PostgresPro

POSTGRESQL DEVELOPMENT CONFERENCE 2024

PGConf.dev

Adaptive Query Optimization (AQO)

Speaker: Alena Rybakina

Presentation was made by: Alena Rybakina, Andrei Lepikhov

Self Introduction

PGConf.dev

- Core developer in Postgres Professional since **2021**
- B.S. in Computer Science (Informatics and Computer Science), State Dubna University, **2021**
- Certificate of advanced training “Big data analytics”, **2021**
- Contributing to the PostgreSQL project since **2023**
    - OR to ANY transformation
    - Self–Join–Elimination
- Participated in extension development:
    - AQO
    - sr_plan
    - replaning

What is AQO?

Outline Of The Talk

1. How does Adaptive Query Optimization(AQO) work?

What is AQO?

Outline Of The Talk

1. How does Adaptive Query Optimization(AQO) work?

2. Problems & Features

What is AQO?

Outline Of The Talk

- 1. How does Adaptive Query Optimization(AQO) work?
- 2. Problems & Features
- 3. Examples
-

What is AQO?

Outline Of The Talk

- 1. How does Adaptive Query Optimization(AQO) work?
- 2. Problems & Features
- 3. Examples
- 4. Testing results

What is AQO?

How does AQO work?

Adaptive Query Optimization

PGConf.dev

- It improves cardinality estimation
- It can effect the planner to find more optimal plan
- It saves the real cardinality information to use it in the future

Adaptive Query Optimization

- It improves cardinality estimation
- It can effect the planner to find more optimal plan
- It saves the real cardinality information to use it in the future

How does it work?

Optimization Issues

PGConf.dev

- It needs always to use actual statistics
-
-

Optimization Issues

PGConf.dev

- It needs always to use actual statistics
- It poorly works with a large number of joins in the query
-

Optimization Issues

PGConf .dev

- It needs always to use actual statistics
- It poorly works with a large number of joins in the query
- It has an assumption in uniform distribution between columns

What Does This Lead To

PGConf.dev

Wrong cardinality estimation

What Does This Lead To

Wrong cardinality estimation

Wrong cost estimation

What Does This Lead To

PGConf.dev

Wrong cardinality estimation

↓

Wrong cost estimation

↓

Choose nonoptimal plans

1. How does AQO work?

2. Problems & Features

What Does This Lead To

PGConf.dev

Wrong cardinality estimation

Wrong cost estimation

Choose nonoptimal plans

We are only engaged in improving the assessment of cardinality and don't consider assessment of cost

Problem With Cardinality Estimation

PGConf.dev

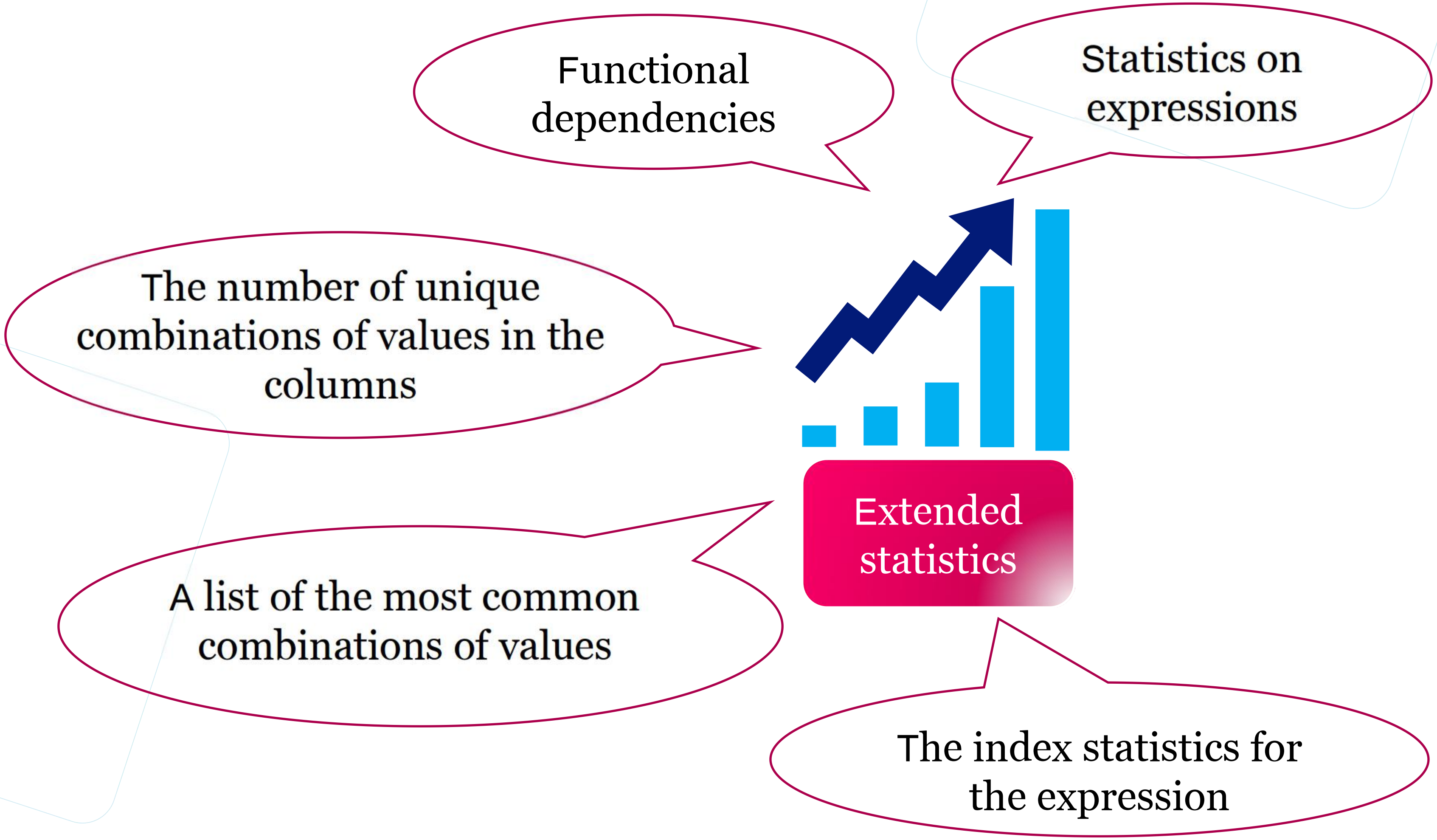
How good are query optimizers, really?
V.Levis, A.Gubichev, A.Mirchev, P.Boncz,
A.Kemper and T.Neumann,
Proc. VLDB, Nov.2015

Adaptive Cardilnality Estimation
O.Ivanov, S.Bartunov,
Arxiv, Nov.2017

1. How does AQO work?

2. Problems & Features

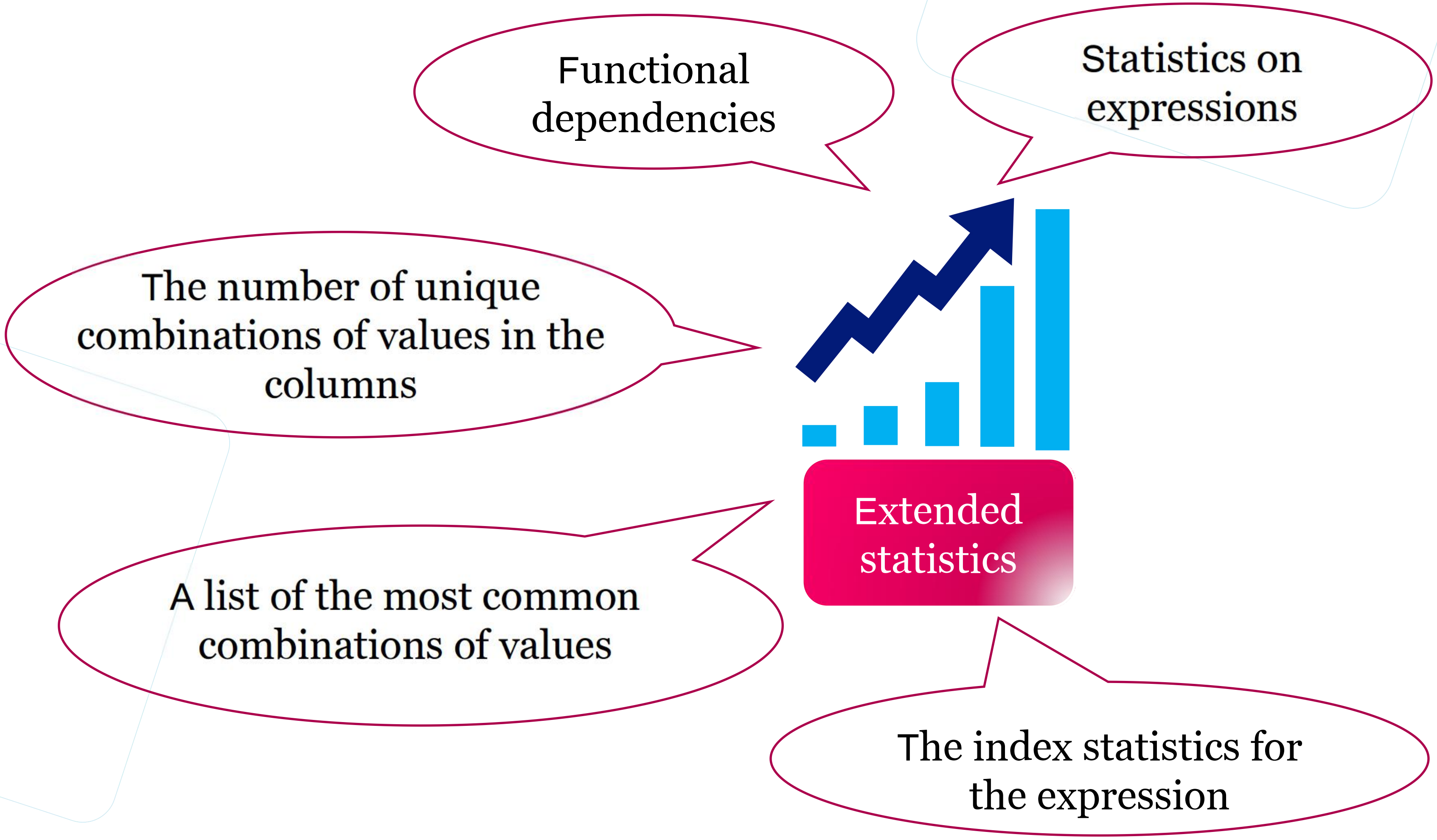
Solutions From PostgreSQL



Pros:

- Solve the problem
- Have theoretical guarantees

Solutions From PostgreSQL



Pros:
- Solve the problem
- Have theoretical guarantees

Contras:
- Require memory
- Require time for building or updating
- Not clear which of all possible column subsets are needed

The Memory For Planner

PGConf.dev

- It can store the actual cardinality of nodes and passes it to the optimizer next time
- It should store the selectivity of nodes to determine whether cardinality is appropriate for current selectivities
- It should learn from mistakes and correct data

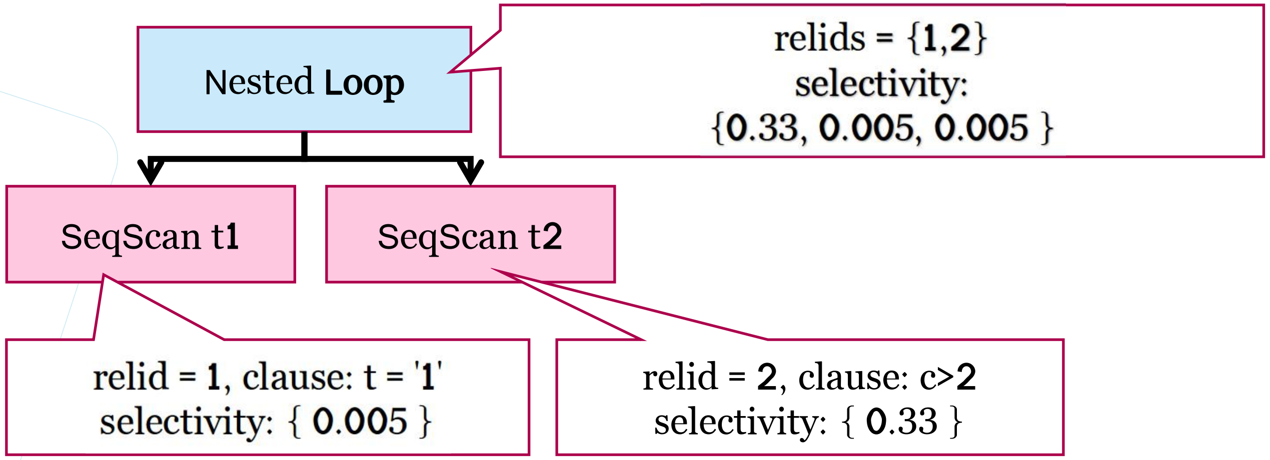
The Memory For Planner

- It can store the actual cardinality of nodes and passes it to the optimizer next time
- It should store the selectivity of nodes to determine whether cardinality is appropriate for current selectivities
- It should learn from mistakes and correct data

It is the main idea that AQO is based on!

Discuss The Model Description Of AQO

For example:
explain analyze SELECT *
FROM t1, t2
WHERE t1.x = t2.y AND
t1.t = '1' AND
t2.c > 2;



- Every node is described through its selectivity of all its conditions

1. How does AQO work? → 2. Problems & Features

Discuss The Model Description Of AQO

For example:

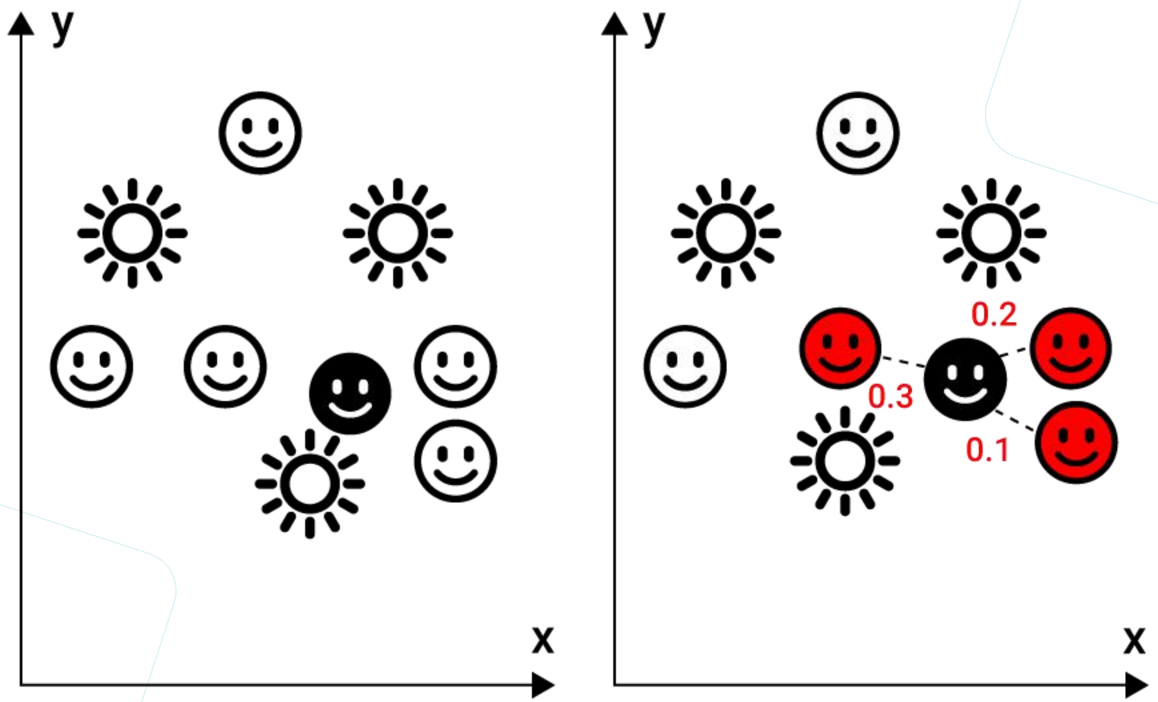
```
explain analyze SELECT *
                 FROM t1, t2
                 WHERE t1.x = t2.y AND
                       t1.t = '1' AND
                       t2.c > 2;
```

=

```
explain analyze SELECT *
                 FROM t1, t2
                 WHERE t1.x = t2.y AND
                       t1.t = '3' AND
                       t2.c > 5;
```

