

Database Lab

Queries

Fall Term 2023

Dr. Andreas Geppert

geppert@acm.org

Topics

- conceptual design
- logical design
- consistency constraints
- data manipulation
- **queries**
- transactions
- views
- stored procedures and user-defined functions
- triggers
- database applications with Java (JDBC)

(Query) Performance

- Query evaluation:
 - Bring data onto server (main memory)
 - Evaluate query
- Query optimization
 - Done by the optimizer (planner in Postgres)
 - Searches „optimal“ query execution plan
 - Input:
 - Query
 - Access paths like indexes
 - Statistics
 - Cost parameters
 - (Implementation alternatives)

(Query) Performance

- Example: select resnumber from reservation order by resnumber
 1.
 - read all rows from disk (full table scan)
 - Sort rows
 2.
 - Use (B-tree) index on table reservation over column resnumber (index scan)
- Multiple alternatives exist for operators
 - e.g. join: nested loop, sort/merge, hash, ...

(Query) Performance

- Taking the time of query execution

```
cashdb=> \timing
```

Timing is on.

```
cashdb=> select * from reservation order by resnumber;
```

Time: 6.286 ms

- Investigating execution plans in Postgres

- Explain operator:

```
cashdb=> explain select * from reservation order by reservation;
```

QUERY PLAN

Sort (cost=212.14..217.64 rows=2200 width=130)

Sort Key: reservation.*

-> Seq Scan on reservation (cost=0.00..90.00 rows=2200 width=130)

(3 rows)

(Query) Performance

- Explain analyze:

cashdb=> explain analyze select * from reservation order by reservation;

QUERY PLAN

Sort (cost=212.14..217.64 rows=2200 width=130) (actual time=34.370..34.452 rows=2200 loops=1)

Sort Key: reservation.*

Sort Method: quicksort Memory: 681kB

-> Seq Scan on reservation (cost=0.00..90.00 rows=2200 width=130) (actual time=12.275..22.053 rows=2200 loops=1)

Planning Time: 0.104 ms

Execution Time: 34.664 ms

(6 rows)

Plan Visualization

- Sample query: all adjacent pairs of reservations of the same vehicle

```
• select v.licenseplate,  
       (r1.interval).begints,  
       (r2.interval).begints  
  from reservation r1 join  
       reservation r2 on  
         (r1.interval).endts = (r2.interval).begints  
         and r1.vehicle = r2.vehicle join  
       limousine v on r1.vehicle = v.id  
 where r1.id < r2.id;
```

