#### Databasesystems 2

Forum: https://forum-db.informatik.uni-tuebingen.de/c/ss18-db2

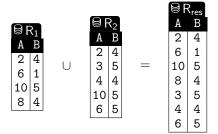
# Assignment 11 (10.07.2018)

Submission: Tuesday, 17.07.2018, 10:00 AM

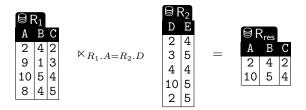
## 1. [15 Points] Implementation of Relational Operators

The following examples outline the semantics of the *Relational Union* ( $\cup$ ) and *Left Semi Join* ( $\ltimes_{a=b}$ ) which are operators used in the relational algebra:

• Union ( $\cup$ ): The result contains all rows of  $R_1$  and  $R_2$  without duplicates:



• Left Semi Join  $(\ltimes_{a=b})$ : The result contains every row of  $R_1$  that finds at least one join partner in  $R_2$ :



Formulate hash- and sorting-based algorithms in pseudocode which implement the above mentioned operators. Use pseudocode syntax based on the sample code found on slides 15 and 23 in slide set 12. You are also allowed to use operations like sort() or [not] in <hashtable>. If you use unusual operators in your pseudocode, please include a short note on their semantics.

Note: Disregard blocked I/O in your pseudocode implementation.

### 2. [15 Points] Join Operators in PostgreSQL

Nested Loop Join: Disable Hash- and Merge Join to answer the following questions!

```
set enable_hashjoin=off;
set enable_mergejoin=off;
```

(a) Create tables one and many as provided in one-many.sql. As long as there are no keys or indexes created on both tables, a query to join them using *Nested Loops Join* will not terminate in reasonable time. Use EXPLAIN to show the plan of the following query Q:

```
SELECT *
FROM one AS o, many AS m
WHERE o.a = m.a
```

- Based on the estimated rows, how often would the Join Filter o.a = m.a be evaluated?
- (b) A PRIMARY KEY index on one (a) supports the Nested Loops Join on query Q. How?

```
ALTER TABLE one ADD CONSTRAINT one_a PRIMARY KEY (a);
ALTER TABLE many ADD FOREIGN KEY (a) REFERENCES one(a);
ANALYZE one; ANALYZE many;
```

- Show the plan using EXPLAIN ANALYZE and explain briefly.
- (c) An additional PRIMARY KEY index on many(a,c) again improves the query performance.

```
ALTER TABLE many ADD CONSTRAINT many_a_c PRIMARY KEY (a,c); ANALYZE one; ANALYZE many;
```

- Why is only one of both indexes used, while the other table is accessed using a Seq Scan?
- Why is table many with index many\_a\_c (and not table one with one\_a) preferred as inner join table here?
- (d) How does the following modification of query Q benefit from both indexes, instead?

```
SELECT *
FROM one AS o, many AS m
WHERE o.a = m.a
ORDER BY m.a
```

#### Hash Join: Re-enable Hash- and Merge Join to answer the following questions!

```
set enable_hashjoin=on;
set enable_mergejoin=on;
```

- (e) If available, a  $Hash\ Join$  is used to answer the equi-join query Q. Show the plan using EXPLAIN (VERBOSE, ANALYZE, BUFFERS).
  - Why is table one (and not table many) chosen as the build table here?
  - Why are the indexes one\_a and many\_a\_c not used here?
- (f) Hash Join builds a temporary hash table and thus suffers when work\_mem is reduced.

  set work\_mem='64kB' (instead of default '4MB') and re-execute the query. The Hash Join performance decreases significantly. Use the output of EXPLAIN (VERBOSE, ANALYZE, BUFFERS) to compare it with 2e and explain why.
- (g) Since a part of the hash table is stored in-memory even for low work\_mem, the actual performance of 2f can highly depend on the data distribution of the probed table. Table many\_skewed in one-many.sql provides a variant of table many with a heavily skewed distribution on column a. Examine columns n\_distinct, most\_common\_vals and most\_common\_freqs in table pg\_stats¹ to show statistics about the distribution of values for both, attribute a in many and a in many\_skewed.
  - Give a short comparison.

Execute the following query on work\_mem='64kB' and compare its plan to 2f.

- The I/O on temporary tables is reduced. To which number and why?

```
SELECT *
FROM one AS o, many AS m
WHERE o.a = m.a
```

#### Merge Join: Disable Hash Join to answer the following question!

```
set enable_hashjoin=off;
```

- (h) When we enforce Merge Join in 2f, we can observe that Merge Join performs quite better than the Hash Join.
  - How does the *Merge Join* make use of indexes one\_a and many\_a\_c?
  - Why does it outperform the *Hash Join* on work\_mem='64kB'?

https://www.postgresql.org/docs/current/static/view-pg-stats.html