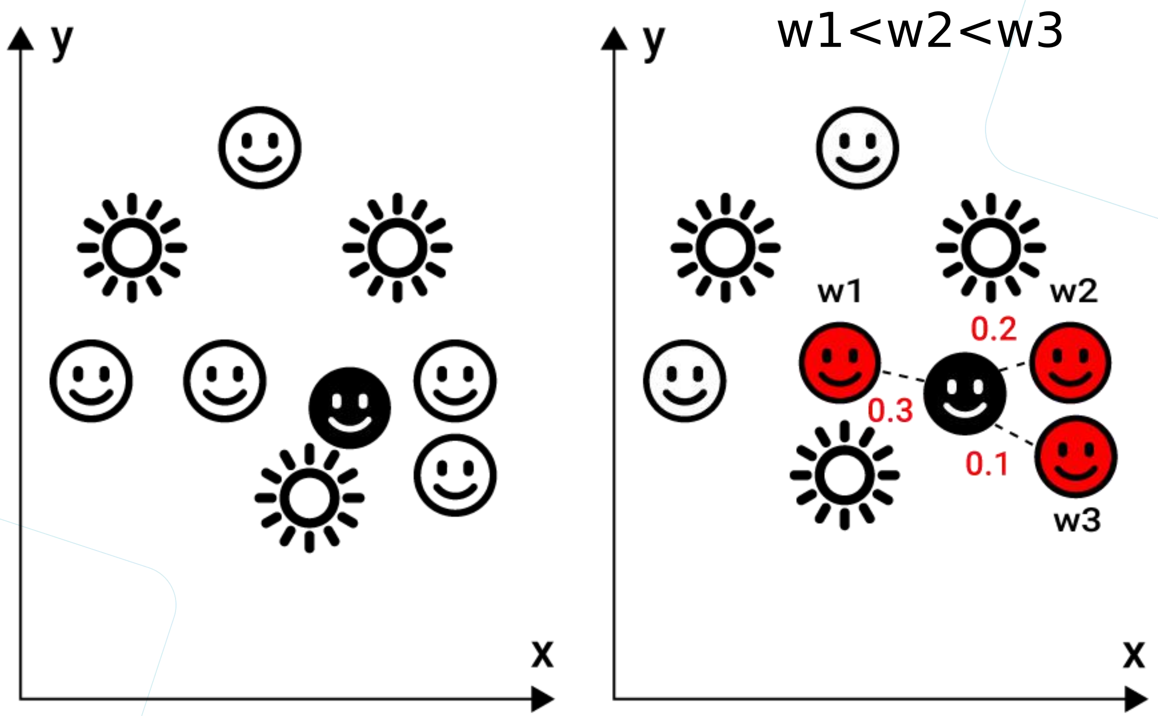
- Every node is described through its selectivity of all its conditions
- All conditions in the node that differ only in constants are equivalent

Discuss The Model Description Of AQO



- Every node is described through its selectivity of all its conditions
- All conditions in the node that differ only in constants are equivalent
- The closest nodes of the query based on R2–distance are neighbours

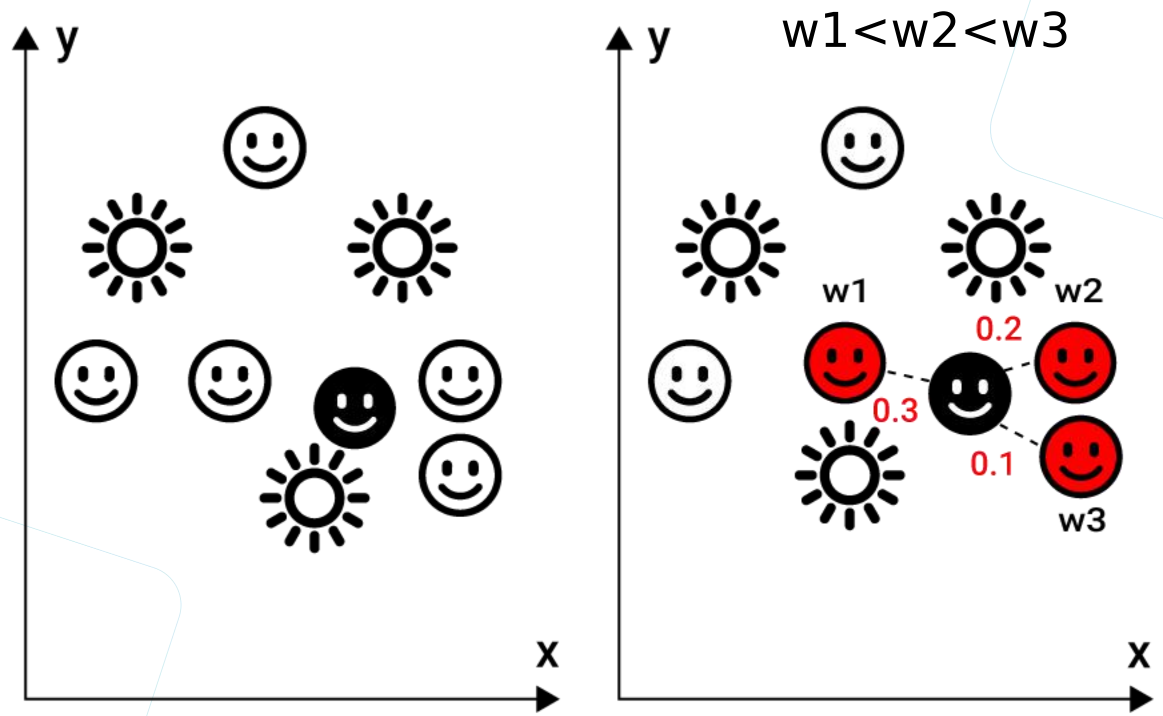
Discuss The Model Description Of AQO



- Every node is described through its selectivity of all its conditions
- All conditions in the node that differ only in constants are equivalent
- The closest nodes of the query based on R2–distance are neighbours
- We need to predict the number of rows of a new node based on the known data of its neighbours

1. How does AQO work? → 2. Problems & Features

Discuss The Model Description Of AQO



$$\text{Cardinality} = \sum_{i=1}^{30} \frac{\text{cardinality}_i * \text{weight}_i}{\sum \text{weight}}$$

- Every node is described through its selectivity of all its conditions
- All conditions in the node that differ only in constants are equivalent
- The closest nodes of the query based on R2–distance are neighbours
- We need to predict the number of rows of a new node based on the known data of its neighbours
- Define the number of rows as a weighted average of the cardinalities of neighbours

Basic Priciples Of Implementation Of AQO

Using K Nearest Neighbours method

Learning workflow is iterative:

- After the execution stage some of these objects are appended to the train set (set of queries) and the model can learn from them.
- On the planning stage the model tries to predict cardinality for a node

Basic Priciples Of Implementation Of AQO

Using K Nearest Neighbours method

Learning workflow is iterative:

- After the execution stage some of these objects are appended to the train set (set of queries) and the model can learn from them.
- On the planning stage the model tries to predict cardinality for a node

Basic Priciples Of Implementation Of AQO

Using K Nearest Neighbours method

Learning workflow is iterative:

- After the execution stage some of these objects are appended to the train set (set of queries) and the model can learn from them.
- On the planning stage the model tries to predict cardinality for a node

Basic Priciples Of Implementation Of AQO

Using K Nearest Neighbours method

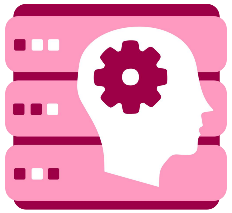
Learning workflow is iterative:

- After the execution stage some of these objects are appended to the train set (set of queries) and the model can learn from them.
- On the planning stage the model tries to predict cardinality for a node

Math for learning:

- Loss function – evaluate the discrepancy between predicted and actual rows
- Stochastic gradient optimizes data in the train set

AQO Components



AQO Data

It stores selectivities and number of rows of nodes

AQO Components

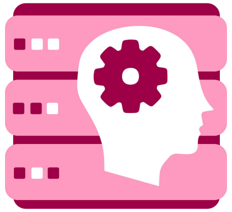
AQO Data

It stores selectivities and number of rows of nodes

AQO queries

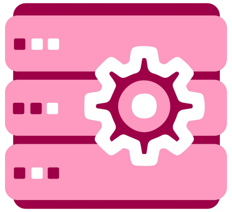
Settings for all known queries: learning, using and autotunning AQO

AQO Components



AQO Data

It stores selectivities and number of rows of nodes



AQO queries

Settings for all known queries: learning, using and autotunning AQO

AQO query text

It stores all known queries and it's hashes and query texts

1. How does AQO work?

2. Problems & Features

AQO Components

AQO Data

It stores selectivities and number of rows of nodes

AQO queries

Settings for all known queries: learning, using and autotunning AQO

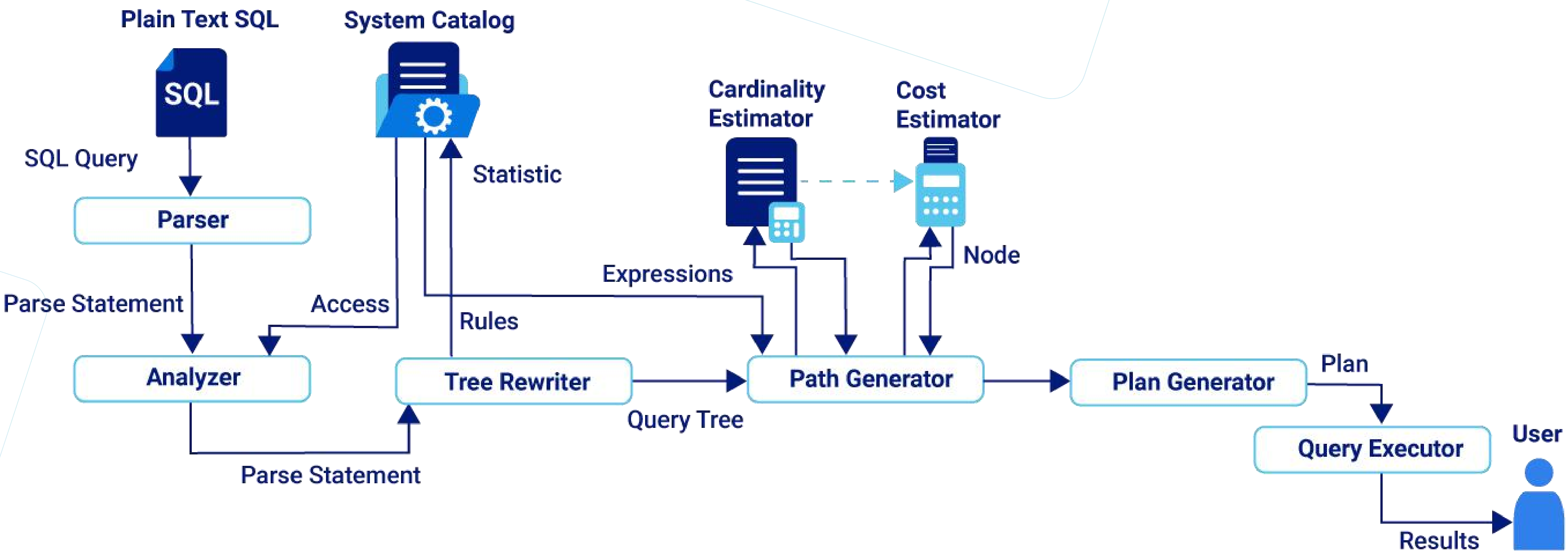
AQO query text

It stores all known queries and it's hashes and query texts

Execution statistics

It stores information about execution and planning time, cardinality error between predicted and actual number of rows – everything gathered with AQO and without AQO

