 bytes long. The Employees relation contains 10,000 pages and each page holds 20 data records.

Consider each of the following selection conditions. Assume that the *reduction factor* (RF) for each term that matches an index is 0.1, except those terms on the primary key. Compute the cost of the <u>*most selective access path*</u>, including the file scan and various index scans, for retrieving all Employees tuples that satisfy the condition:

**(1)** $sal > 100$

**(2)** $age = 25$

**(3)** $eid = 1000$

**(4)** $sal > 200 \wedge age > 20$

**Question 3 [10 points] Output of join algorithms**
Suppose we have two unary (one attribute only) relations, R and S:

| R | S |
|---|---|
| 7 | 8 |
| 2 | 4 |
| 9 | 2 |
| 8 | 1 |
| 3 | 3 |
| 9 | 2 |
| 1 | 7 |
| 3 | 3 |
| 6 | |

Show the result of joining R and S using each of the following algorithms. List the results in the order that they would be output by the join algorithm. Note that the result relation contains *only one attribute*, which is the common attribute between R and S.

**(1)** Sort merge join.

**(2)** Hash join. Assume there are two hash buckets, numbered 0 and 1, and that the hash function sends even values to bucket 0 and odd values to bucket 1. In Phase II, use R as the "build" relation and S as the "probe" relation. Assume that bucket 0 is read first and that the contents of a bucket are read in the same order as they were written.

**Question 4 [32 points] Improved Hash Algorithms**

**Hybrid Hash Join.** Consider a join of relations R and S using hash join, where *M* = *num_pages_in_R* = 400 pages and *N* = *num_pages_in_S* = 500 pages. Assume that we have B = 40 pages of memory available as our buffer. We observe that there is more than enough memory to run the two-phase hash join algorithm, so we try to keep one of the hash buckets in memory to avoid writing it during the partitioning phase and then re-reading it during the probing phase. Assuming that all buckets have the same size.

**(1) [6 points]** Do we have enough memory to perform the hybrid hash join still in two phases (or two passes)?

**(2) [6 points]** How many buckets should be used to perform the join in two phases?

**(3) [4 points]** What would be the cost in number of I/O operations in the improved join algorithm?

**Advanced hash algorithms for aggregation.** Consider the following query:

```
Select count(*)
From R
Group By R.a
```

Again, R has 400 pages, and the memory size is 40 pages.

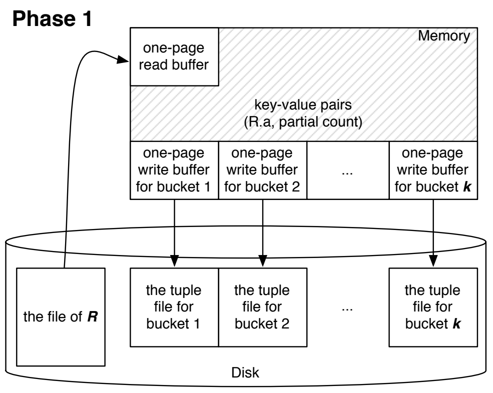
**(4) [6 points]** Now apply the idea of hybrid hash to group by-aggregation by still keeping the first hash bucket in memory. Compute the I/O cost of answering the above query using the new algorithm.

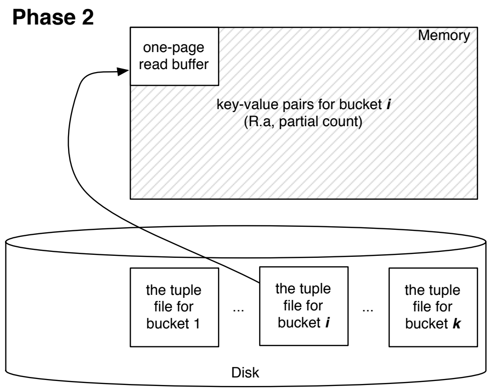
**(5) [10 points]** There exists a more advanced hash algorithm for group by-aggregation. Given the above query, it works as follows:

- As we read tuples in R, we build an in-memory data structure H that maps each distinct value of R.a (the key of H) to the partial count (the value of H).
- As more tuples are read, the partial counts of existing keys in H can be updated and new key-value pairs can be added to H.
- Let us set the maximum size of H to be B_H, which is less than the total memory size B. Once the map H reaches its maximum size, there are two possible cases of dealing with each tuple: (a) If R.a already exists in H, we simply update the partial count. (b) Otherwise, we hash the tuple to one of k buckets, place the tuple in the write buffer of that bucket (just 1 page), and flush the write buffer when it becomes full.
- After we have read all tuples from R, we already have the counts for a subset of values of R.a. To complete the computation for other values of R.a, we read each bucket back, one at a time, and perform any in-memory processing needed for the group by-aggregation in this bucket.

Assume that all distinct values of R.a and their counts can be held in U pages.

- What is the I/O cost of the query if the memory size B is larger than U?
- What is the minimum memory size that allows the query to be completed by reading R at most twice?