Plan Visualization

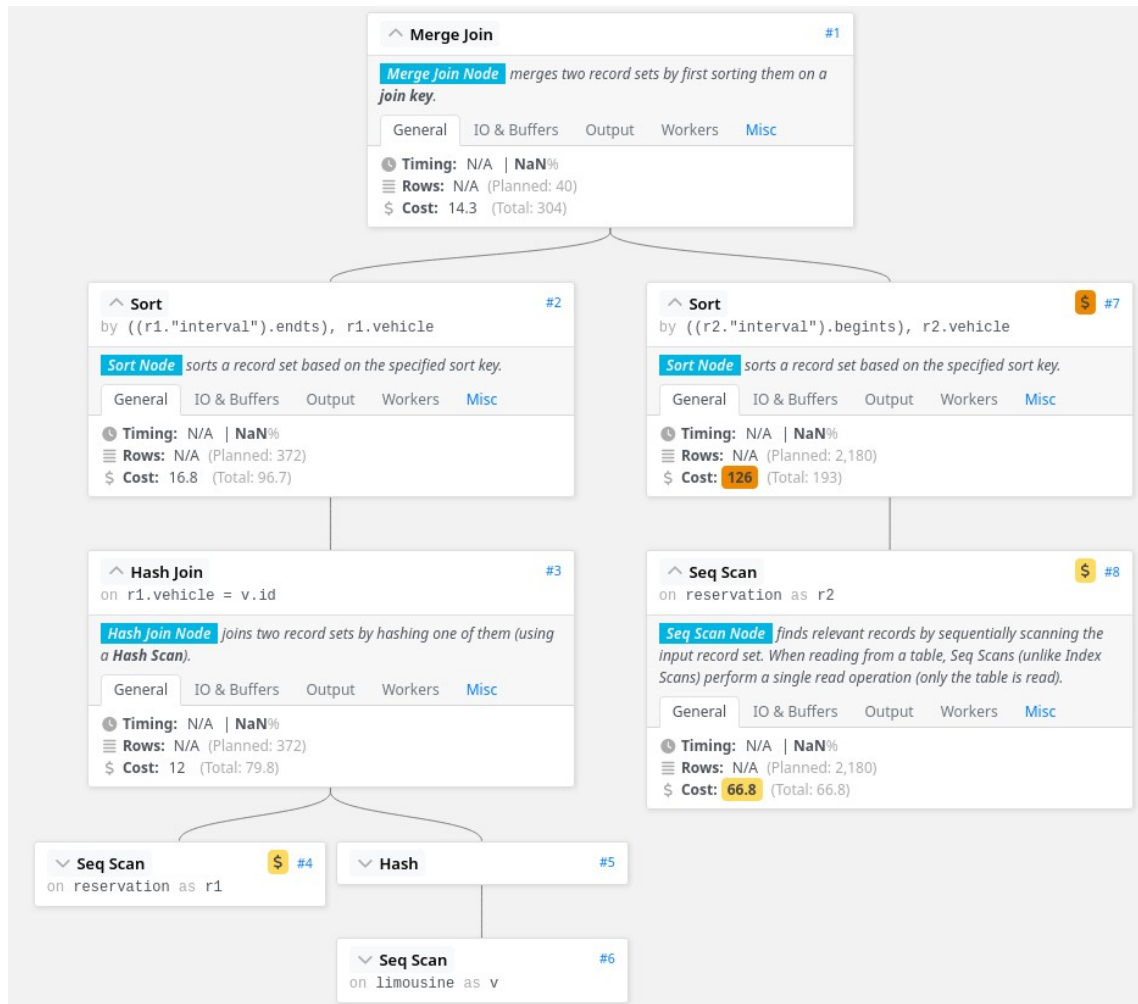
Execution Plan

- Merge Join (cost=283.42..304.07 rows=40 width=54)
 - Merge Cond: (((r1."interval").endts) = ((r2."interval").begints)) AND (r1.vehicle = r2.vehicle)
 - Join Filter: (r1.id < r2.id)
 - > Sort (cost=95.74..96.67 rows=372 width=87)
 - Sort Key: ((r1."interval").endts), r1.vehicle
 - > Hash Join (cost=1.16..79.85 rows=372 width=87)
 - Hash Cond: (r1.vehicle = v.id)
 - > Seq Scan on reservation r1 (cost=0.00..66.80 rows=2180 width=45)
 - > Hash (cost=1.07..1.07 rows=7 width=42)
 - > Seq Scan on limousine v (cost=0.00..1.07 rows=7 width=42)
 - > Sort (cost=187.68..193.13 rows=2180 width=45)
 - Sort Key: ((r2."interval").begints), r2.vehicle
 - > Seq Scan on reservation r2 (cost=0.00..66.80 rows=2180 width=45)

Plan Visualization

Visualization Using Dalibo

- Paste execution plan at <https://explain.dalibo.com/>



Plan Visualization

Visualization Using PGAdmin4

The screenshot shows the PGAdmin4 interface with a query plan visualization. The query is:

```
1 select v.licenseplate,  
2     (r1.interval).begints,  
3     (r2.interval).begints  
4 from cash_reservation r1 join
```

The plan visualization shows the following flow:

- Two **reservation** tables are processed.
- Each **reservation** table is followed by a **Sort** node.
- The **Sort** nodes feed into a **Merge Inner Join** node.
- A **limousine** table is processed, followed by a **Hash** node.
- The **Hash** node and the **Merge Inner Join** node feed into a final **Hash Inner Join** node.

The right-hand pane shows the details for the **Merge Inner Join** node:

Merge Inner Join	
Node Type	Merge Join
Parent Relationship	Outer
Parallel Aware	
Async Capable	
Join Type	Inner
Inner Unique	
Merge Cond	$((r1.vehicle = r2.vehicle) \wedge (((r1."interval").endts < (r2."interval").begints)))$
Join Filter	$(r1.id < r2.id)$
loops	