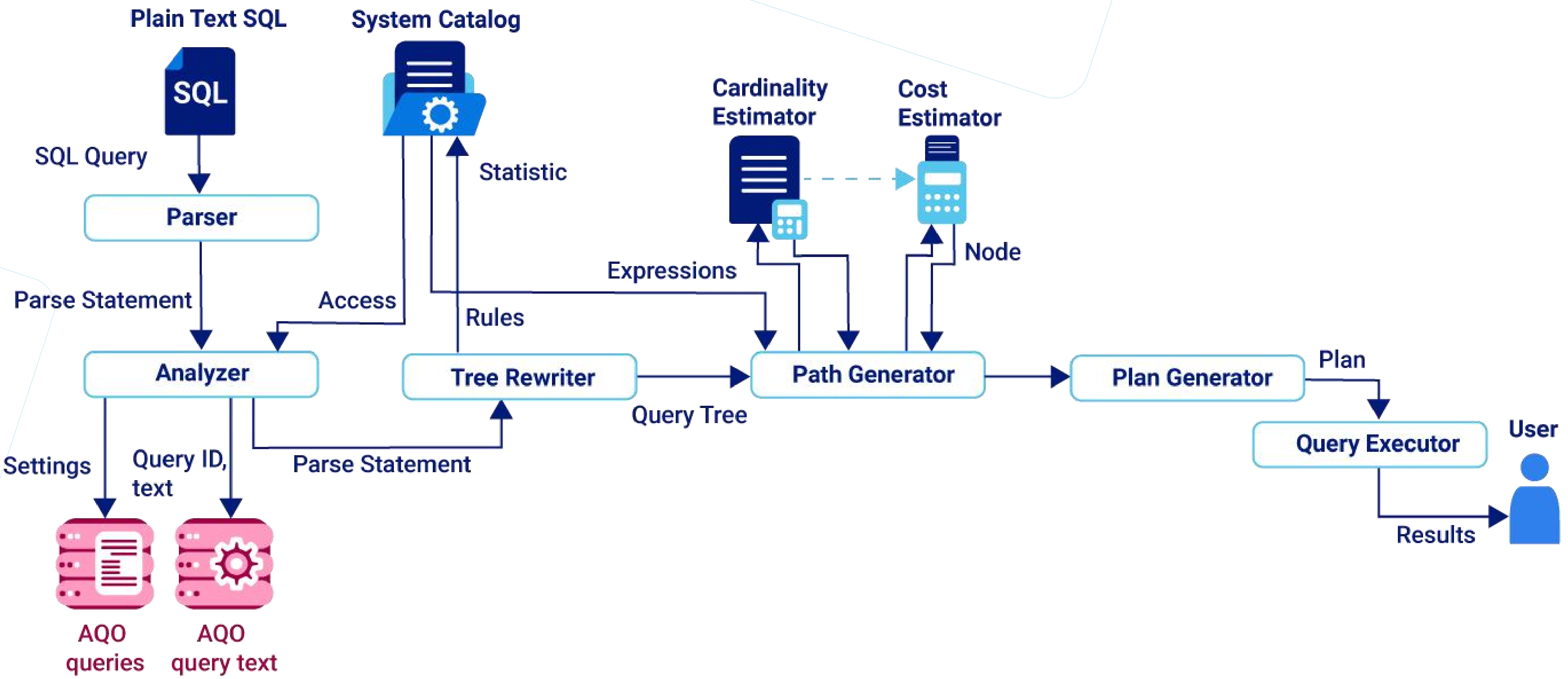
How Optimizer Works



1. How does AQO work?

2. Problems & Features

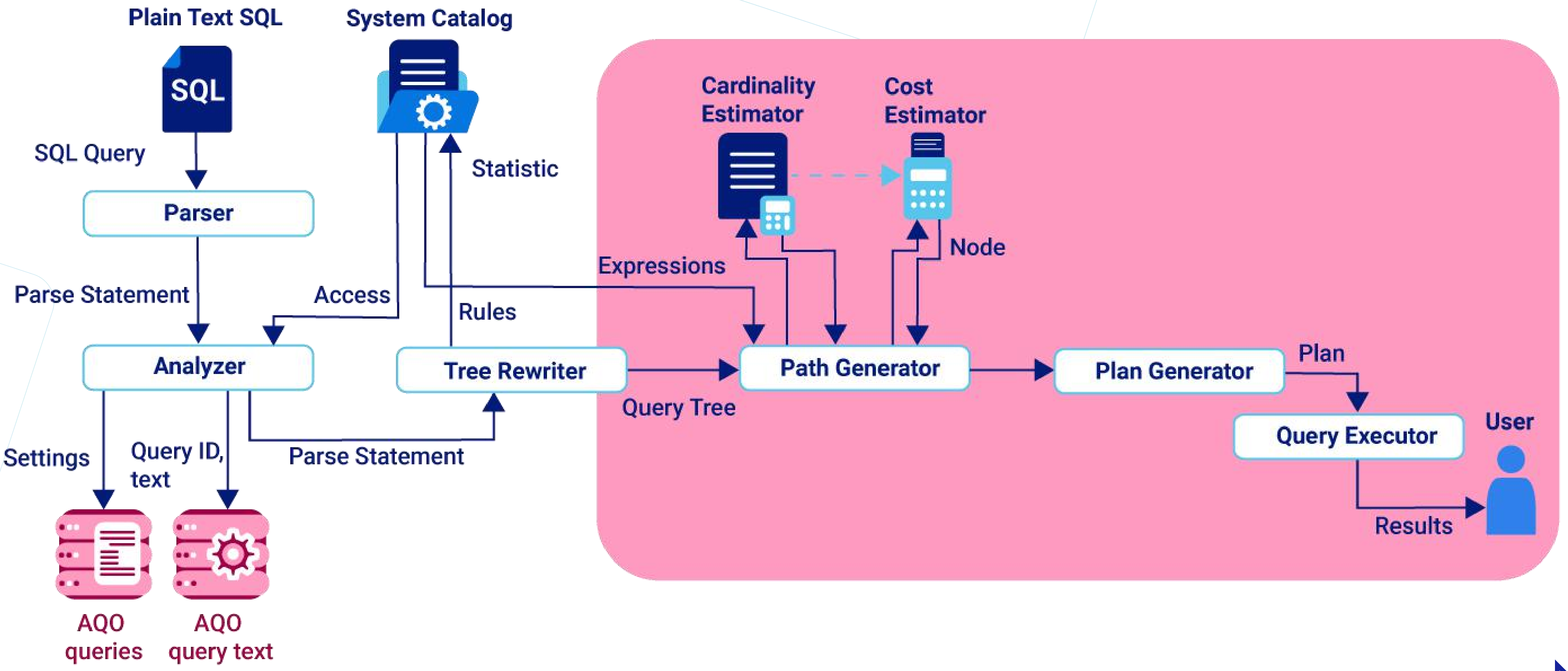
How AQO Works



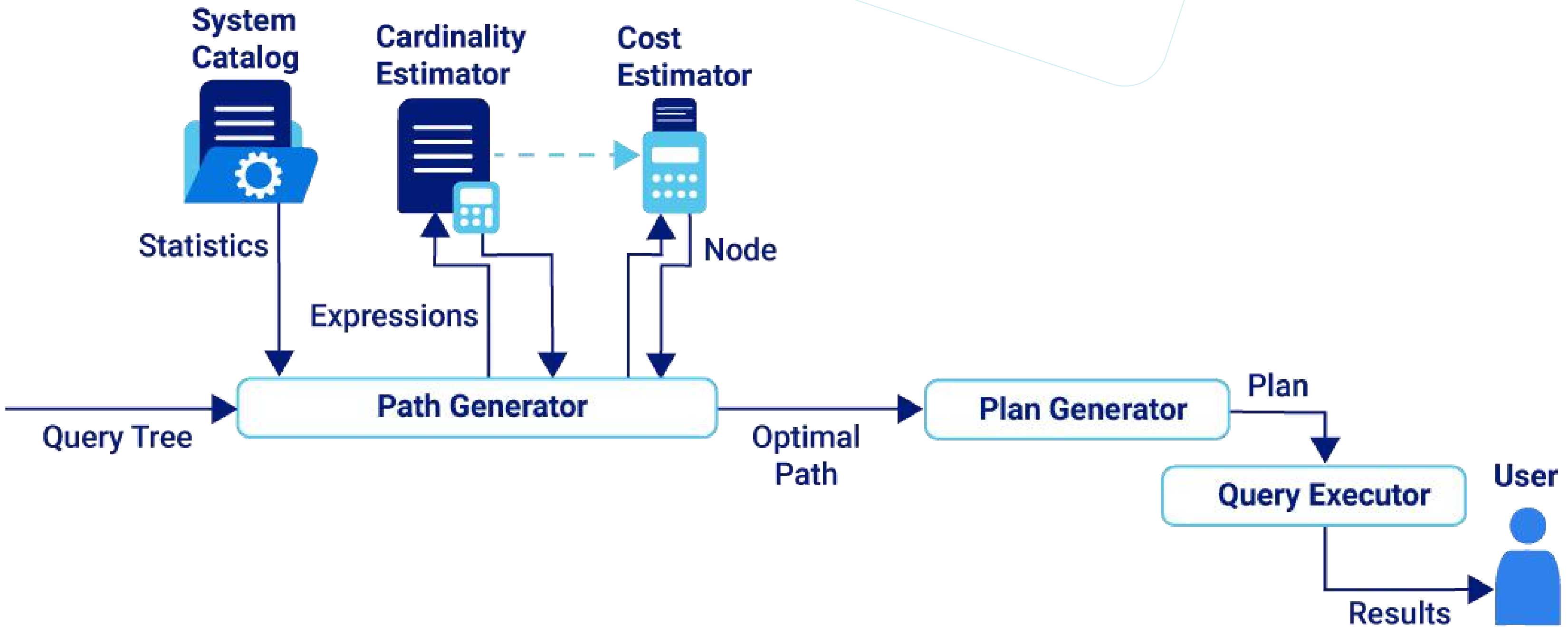
1. How does AQO work?