Phase 1
Memory
one-page read buffer
key-value pairs (R.a, partial count)
one-page write buffer for bucket 1
one-page write buffer for bucket 2
...
one-page write buffer for bucket k
the file of R
the tuple file for bucket 1
the tuple file for bucket 2
...
the tuple file for bucket k
Disk


Phase 2
Memory
one-page read buffer
key-value pairs for bucket i (R.a, partial count)
the tuple file for bucket 1
...
the tuple file for bucket i
...
the tuple file for bucket k
Disk

**Question 5 [16 points] Complexity of Query Optimization**

Analyze the complexity of System R-style query optimization:

**(1) [10 points]** What is the complexity of dynamic programming for finding an optimal plan for an *n*-way join? Here, consider the total number of plans that the query optimizer enumerates and denote the complexity using the big-O notation.
**(2) [6 points]** What is the maximum number of plans stored in an intermediate pass?

Hint: consider the star join graph that we studied in class and analyze the complexity related to this graph.

**Question 6 [30 points]: Query Optimization in PostgreSQL**

In this problem set, we consider a 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 is allowed and only allowed to have the following indexes:
- All primary key indexes,
- A B+ tree on the <name> attribute of the **authors** table, and
- A B+ tree on the <authid,paperid> attributes of the **paperauths** table.

Recall that the command to create an index, e.g., for the last index, is:

```
create index authors_name on authors (name);
create index paperauths_pkey on paperauths (authid,paperid);
```

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

* **System catalog and statistics.** As we learned in class, the query planner needs to estimate the number of rows retrieved by a query in order to make good choices of query plans.

One important type of the statistics is the total number of entries in each table and index, as well as the number of disk blocks occupied by each table and index. This information is kept in the table **pg_class**, in the columns reltuples and relpages. We can look at it with queries similar to this one:

```
SELECT relname, relkind, reltuples, relpages
FROM pg_class
WHERE relname LIKE 'author%';
```

When a table is large, the number of tuples in the pg_class may differ slightly from the actual number of tuples (which you can compute using "Select count(*) From table_name"). For large tables,  ANALYZE takes a random sample of the table contents, rather than examining every row. This allows even very large tables to be analyzed in a small amount of time.

The view **pg_stats** provides additional information about each attribute. Of particular importance are:

- *avg_width*: the average width of an attribute, which is especially useful for variable length attributes.
- *n_distinct*: the number of distinct values in an attribute. If greater than zero, it is the estimated number of distinct values in the column. If it is -1, then this attribute is declared to be unique so it has a distinct value in each row.
- *histogram_bounds*: and the histogram bounds that divide the column's values into groups of approximately equal population.

```
select tablename, attname, avg_width, n_distinct, histogram_bounds
from pg_stats
where tablename like 'author%';
```

Other useful commands:

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

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

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

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

* **Query plans.** We use the EXPLAIN command to see the query plan used by PostgreSQL:

```
explain select * from authors where name = 'David J. DeWitt';

                              QUERY PLAN
-----------------------------------------------------------------------------
Index Scan using authors_name on authors  (cost=0.43..8.45 rows=1 width=19)
Index Cond: ((name)::text = 'David J. DeWitt'::text)
```

This query plan uses an index scan based on authors.name. Its estimated costs are in units of disk page fetches, between 0.43  (time expended before output scan can start) and 8.45 (total cost). Furthermore, the output is expected to contain 1 answer of width 19 in bytes.

**(1)** Consider the following query:

```
select * from authors
where name < 'David J. DeWitt';
```

a) What is the query plan that PostgreSQL uses for this query? What is the textbook description of this type of query plan? (Or name the slide in the lecture notes that describe this type of query plan.)
b) What is the estimated number of rows? What is the actual number of rows returned by the query?
c) If the estimated number of rows differs from the actual number of rows returned, briefly explain how this happens. (Hint: Use pg_class and pg_stats to look for the statistics about the authors relation and the name attribute.)

**(2)** Consider the following query:

```
select * from authors
where name < 'A. A';
```

a) What is the query plan that PostgreSQL uses for this query? What is the textbook description of this type of query plan? (Or name the slide in the lecture notes that describe this type of query plan.)
b) Briefly explain why the plan chosen from PostgreSQL for this query differs from that for the previous query.

**(3)** Consider the next query:

```
SELECT * FROM papers p, authors a, paperauths pa
WHERE pa.paperid = p.id
AND pa.authid = a.id
AND a.name = 'David J. DeWitt';
```

a) What is the query plan that PostgreSQL uses for this query?
b) Briefly explain why PostgreSQL chooses this plan.
c) What is the estimated number of returned rows? What is the actual number of rows returned by the query? Briefly explain why the estimated number of rows differs from the actual number of rows returned.

**(4)** Now change the above query to:

```
SELECT * FROM papers p, authors a, paperauths pa
WHERE pa.paperid = p.id
AND pa.authid = a.id
AND a.name < 'David J. DeWitt';
```

a) What is the query plan that PostgreSQL uses for this query?
b) Briefly explain why PostgreSQL chooses this plan.

**(5)** Finally, let us change the query to:

```
SELECT * FROM papers p, authors a, paperauths pa
WHERE pa.paperid = p.id
AND pa.authid = a.id
AND a.name > 'David J. DeWitt'
```

a) What is the query plan that PostgreSQL uses for this query?
b) Briefly explain why PostgreSQL chooses this plan.