2. Problems & Features

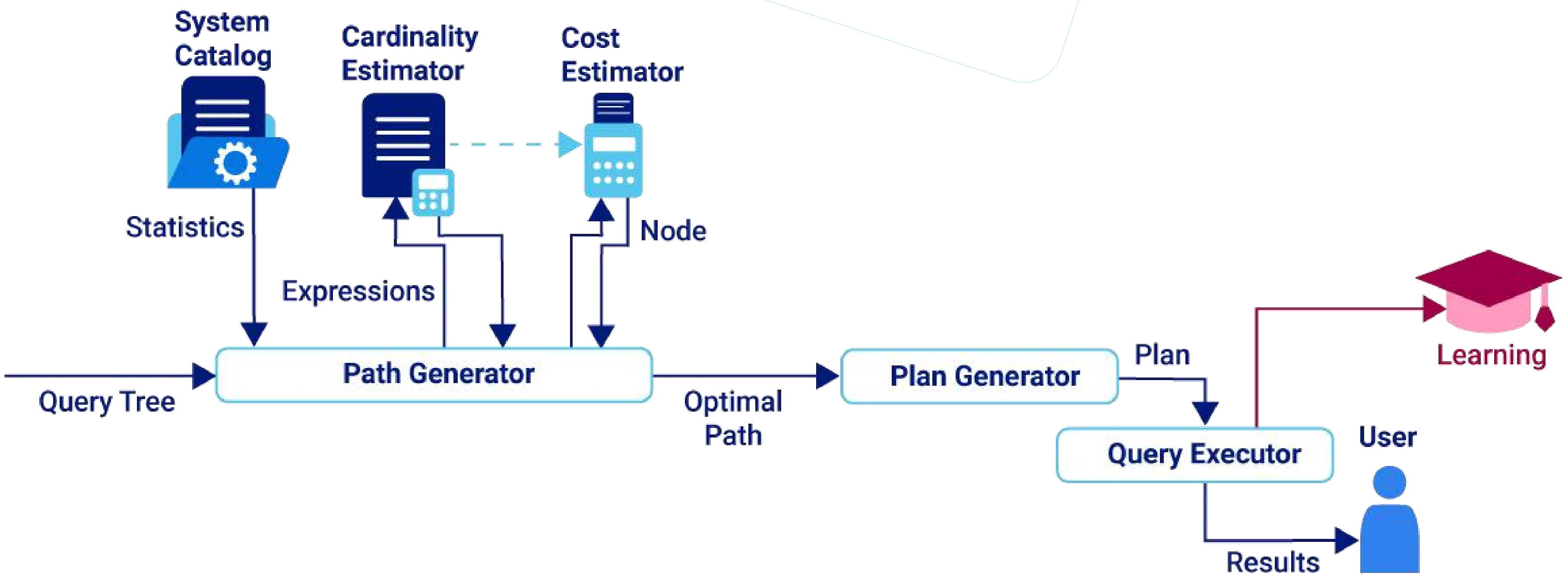
How AQO Works



How AQO Works: Collecting Statistics

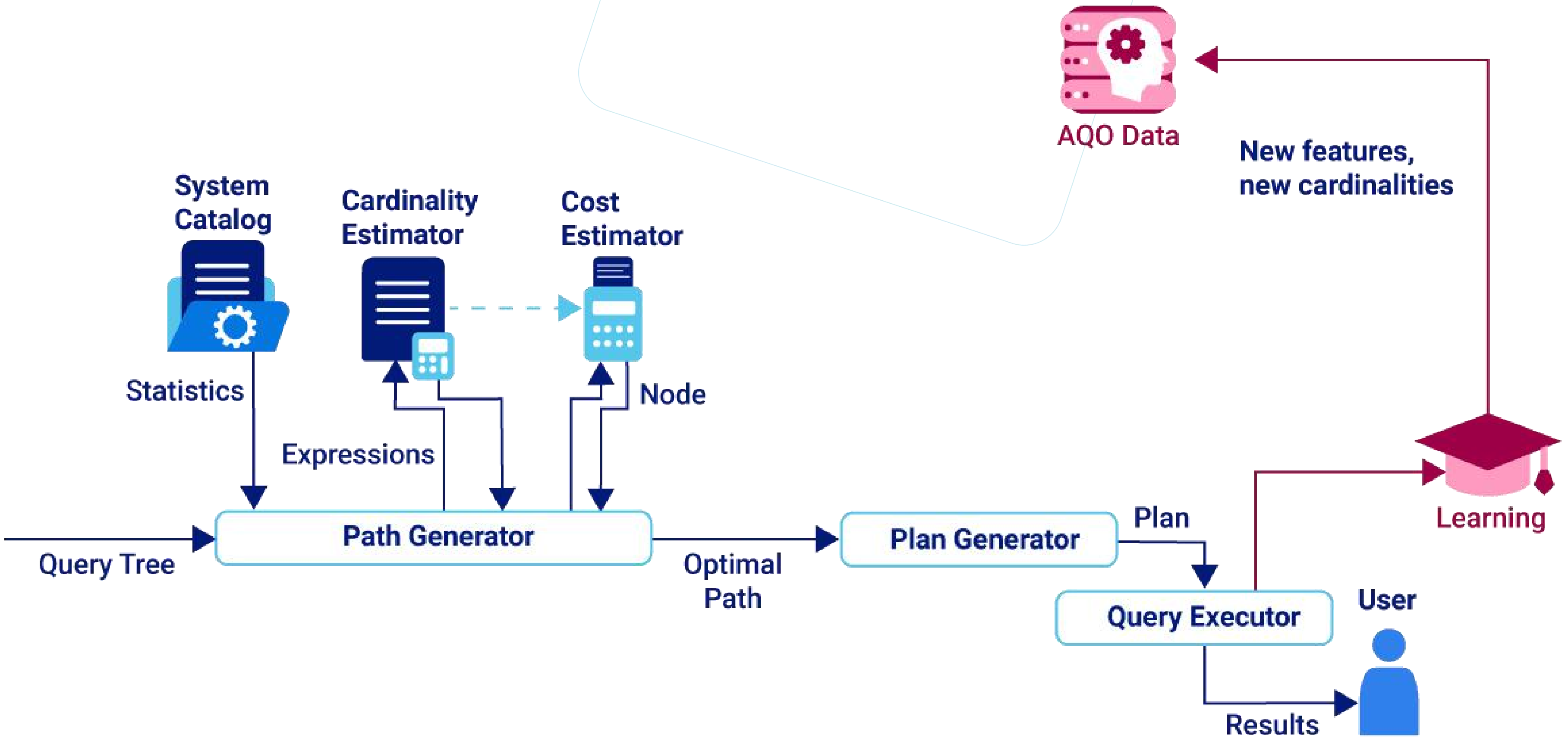


How AQO Works: Collecting Statistics



How AQO Works: Collecting Statistics

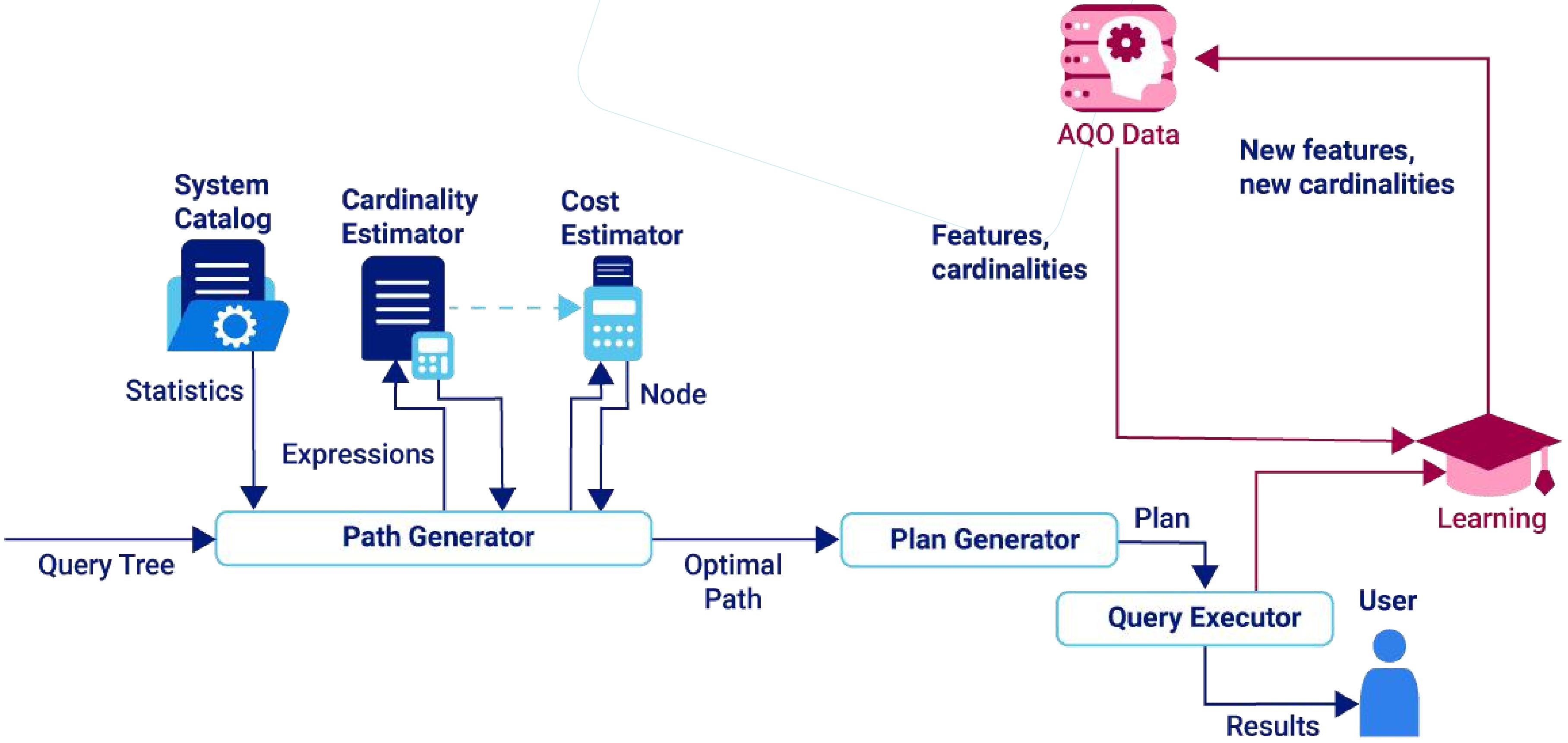
PGConf.dev



1. How does AQO work?

2. Problems & Features

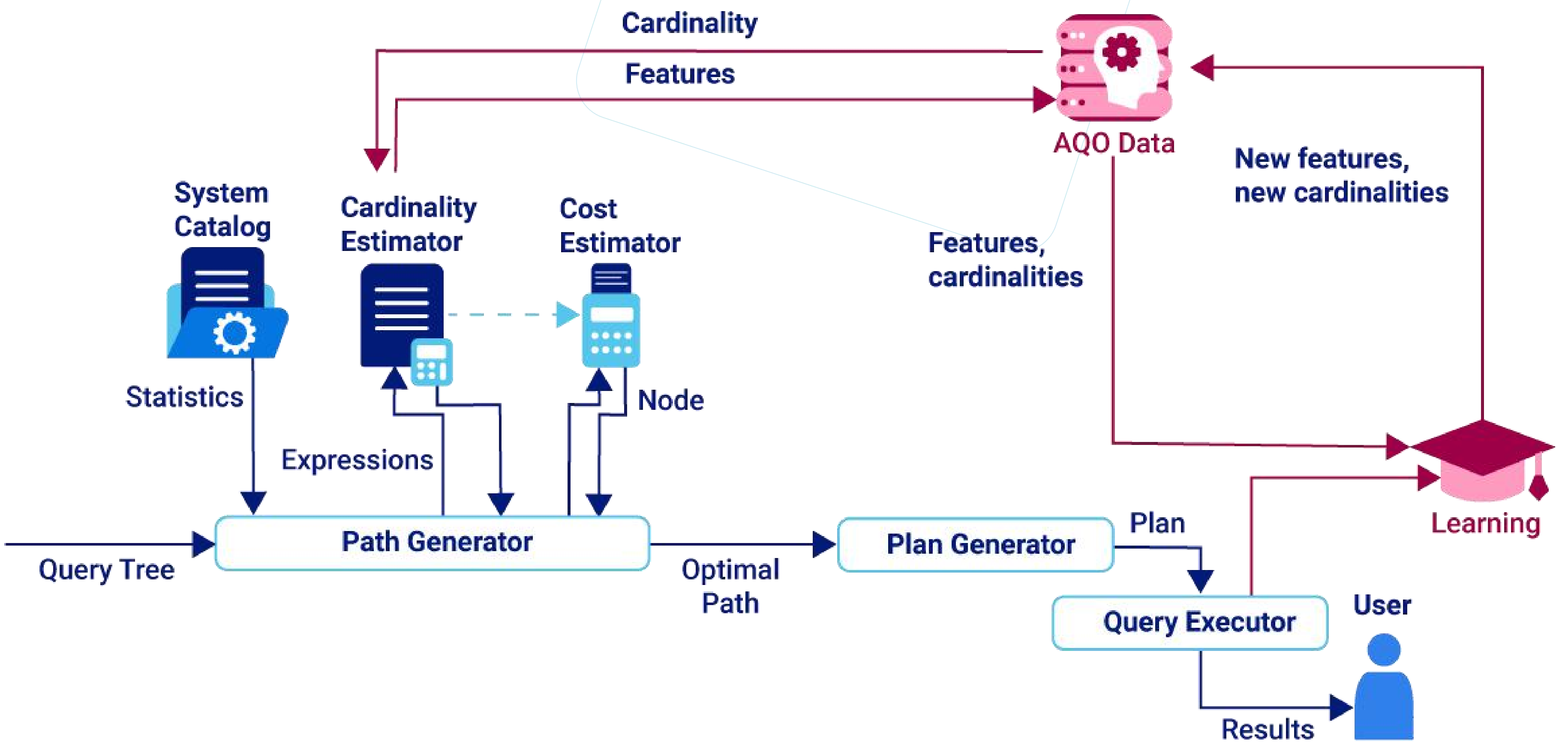
How AQO Works: Collecting Statistics



1. How does AQO work?

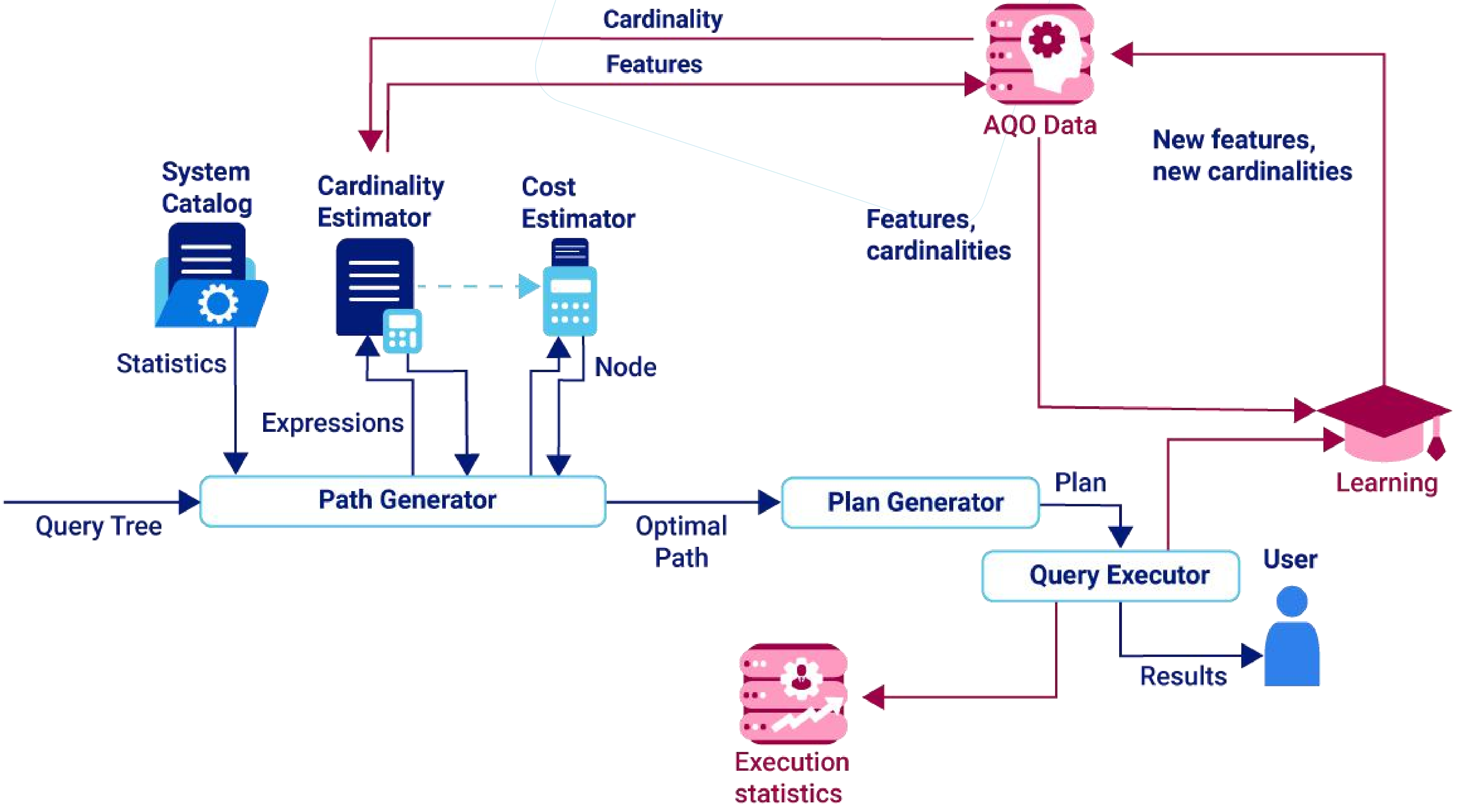
2. Problems & Features

How AQO Works: Collecting Statistics

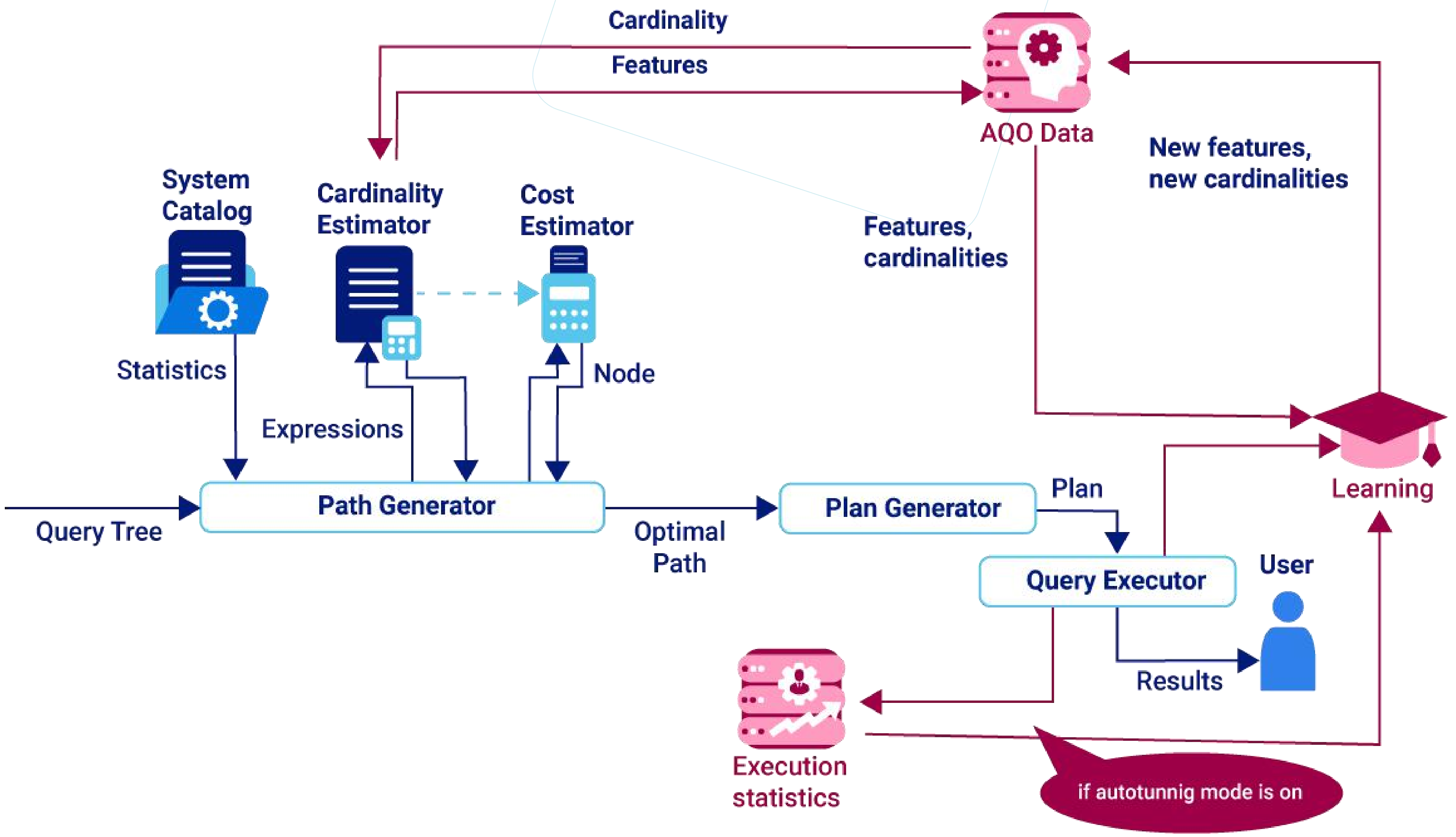


1. How does AQO work? | 2. Problems & Features

How AQO Works: Collecting Statistics



How AQO Works: Collecting Statistics



1. How does AQO work?

2. Problems & Features

Hooks

PGConf.dev

Planning stage (prediction):

- set_baserel_size_estimates
- set_joinrel_size_estimates
- set_foreign_rows_estimate
- get_parameterized_baserel_size
- get_parameterized_joinrel_size
- estimate_num_groups

After–execution stage:

- ExecutorEnd – learning
- ExplainOnePlan – visualization

Other:

- planner_hook – prepare to the planning stage
- ExecutorStart – setting the flags for statistics collection
- copy_generic_path_info – transmit Path information to Plan node
- create_plan_hook – transmit Plan information to the Execution stage for learning

Hooks

PGConf .dev

Planning stage (prediction):

- set_baserel_size_estimates
- set_joinrel_size_estimates
- set_foreign_rows_estimate
- get_parameterized_baserel_size
- get_parameterized_joinrel_size
- estimate_num_groups

After–execution stage:

- ExecutorEnd – learning
- ExplainOnePlan – visualization

transmit cardinality information from the optimizer to the AQO and vice versa

Other:

- planner_hook – prepare to the planning stage
- ExecutorStart – setting the flags for statistics collection
- copy_generic_path_info – transmit Path information to Plan node
- create_plan_hook – transmit Plan information to the Execution stage for learning

Hooks

PGConf.dev

Planning stage (prediction):

- set_baserel_size_estimates
- set_joinrel_size_estimates
- set_foreign_rows_estimate
- get_parameterized_baserel_size
- get_parameterized_joinrel_size
- estimate_num_groups

After–execution stage:

- ExecutorEnd – learning
- ExplainOnePlan – visualization

visualize debugging information about the Query plan with the AQO

Other:

- planner_hook – prepare to the planning stage
- ExecutorStart – setting the flags for statistics collection
- copy_generic_path_info – transmit Path information to Plan node
- create_plan_hook – transmit Plan information to the Execution stage for learning

Problems & features

Incompletely Executed Node

- Presence of a limit node in the query plan
- No rows in one of the subnodes of the connection node

Incompletely Executed Node

PGConf.dev

- Presence of a limit node in the query plan
- No rows in one of the subnodes of the connection node

We shouldn't save the actual data and allow to learn from them

Incompletely Executed Node

PGConf.dev

- Presence of a limit node in the query plan
- No rows in one of the subnodes of the connection node

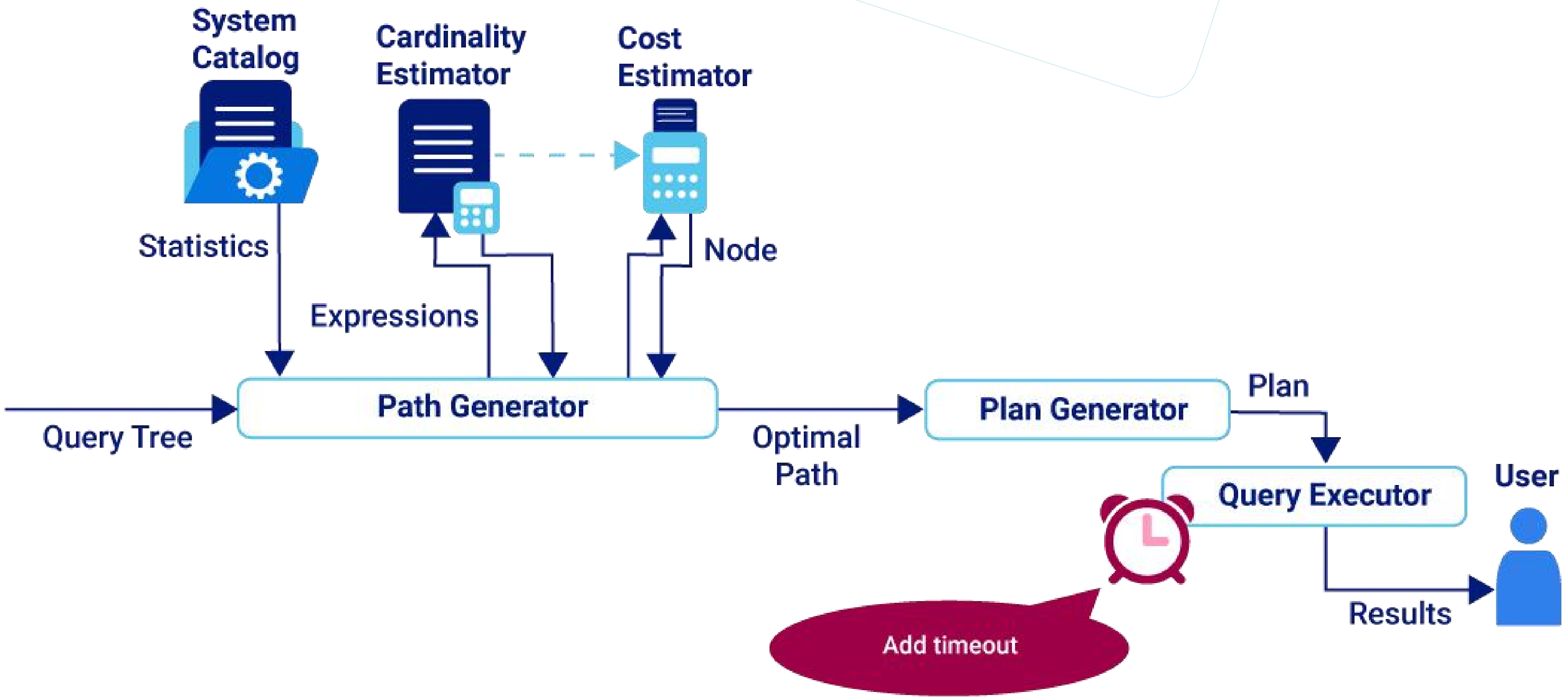
We shouldn't save the actual data and allow to learn from them

In progress...

Incompletely Executed Node

- Presence of a limit node in the query plan
- No rows in one of the subnodes of the connection node
- Query execution time limit (statement timeout)

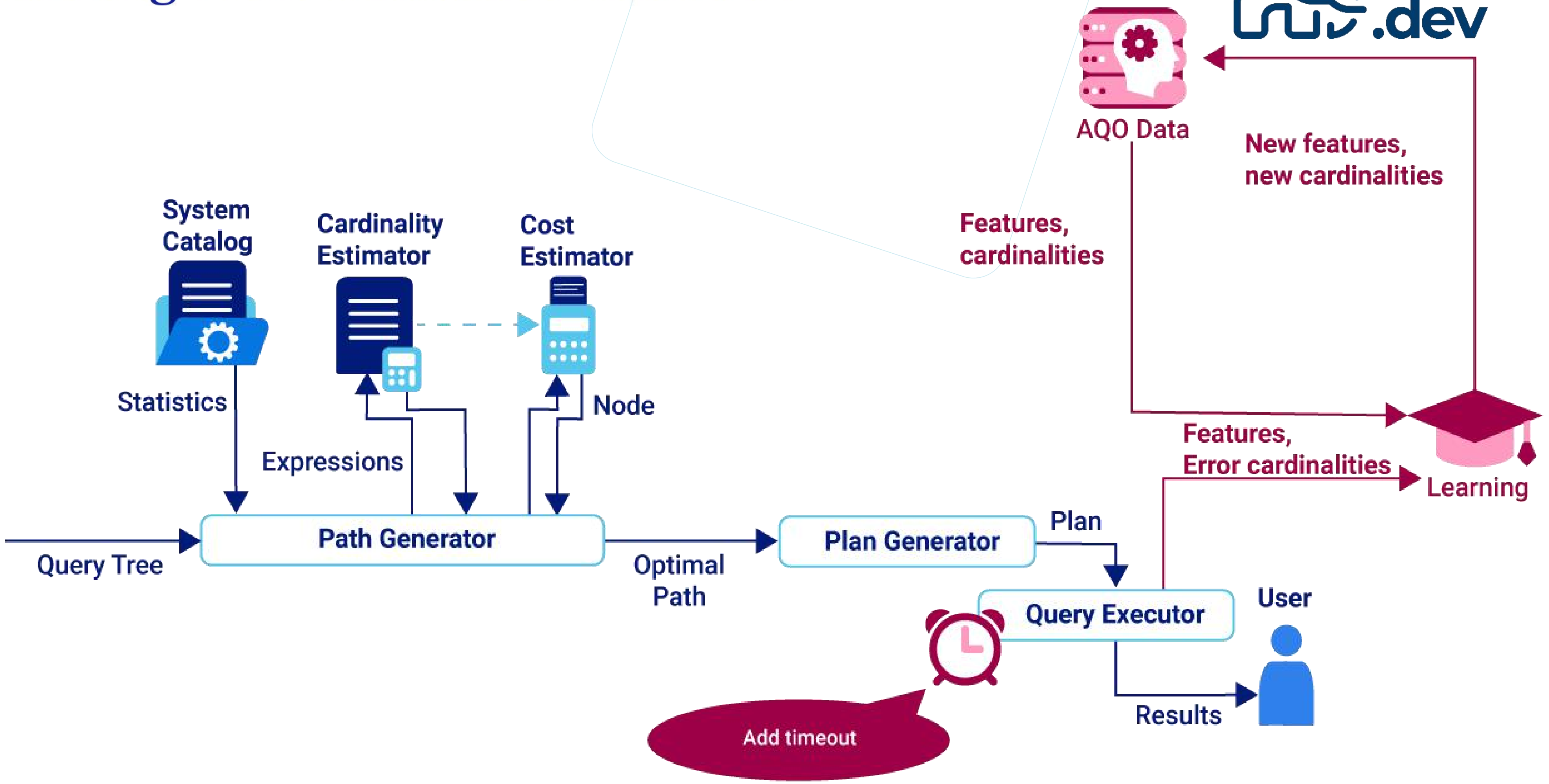
Statement timeout



2. Problems & Features

3. Examples

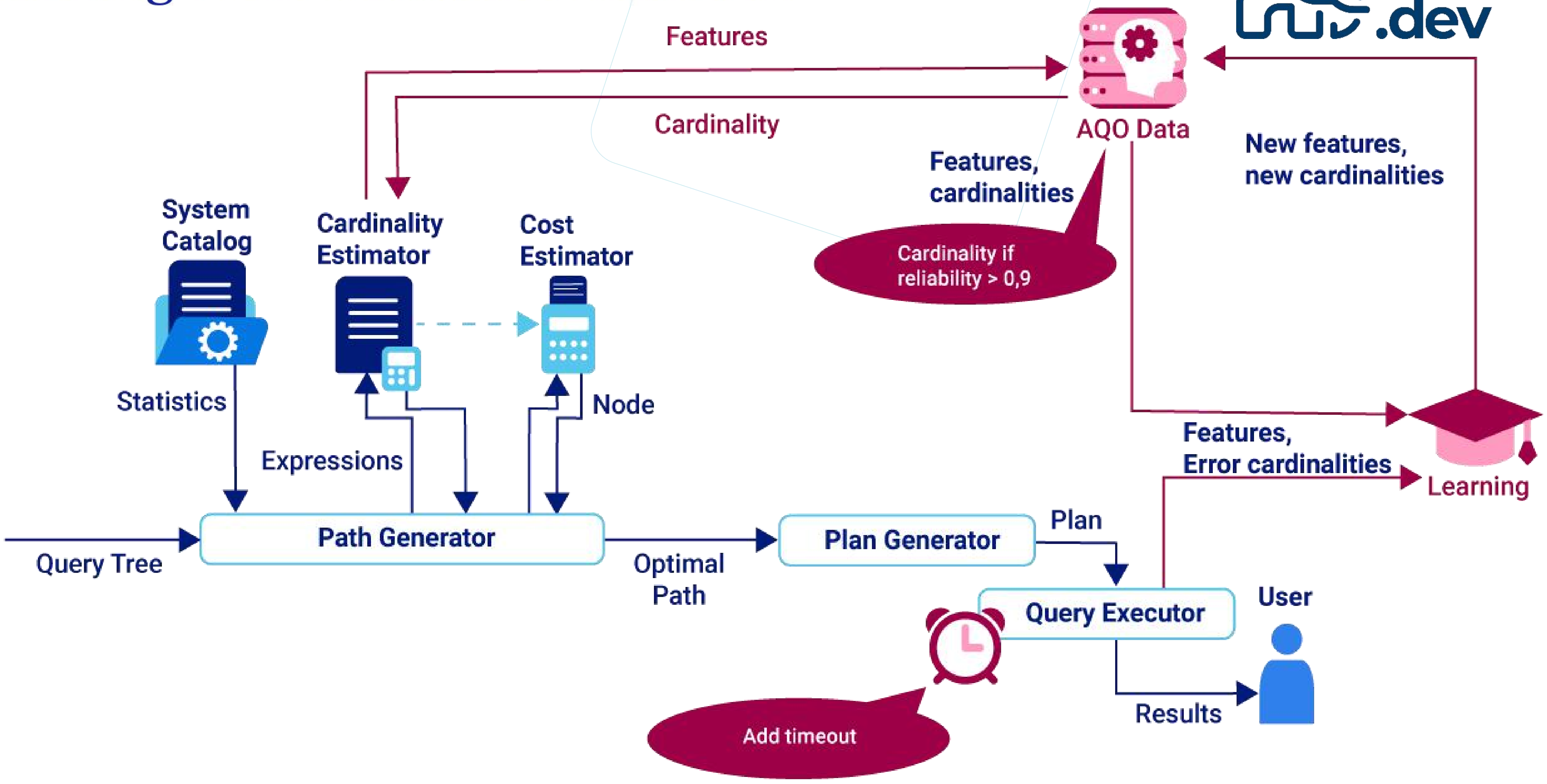
AQO Learning With Statement Timeout



2. Problems & Features

3. Examples

AQO Learning With Statement Timeout



2. Problems & Features | 3. Examples

Look–a–like Feature

- A new query run
- A lot of information about similar queries

Look–a–like Feature

- A new query run
- A lot of information about similar queries
- AQO uses it to make a prediction

Look–a–like Feature

- Try to find the closest nodes from any kind of queries

Look–a–like Feature

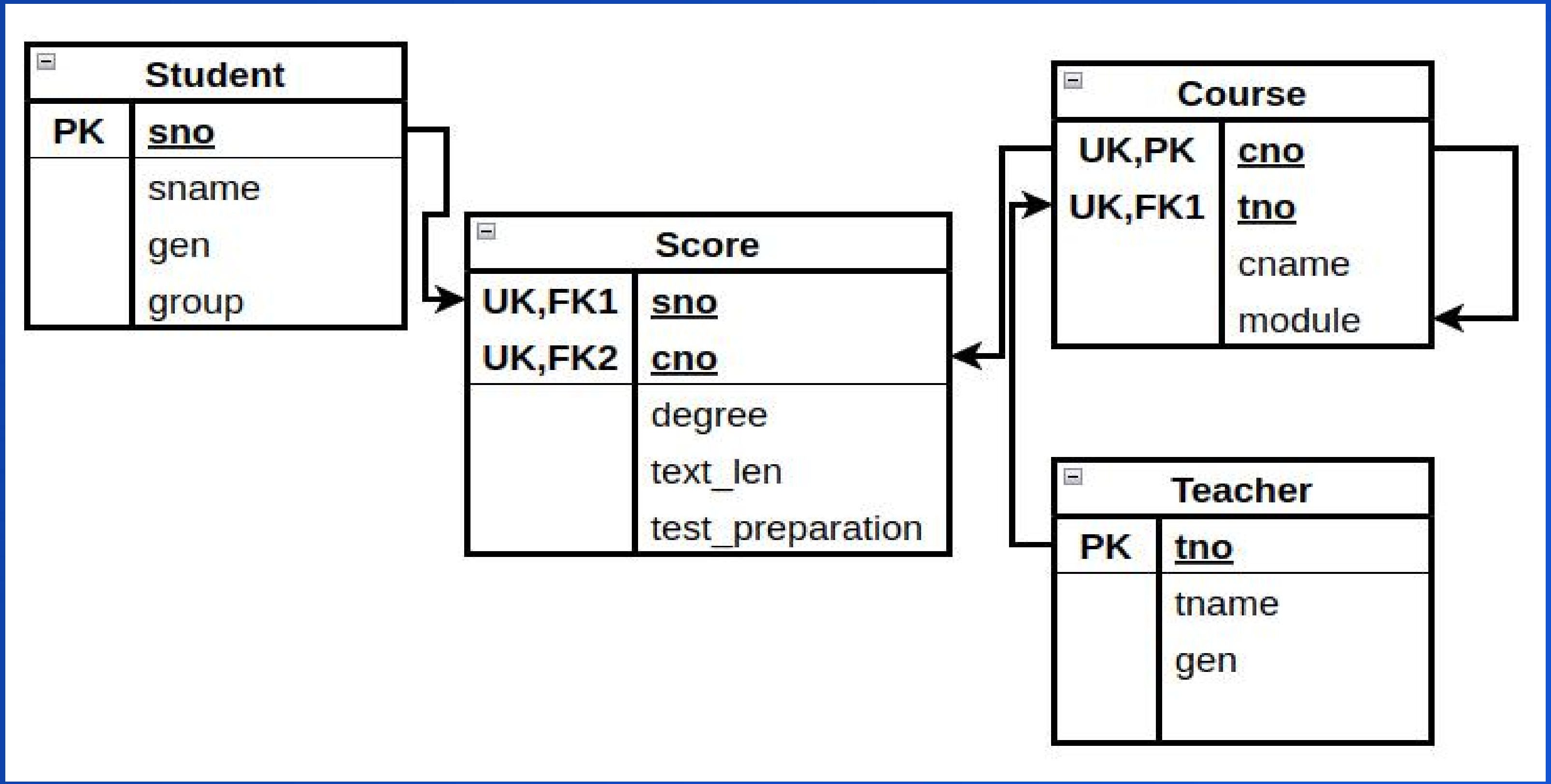
PGConf.dev

- Try to find the closest nodes from any kind of queries

- Avoid collision – consider neighbours only with the same relation oids and number of clauses

Examples

Examples



You can find the database here:
https://github.com/Alena0704/Test-AQO/tree/main

AQO Terminology

- feature space (fs) – the space where statistics on this class of queries are collected
- feature subspace (fss) – the space where information about selectivity and cardinality of every node are collected for each item of feature space

3. Examples

4. Testing results

AQO Terminology

PGConf .dev

SELECT * FROM STUDENT WHERE GROUP = 'classA';

SELECT * FROM STUDENT WHERE GROUP = 'classB';

fs1

SELECT * FROM STUDENT WHERE gen = 'female';

fs2

AQO Terminology

SELECT * FROM STUDENT WHERE GROUP = 'classA';

SELECT * FROM STUDENT WHERE GROUP = 'classB';

```
-> Seq Scan on student
      Filter: (group = classA)                     } fss1

-> Index Scan using student_idx1 on student
      Filter: (group = classA)                     } fss2

-> Index Scan using student_idx1 on student
      Filter: (group = classB)
```

1. Functional Dependences

PGConf.dev

```
explain analyze select cname, avg(degree)
        from score, course
        where test_preparation=1 and
                degree>90
        group by (cname);
------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=6710.96..6712.96 rows=10 width=78) (actual time=27.632..29.604 rows=10 loops=1)
   Group Key: course.cname
   ->  Sort  (cost=6710.96..6711.58 rows=250 width=50) (actual time=27.407..27.954 rows=10870 loops=1)
         ...
         ->  Nested Loop  (cost=1000.00..6701.00 rows=250) (actual time=20.113..24.585 rows=10870 loops=1)
             ...
              ->  Materialize  (cost=0.00..1.15 rows=10) (actual time=0.000..0.001 rows=10 loops=1087)
                    ->  Seq Scan on course  (cost=0.00..1.10 rows=10 width=46) (actual time=0.015..0.017 rows=10 loops=1)
```

	degree	essay_text_len	clevel	sgen	sgroup	test_preparation
degree	1.000	0.491	0.012	0.000	0.318	0.917
essay_text_len	0.491	1.000	0.000	0.000	0.286	0.789
clevel	0.012	0.000	1.000	0.000	0.000	0.000
sgen	0.000	0.000	0.000	1.000	0.018	0.000
sgroup	0.318	0.286	0.000	0.018	1.000	0.294
test_preparation	0.917	0.789	0.000	0.000	0.294	1.000

1. Functional Dependences

AQO: 2 iterations

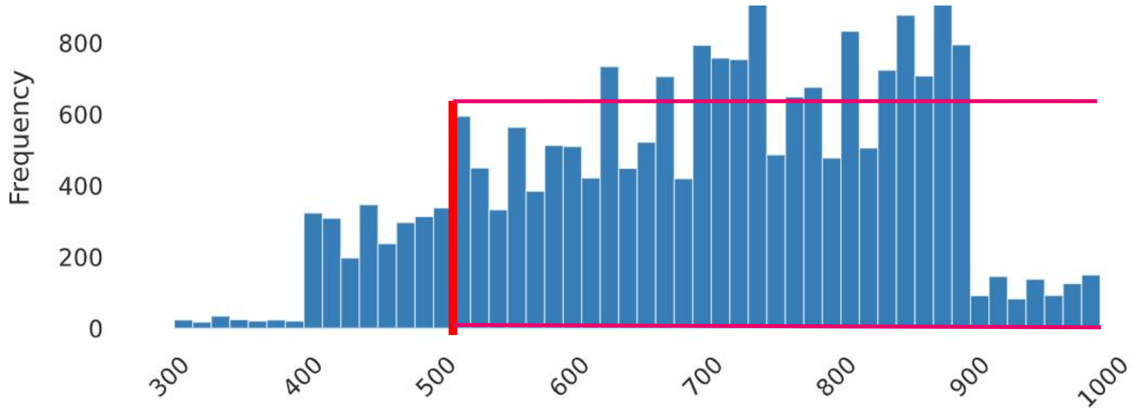
```
explain analyze select cname, avg(degree)
        from score, course
        where test_preparation=1 and
                degree>90
        group by (cname);
-------------------------------------------------------------------------------------------
HashAggregate  (cost=6994.85..6994.97 rows=10 width=78) (actual time=33.869..33.930 rows=10 loops=1)
   AQO: rows=10, error=0%, fss=1419871189
      Group Key: course.cname
   ->  Nested Loop  (cost=1000.00..6940.40 rows=10890) (actual time=25.147..30.012 rows=10870 loops=1)
         AQO: rows=10890, error=0%, fss=-882375677
        ....
         ->  Materialize  (cost=0.00..1.15 rows=10 width=46) (actual time=0.000..0.001 rows=10 loops=1087)
               AQO: rows=10, error=0%, fss=-1076069505
               ->  Seq Scan on course  (cost=0.00..1.10 rows=10) (actual time=0.016..0.020 rows=10 loops=1)
                     AQO: rows=10, error=0%, fss=-1076069505
```

3. Examples

4. Testing results

2. Non–Uniformed Data Distribution

```
explain analyze select avg(degree), sgroup
        from score, course, student
        where essay_text_len>500 and
                course.cno=score.cno and
                student.sno = score.sno
        group by (sgroup);
---------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=2411.38..2411.44 rows=5 width=39) (actual time=170.344..170.352 rows=5 loops=1)
   Group Key: student.sgroup   Batches: 1  Memory Usage: 24kB
   ->  Hash Join  (cost=2.01..2403.53 rows=1570 width=11) (actual time=0.110..162.806 rows=18798 loops=1)
         Hash Cond: (score.cno = course.cno)
         ->  Merge Join  (cost=0.78..2396.43 rows=1570) (actual time=0.076..156.590 rows=18798 loops=1)
               Merge Cond: (score.sno = student.sno)
               ->  Index Scan on score  (cost=0.42..26280.76 rows=18901) (actual time=0.046..150.349 rows=18798 loops=1)
                     ...
```



2. Non–Uniformed Data Distribution

PGConf .dev

AQO:2 iterations

```
explain analyze select avg(degree), sgroup from score, course, student
        where essay_text_len>500 and
                course.cno=score.cno and
                student.sno = score.sno
        group by (sgroup);
------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=2528.64..2528.71 rows=5 width=39) (actual time=179.476..179.485 rows=5 loops=1)
   AQO: rows=5, error=-0%, fss=-125982366   Group Key: student.sgroup   Batches: 1  Memory Usage: 24kB
   ->  Merge Join  (cost=0.93..2434.65 rows=18798 width=11) (actual time=0.066..171.115 rows=18798 loops=1)
         AQO: rows=18798, error=0%, fss=390241325    Merge Cond: (score.sno = student.sno)
         ->  Nested Loop  (cost=0.57..26729.30 rows=18798) (actual time=0.048..164.677 rows=18798 loops=1)
               AQO: rows=18798, error=0%, fss=712494197
               ->  Index Scan on score  (cost=0.42..26280.76 rows=18798) (actual time=0.030..151.596 rows=18798 loops=1)
                     AQO: rows=18798, error=0%, fss=-217544758  Filter: (essay_text_len > 500)
                     Rows Removed by Filter: 338202
```

3. Examples

4. Testing results

2. Non–Uniformed Data Distribution

PGConf.dev

WITHOUT AQO

```
HashAggregate
  ->  Hash Join
        Hash Cond: (score.cno = course.cno)
        ->  Merge Join
              Merge Cond: (score.sno = student.sno)

              ->  Index Scan using score_idx1 on score
                    Filter: (essay_text_len > 500)
              ->  Index Scan using student_pkey on student
        ->  Hash
              ->  Seq Scan on course
```

WITH AQO

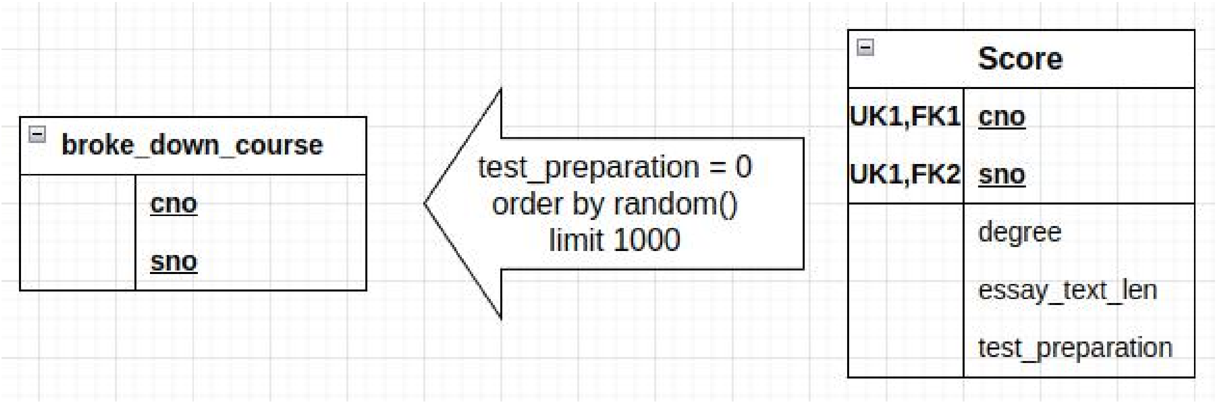
```
HashAggregate
  ->  Merge Join
              Merge Cond: (score.sno = student.sno)
        ->  Nested Loop
              ->  Index Scan using score_idx1 on score
                          Filter: (essay_text_len > 500)
              ->  Memoize
                    ->  Index Only Scan using course_pkey on course
                          Index Cond: (cno = score.cno)
        ->  Index Scan using student_pkey on student
```

3. Examples

4. Testing results

3. Outer Join

```
create table broke_down_course(cno INT, sno INT);
insert into broke_down_course
        select cno, sno from SCORE
                where test_preparation = 1 and degree < 60
        order by random() limit 5000;
insert into broke_down_course
        select cno, sno from SCORE
                where test_preparation = 0 and degree < 60
        order by random() limit 50000;
```



3. Outer Join

PGConf.dev

```
explain analyze select cname, avg(degree)
        from course, student,score
        join broke_down_course on
                (score.cno=broke_down_course.cno and score.sno=broke_down_course.sno)
        where score.sno = student.sno
        group by (cname);
------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=1688.42..1688.55 rows=10) (actual time=86.500..86.509 rows=10 loops=1)
   Group Key: course.cname
   ->  Nested Loop  (cost=91.92..1686.43 rows=399) (actual time=0.961..57.920 rows=77540 loops=1)
         ->  Nested Loop  (cost=91.92..1680.30 rows=40) (actual time=0.954..43.954 rows=7754 loops=1)
               ->  Hash Join  (cost=91.50..226.36 rows=487) (actual time=0.934..5.119 rows=7754 loops=1)
                     Hash Cond: (broke_down_course.sno = student.sno)
                     ->  Seq Scan on broke_down_course  (cost=0.00..114.10 rows=7910) (actual time=0.037..1.262 rows=7754 loops=1)
                     ->  Hash  (cost=54.00..54.00 rows=3000) (actual time=0.889..0.890 rows=3000 loops=1)
                           ->  Seq Scan on student  (cost=0.00..54.00 rows=3000) (actual time=0.006..0.417 rows=3000 loops=1)
                     ...
```

3. Examples → 4. Testing results

3. Outer Join

PGConf.dev

AQO:3 iteartions

```
explain analyze select cname, avg(degree)
from course, student,score join broke_down_course on
        (score.cno=broke_down_course.cno and score.sno=broke_down_course.sno)
where score.sno = student.sno
group by (cname);
------------------------------------------------------------------------------------------
 HashAggregate  (cost=1414.94..1415.07 rows=10 width=78) (actual time=164.494..164.504 rows=10 loops=1)
   AQO: rows=10, error=0%, fss=-651211982
   ->  Merge Join  (cost=614.28..1027.24 rows=77540 width=50) (actual time=3.038..134.968 rows=77540 loops=1)
         AQO: rows=77540, error=0%, fss=29214553
             Merge Cond: (score.sno = student.sno)
         ->  Merge Join  (cost=613.94..3362.22 rows=77540 width=58) (actual time=3.018..124.058 rows=77540 loops=1)
               AQO: rows=77540, error=0%, fss=-1852476170
               Merge Cond: ((score.sno = broke_down_course.sno) AND (score.cno = broke_down_course.cno))
               ->  Nested Loop  (cost=0.42..29139.14 rows=299971 width=58) (actual time=0.055..76.314 rows=299971 loops=1)
                     AQO: rows=299971, error=0%, fss=-2144628856
                     ...
```

3. Examples → 4. Testing results

3. Outer Join

PGConf.dev

Without AQO

```
HashAggregate
 ->  Nested Loop

      ->  Nested Loop
            ->  Hash Join
                  Hash Cond: (broke_down_course.sno = student.sno)

                  ->  Seq Scan on broke_down_course
                  ->  Hash
                        ->  Seq Scan on student
            ->  Index Scan using score_idx1 on score
                  Index Cond: ((sno = broke_down_course.sno)
                  AND (cno = broke_down_course.cno))
      ->  Materialize
            ->  Seq Scan on course
```

With AQO

```
HashAggregate
   ->  Merge Join
        Merge Cond: (score.sno = student.sno)
     ->  Merge Join
          Merge Cond: ((score.sno = broke_down_course.sno)
                  AND (score.cno = broke_down_course.cno))
          ->  Nested Loop
                ->  Index Scan using score_idx1 on score
                ->  Materialize
                      ->  Seq Scan on course
          ->  Sort
                ->  Seq Scan on broke_down_course
     ->  Index Only Scan using student_pkey on student
```

3. Outer Join

explain analyze select cname, avg(degree)
from course, student,score
join broke_down_course on
(score.cno=broke_down_course.cno and score.sno=broke_down_course.sno)
where score.sno = student.sno group by (cname);

```
HashAggregate  (rows=10) (rows=10)
  ->  Merge Join  (rows=77540) (rows=77540)
        Merge Cond: (score.sno = student.sno)
        ->  Merge Join  (rows=77540) (rows=77540)
              Merge Cond: ((score.sno = broke_down_course.sno) AND
                           (score.cno = broke_down_course.cno))
              ->  Nested Loop  (rows=299971) (rows=299971)
                    ->  Index Scan on score  (rows=29998) (rows=29998)
                    ->  Materialize  (rows=10) (rows=10)
                          ->  Seq Scan on course  (rows=10) (rows=10)
              ->  Sort  (rows=7754) (rows=77531)
                    Sort Key: broke_down_course.sno, broke_down_course.cno
                    ->  Seq Scan on broke_down_course  (rows=7754) (rows=7754)
        ->  Index Only Scan on student  (rows=3000) (rows=3000)
```

3. The Same Problem On A Difference Scale

PGConf.dev

```
-> Merge Join (rows=354965847) (rows=1484797760)
     Merge Cond: (((sm.analit_uc_nom)::text = (ob.analit_uc_nom)::text) AND ...))
     -> Sort (rows=240894) (rows=240894)
          Sort Key: sm.analit_uc_nom, sm.razd_uc, sm.vid_zapas, sm.org
          -> Seq Scan on stoimost sm (rows=240894) (rows=240894)
     -> Sort (rows=241941) (rows=1484798487)
          Sort Key: ob.analit_uc_nom, ob.razd_uc, ob.vid_zapas, ob.org
          -> Seq Scan on oborots_work ob (rows=241941) (rows=241941)
```

3. The Same Problem On A Difference Scale

PGConf.dev

```
->  Merge Join  (rows=354965847) (rows=1484797760)
      Merge Cond: (((sm.analit_uc_nom)::text = (ob.analit_uc_nom)::text) AND ...))
      ->  Sort  (rows=240894) (rows=240894)
            Sort Key: sm.analit_uc_nom, sm.razd_uc, sm.vid_zapas, sm.org
            ->  Seq Scan on stoimost sm  (rows=240894) (rows=240894)
      ->  Sort  (rows=241941) (rows=1484798487)
            Sort Key: ob.analit_uc_nom, ob.razd_uc, ob.vid_zapas, ob.org
            ->  Seq Scan on oborots_work ob  (rows=241941) (rows=241941)
```

The Reason:

- The Merge Join rewinds its inner side to the start of the current group of equal keyed tuples if the next outer tuple must be also joined to the same group.
- Explain counts those tuples twice.

You can find the thread here: bit.ly/3yyH6dx

Analysing with aqo_query_stats

PGConf .dev

SELECT * FROM aqo_query_stats \gx

```
-[ RECORD 1 ]----------------+-------------------------------------------------------------------------------
queryid                      | 7430954541387508965
execution_time_with_aqo      | {0.221163375,0.21725739,0.235732091,0.221946228,0.217616499,0.256209121,0.219321755}
execution_time_without_aqo   | {0.237385655,0.242997873,0.230060608,0.235878734,0.231573898,0.229296202,0.229547688}
planning_time_with_aqo       | {0.048900852,0.048985714,0.053167861,0.049327628,0.048804019,0.057644151,0.049276517}
planning_time_without_aqo    | {0.020790356,0.021514073,0.019026199,0.019274039,0.020245325,0.019258199,0.019515377}
cardinality_error_with_aqo   | {0.03850817669777474,0.03850817669777474,0.03850817669777474,0.03850817669777474}
cardinality_error_without_aqo | {0.960947753415567,0.960947753415567,0.960947753415567,0.960947753415567,0.960947753415567}
executions_with_aqo          | 49
executions_without_aqo       | 15
-[ RECORD 2 ]----------------
 +------------------------------------------------------------------------------------------------
queryid                      | -3495764495604230484
execution_time_with_aqo      | {1.575004969,1.686475542,1.497201844,1.574961415,1.710951376,1.625525643,1.658347755}
execution_time_without_aqo   | {0.983308019,0.961930579,0.838651462,1.415978422,0.834555689,0.913765313,0.787577022}
planning_time_with_aqo       | {0.059669438,0.061208187,0.057022197,0.054507226,0.074582017,0.054978341,0.057604733}
planning_time_without_aqo    | {0.023877627,0.023690638,0.025620636,0.02337944,0.023244195,0.024100254,0.024150521}
cardinality_error_with_aqo   | {0.03850817669777474,0.03850817669777474,0.03850817669777474,0.03850817669777474}
cardinality_error_without_aqo | {0.9322820300116723,0.9322820300116723,0.9322820300116723,0.9322820300116723}
executions_with_aqo          | 49
executions_without_aqo       | 14
```

3. Examples

4. Testing results

Control queries with AQO

```
postgres=# SELECT count(*) FROM
      (SELECT queryid AS id FROM aqo_queries) AS q1,
      LATERAL aqo_queries_update(7799030661291734910, NULL, true, false, false);
                                 ⇧                     ⇧     ⇧     ⇧      ⇧
                                 fs                   fss  learn  use  autotunning
 count
-------
    99
(1 row)
```

3. Examples

4. Testing results

Modes

PGConf .dev

AUTO	Intelligent: when the cardinality error remains sufficiently small and stable for several only learned successive executions of a query, aqo turns on use_aqo
DISABLED	Disabled: disabled at all query types
LEARN	Learn: enabled for learning for every query types
PREDICTION	Forced: enabled for all query types Controlled: only learns and makes predictions for known queries Frozen: makes predictions for known queries, but does not learn from any queries

How does AQO work?

Problems & Features

AQO's storage structure

aqo_data	aqo_queries	aqo_query_text	aqo_query_stat
• Feature space (Queries) • Feature subspace (Nodes) • NFeatures • Features (Selectivities) • Targets (Rows) • Oids of relations	• Query hash • Learn AQO • Use AQO • Feature space (Query hash) • Auto tuning	• Query hash • Query text	• Queryid • Execution time with AQO • Execution time without AQO • Planning time with AQO • Planning time without AQO • Cardinality error with AQO • Cardinality Error without AQO • Executions with AQO • Executions without AQO
It stores selectivities for every query statement and it's number of rows	**Settings for all known queries**	**It stores all known queries and it's hashes**	**For analysis of working AQO**

3. Examples

4. Testing results

Testing results

Internet Movie Database (IMDB)

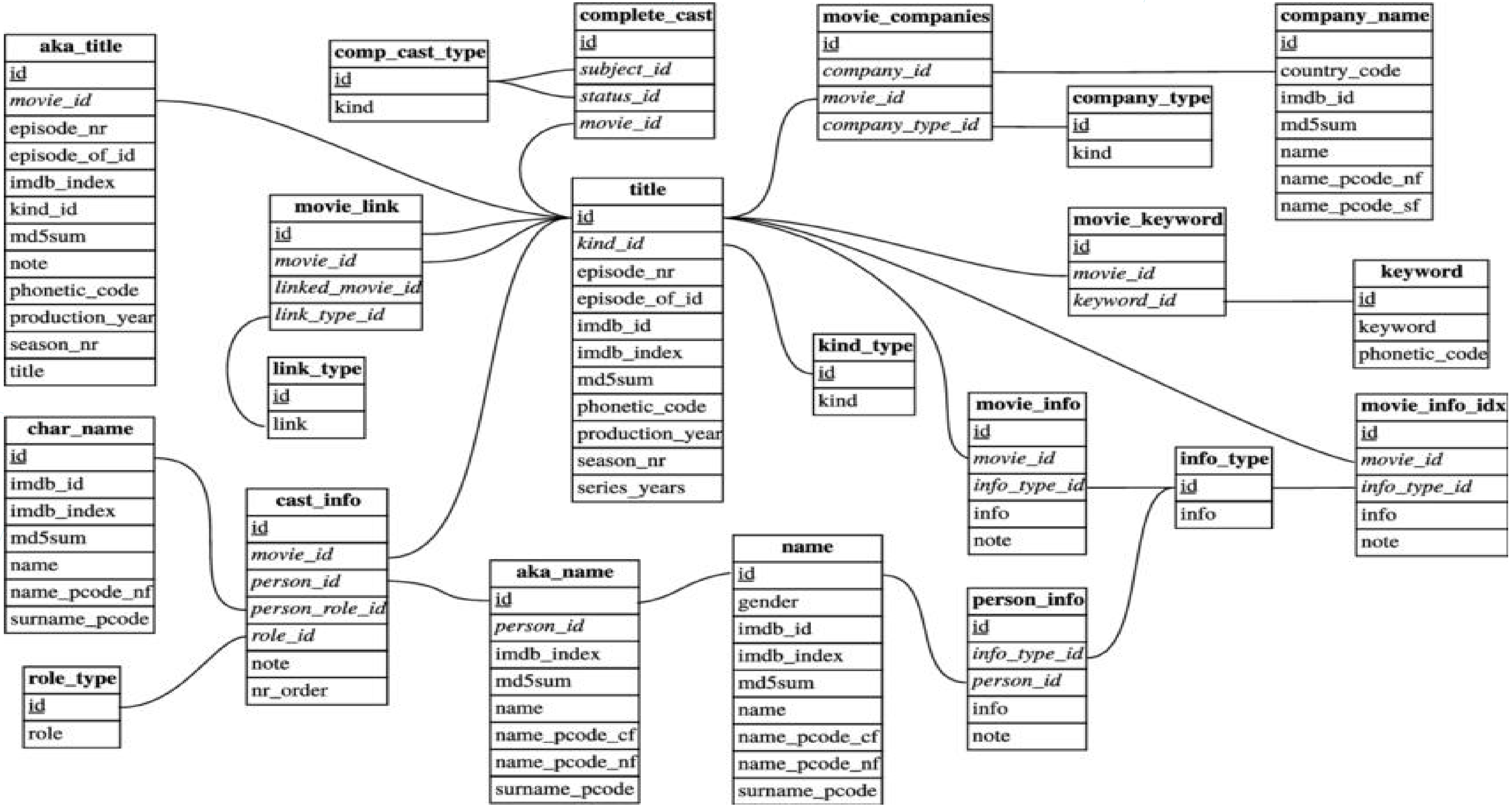
PGConf .dev

- set of 113 queries
- every query have from 3 to 16 joins
- the queries answer the logical questions of a movie lover
- queries are difficult for the optimizer due to the large number of joins and correlations

You can find the thread here: bit.ly/4bCE5ru

Internet Movie Database (IMDB)

PGConf.dev



4. Testing results

Conclusion

Tests On Join Order Benchmark

main parameters on all stages:

- random/seq_page_cost = 1
- from/join_collapse_limit = 4

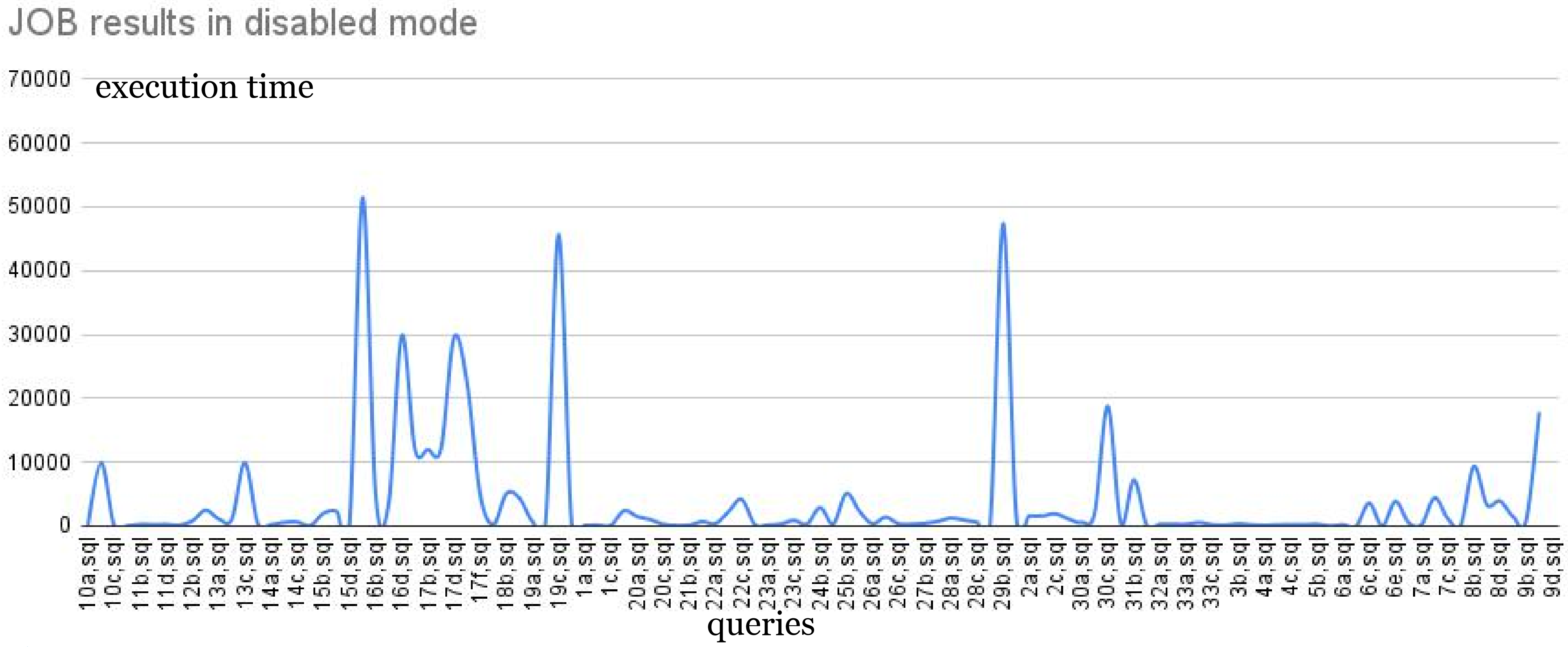
parameters on learning stage:

- disable parallelism

disabled, frozen stages:

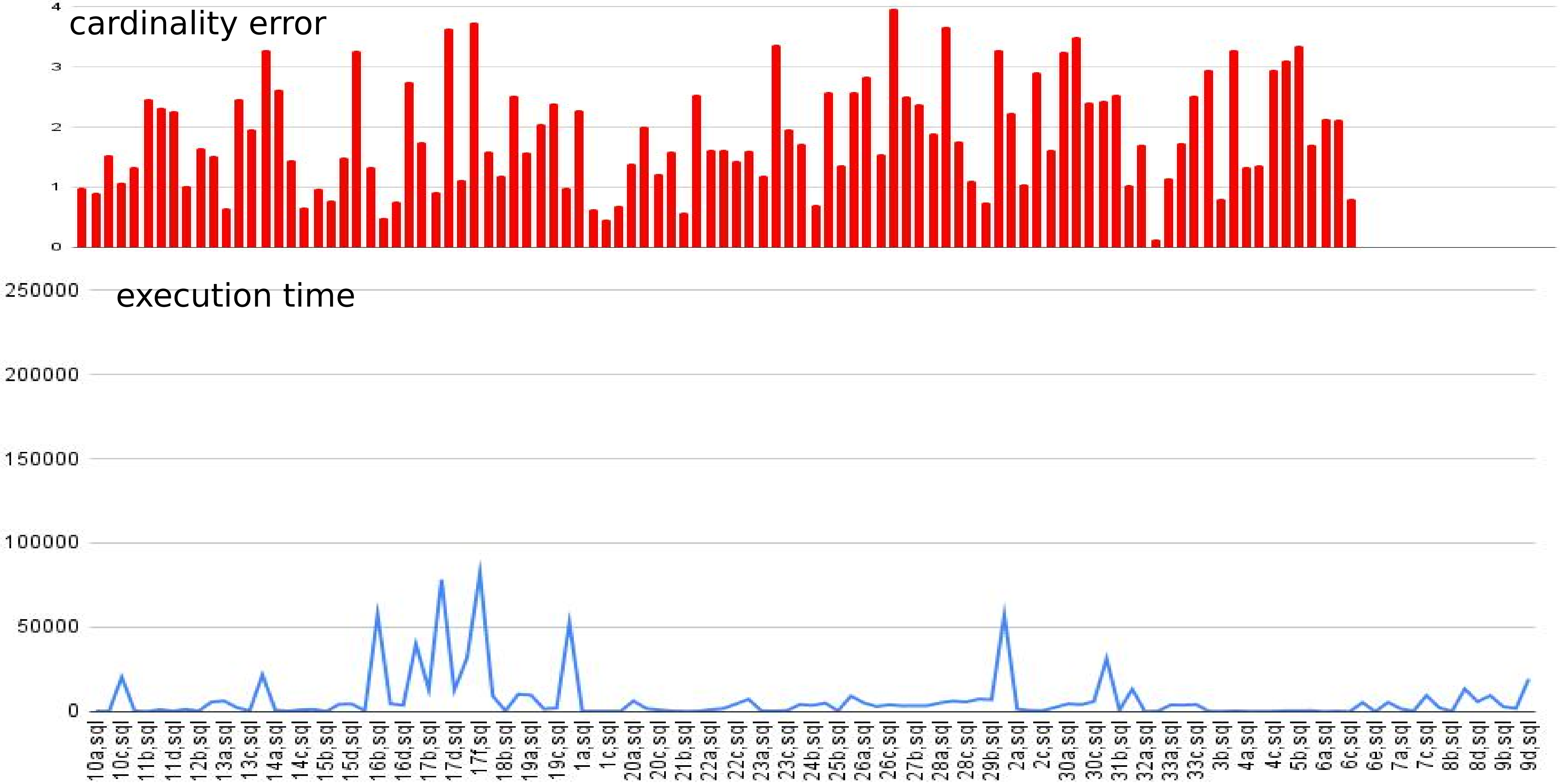
- enable parallelism

JOB Results In Disabled Mode

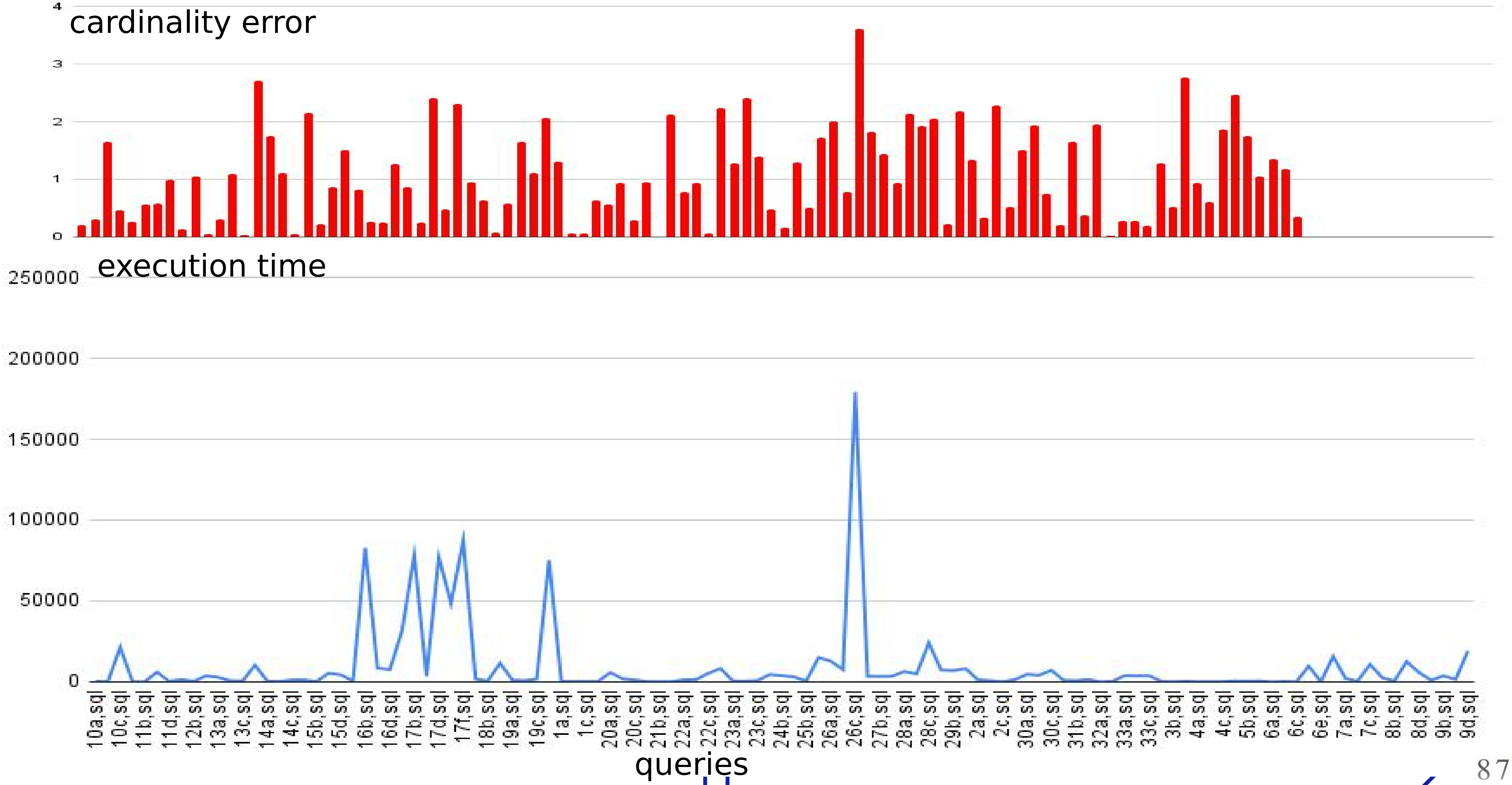


4. Testing results → Conclusion

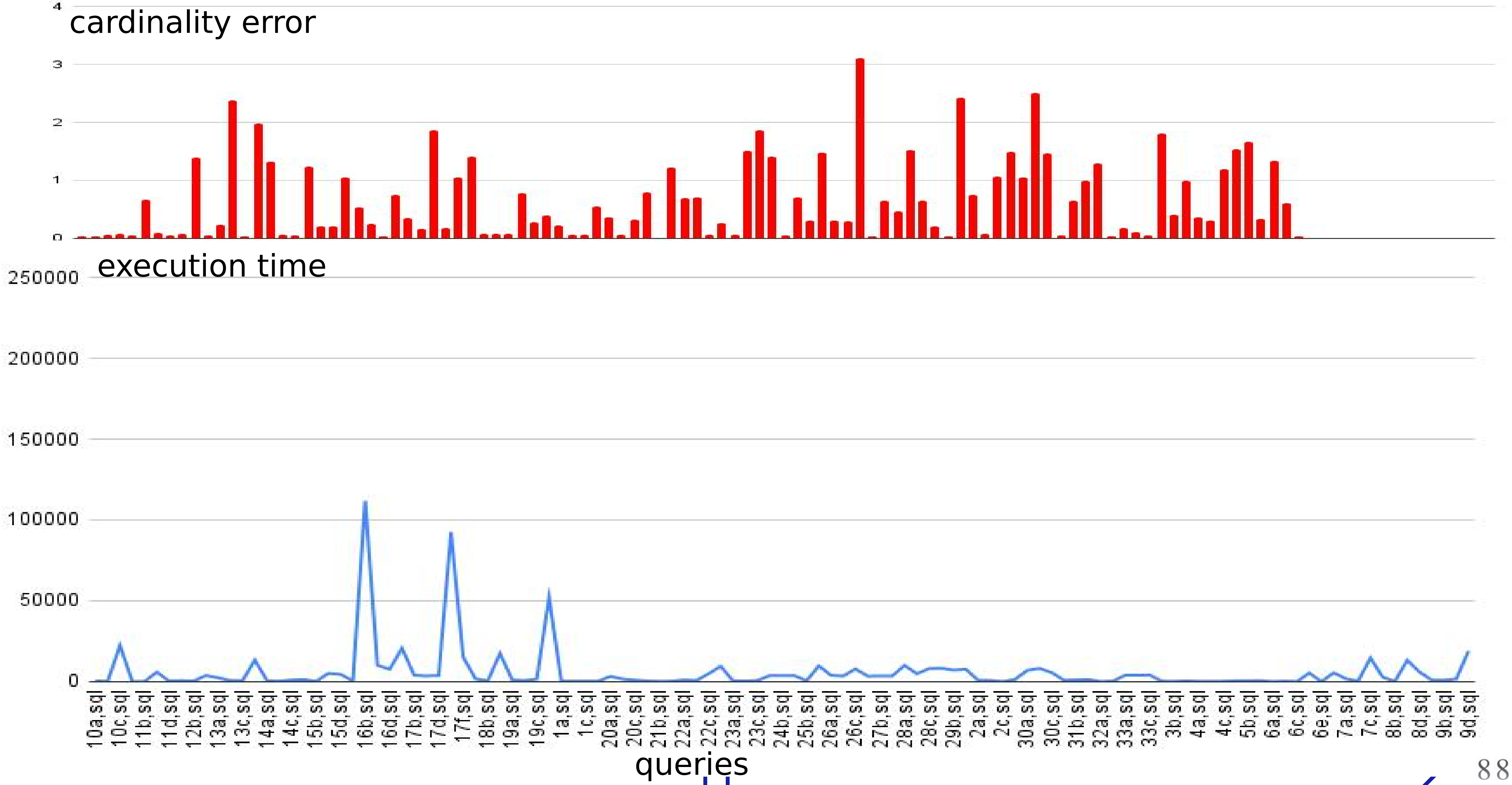
JOB Results In Learn Mode



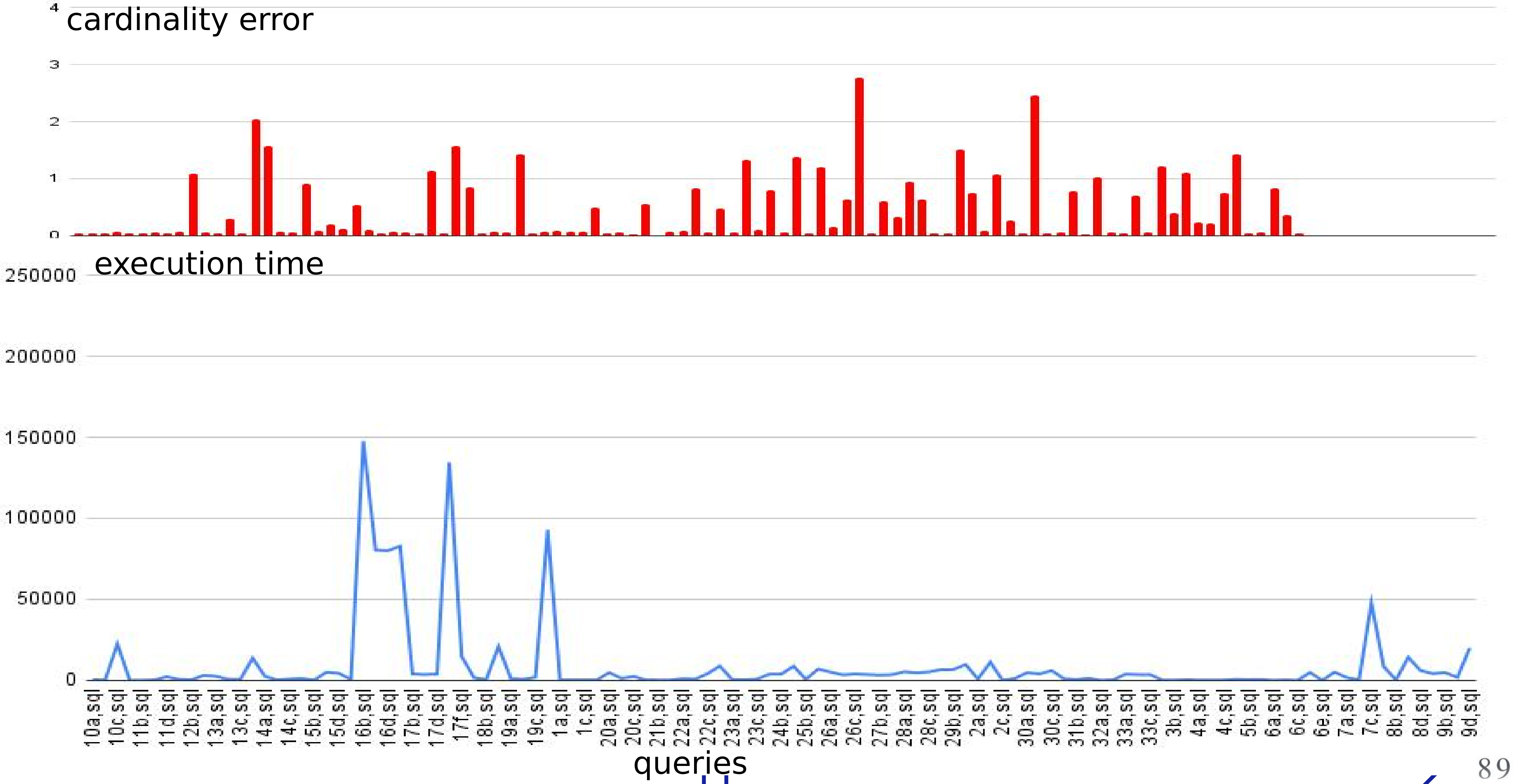
JOB Results In Learn Mode



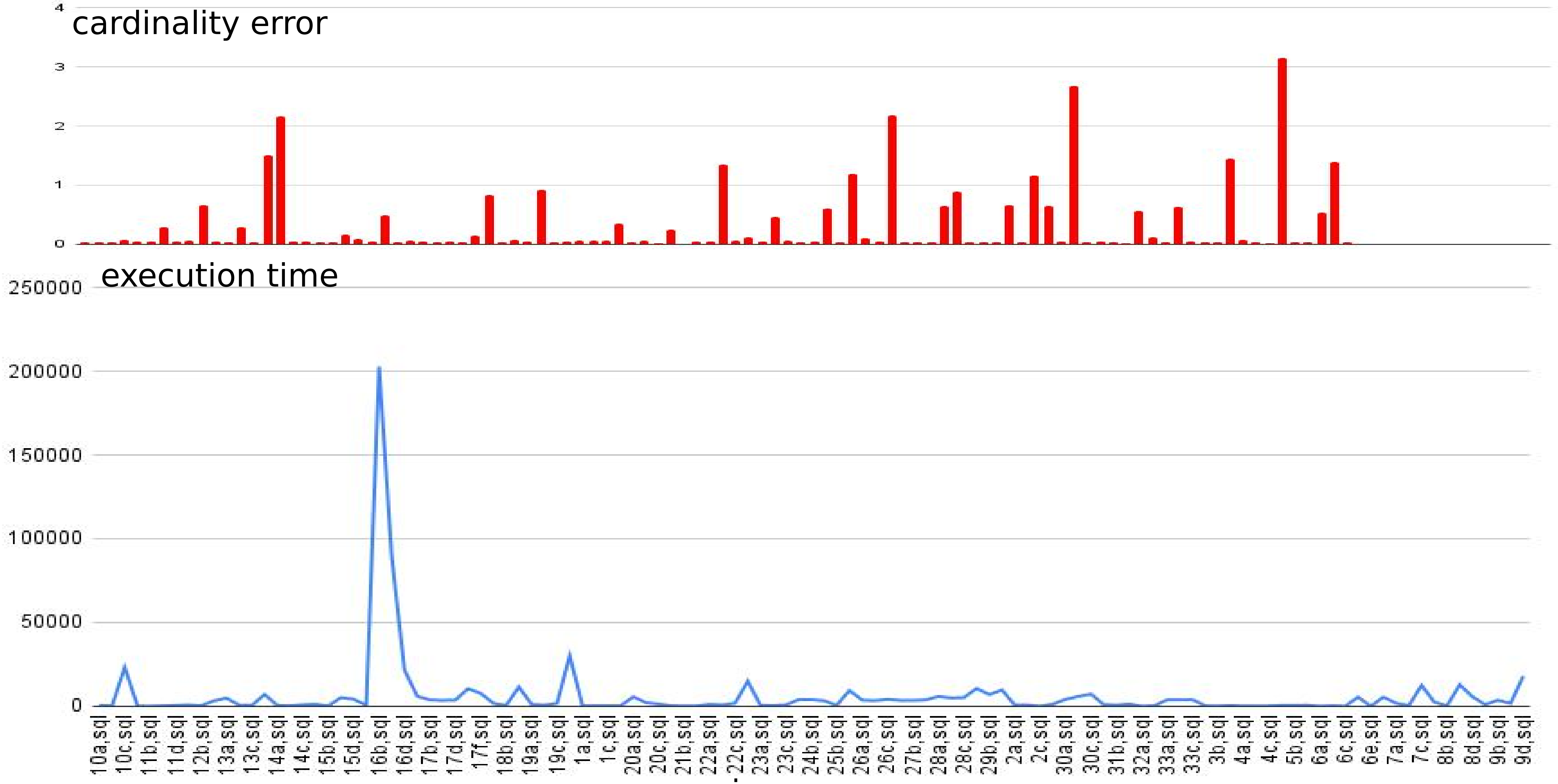
JOB Results In Learn Mode



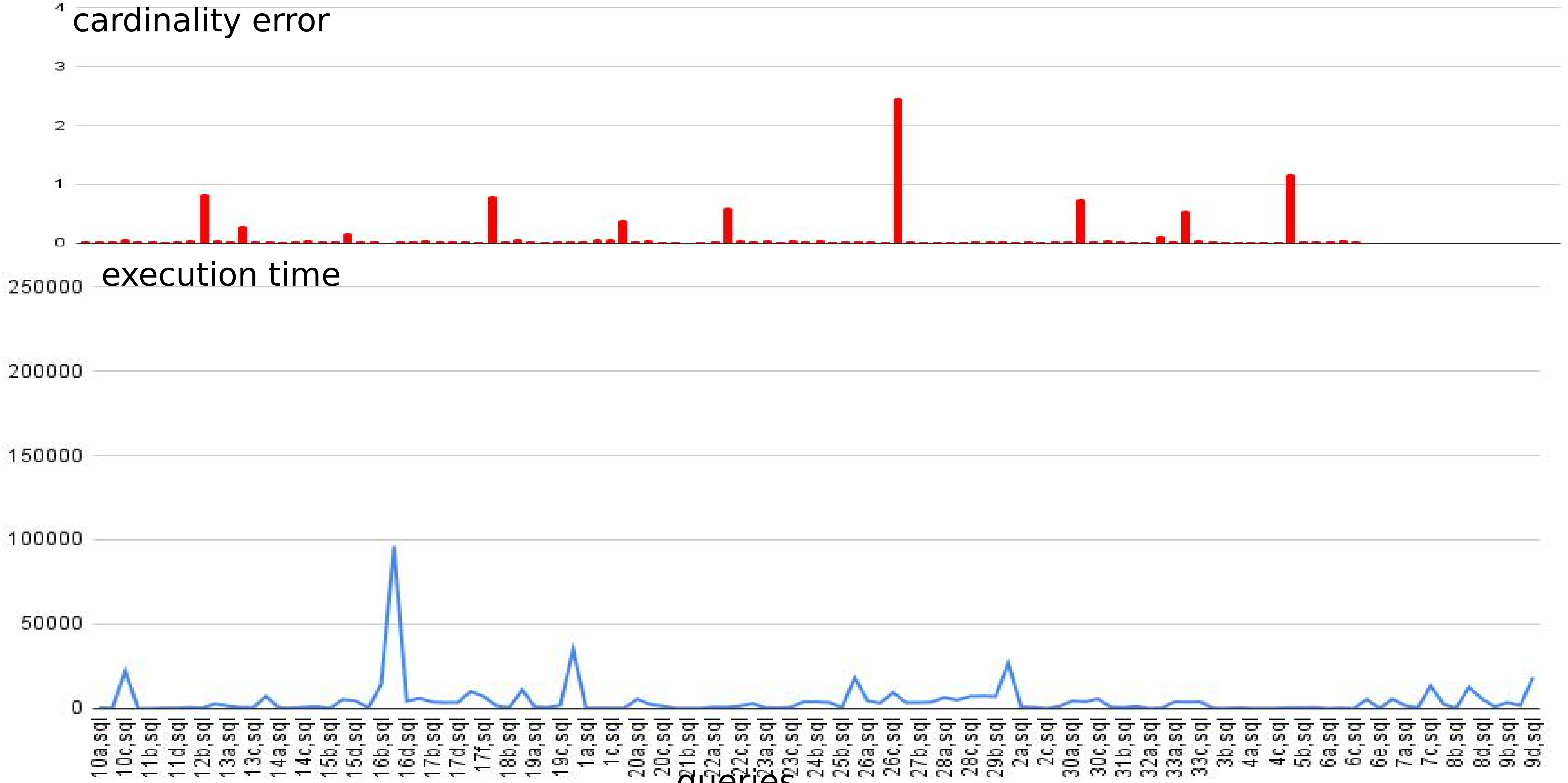
JOB Results In Learn Mode



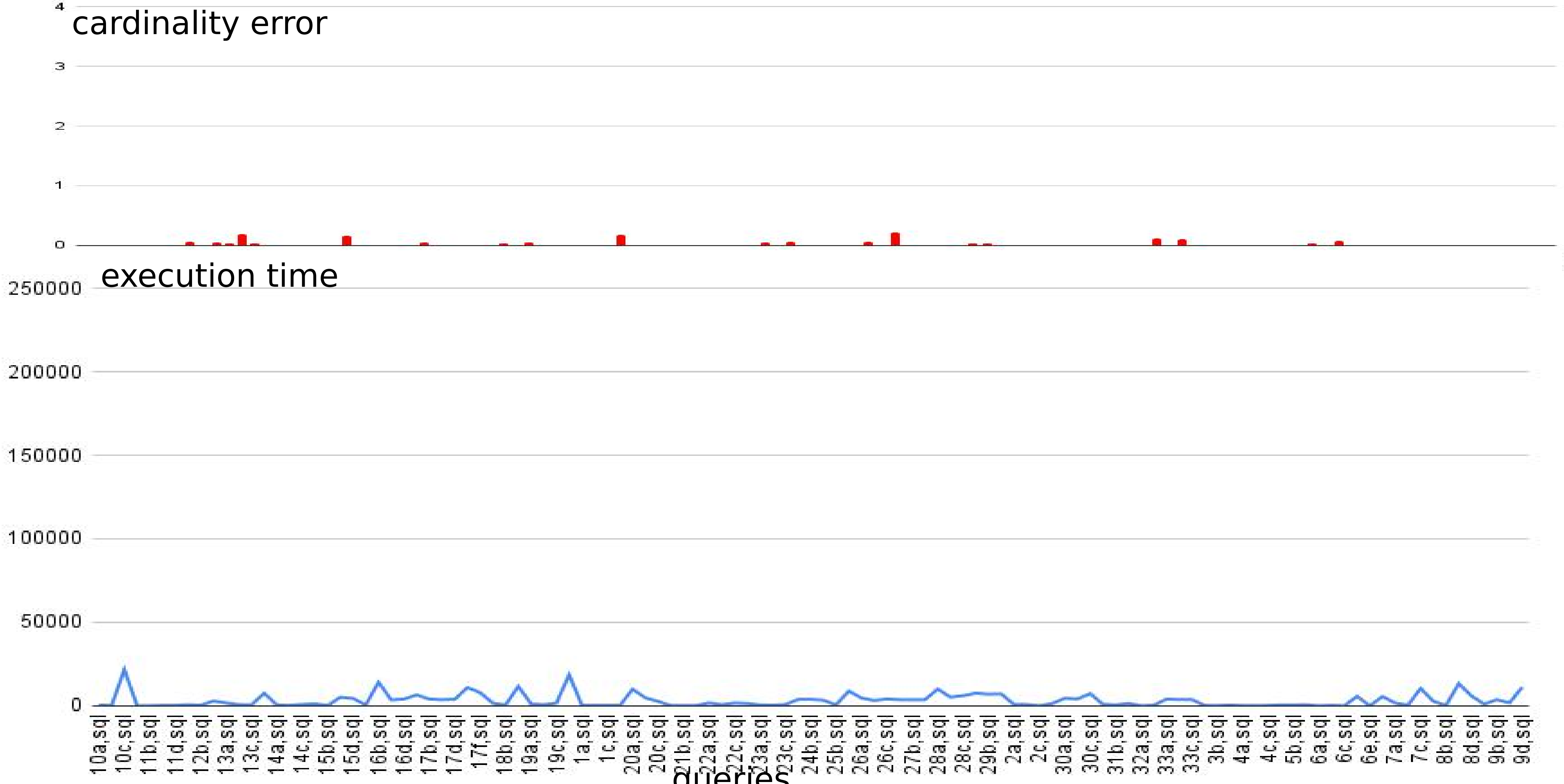
JOB Results In Learn Mode



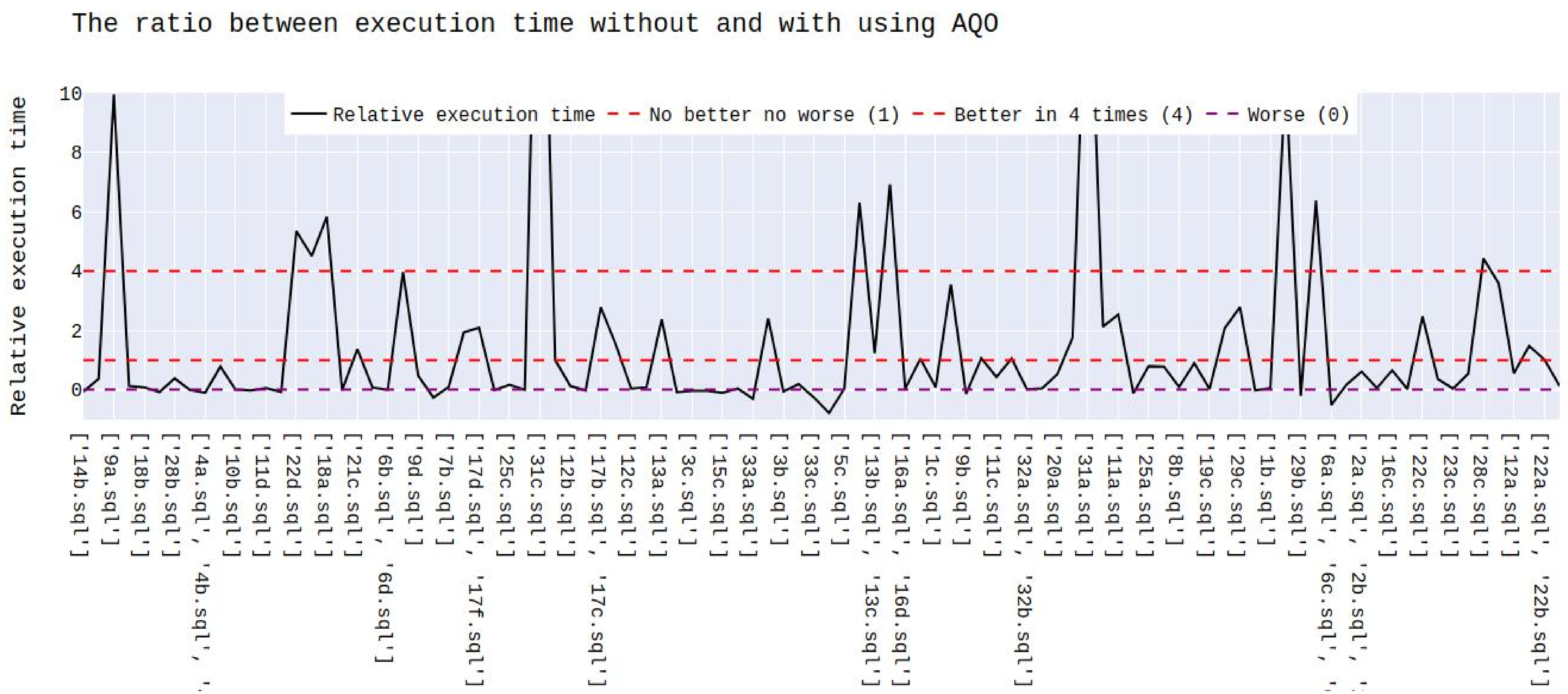
JOB Results In Learn Mode



JOB Results In Learn Mode



Improved Query Performance



4. Testing results

Conclusion

In Conclusion

PGConf.dev

AQO:

+ Stores statistical information about query execution

+ Helps optimizer to improve cardinality estimation

+ Is useful for complicated queries of the same structure with slow plan caused by bad cardinality estimates

+ Works well for OLAP–like queries

You can find the AQO extension here:
https://github.com/postgrespro/aqo

In Conclusion

AQO:

+ Stores statistical information about query execution

+ Helps optimizer to improve cardinality estimation

+ Is useful for complicated queries of the same structure with slow plan caused by bad cardinality estimates

+ Works well for OLAP–like queries

Has limitations:

- Works well when data distribution doesn't change rapidly
- Works well in databases with few temporary tables

You can find the AQO extension here:
https://github.com/postgrespro/aqo

Plans

PGConf .dev

- Learning on replica: the main question is whose knowledge we should use
- Learning on temporary tables: the main question is how to determine them after cancelation of session
- Queries with limit number of tuples (Limit node)
- One of the subnodes of the connection node does not have any rows
- Accounting the side–effect of skyrocketing of tuples because of the presence of dublicate–keyed tuples
- …

You can find the AQO extension here:
https://github.com/postgrespro/aqo

Thank You For Your Attention!

Speaker: Alena Rybakina
email: a.rybakina@postgrespro.ru
LinkedIn: https://www.linkedin.com/in/alena-rybakina