Automating Index Selection Using Constraint Programming

PGCon 2023

Lukas Fittl & Philippe Olivier

Agenda

1. Background on Index Selection
2. A Constraint Programming Model for Index Selection
3. Utilizing the Index Selection Model in Practice

Background on Index Selection

The Index Selection Problem

We want to select which indexes to create on a table, so that:

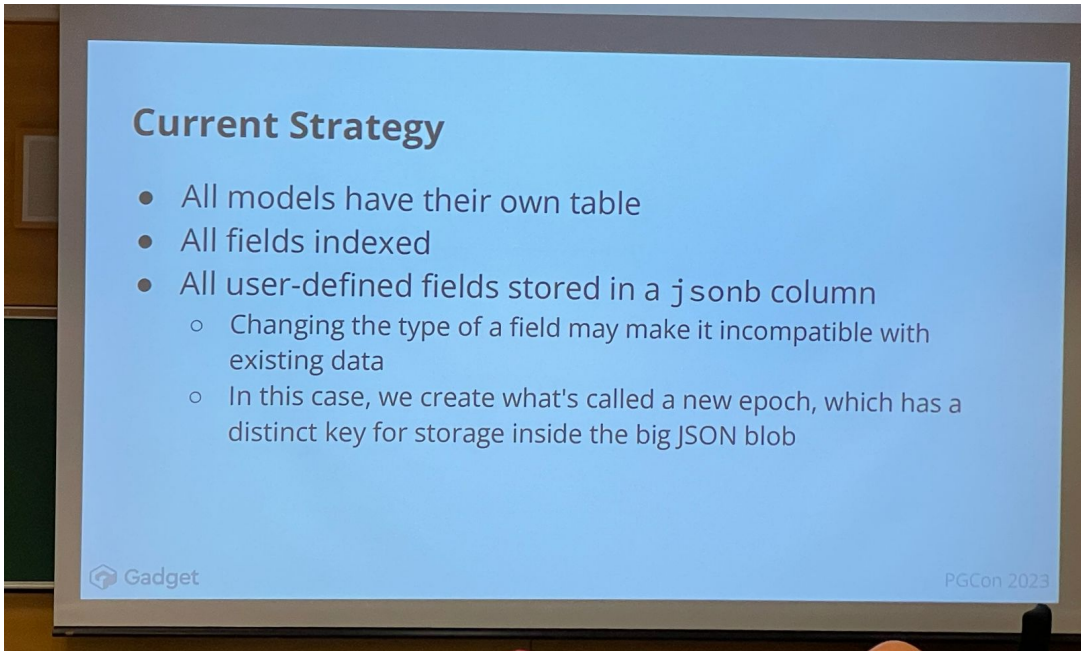
- Queries are fast
- Write overhead is kept low

Which indexes should we select?

Research Background

- An Optimization Problem on the Selection of Secondary Keys (Lum & Ling, 1971)
- Index Selection in Relational Databases (Whang, 1987)
- CoPhy: A Scalable, Portable, and Interactive Index Advisor for Large Workloads (Dash et al., 2011)
- Dexter -- The Automatic Indexer for Postgres (Kane, 2017)
- An Experimental Evaluation of Index Selection Algorithms (Kossmann et al., 2020)

“Let’s index all columns”



“Let’s pick some indexes that seem right”

_ _ _

\di index_issues*

```
 public | index_issues_on_check
 public | index_issues_on_database_id
 public | index_issues_on_database_id_and_check
 public | index_issues_on_database_id_and_severity
 public | index_issues_on_organization_id_and_check
 public | index_issues_on_reference_type_and_reference_id
 public | index_issues_on_server_id
 public | index_issues_on_server_id_and_check
 public | index_issues_on_server_id_and_check_and_grouping_key
```

Hypothetical Indexes & HypoPG

The **HypoPG extension** lets us ask “What would be the estimated cost of this query, if this index existed?”, without having to create that index.

In the simplest approach to solving index selection, we could:

- Find all columns a query filters by
- Come up with possible indexes based on the columns
- Run each possible index through HypoPG
- Select the index with the lowest cost

Hypothetical Indexes & HypoPG

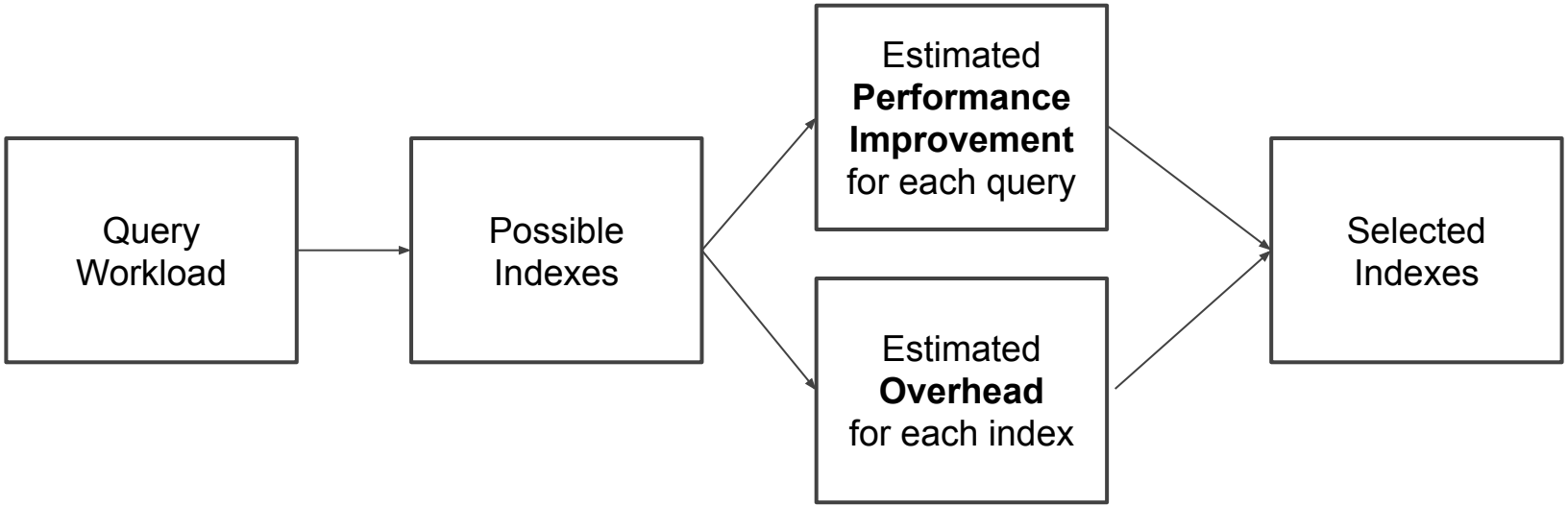
But…

How to create indexes for a **whole workload**, not just a single query?

Which multi-column indexes make sense to cover multiple queries?

How can we avoid badly slowing down writes with too many indexes?

The Index Selection Problem



Estimating Performance Improvement for each query

- Use Postgres planner **costs** to estimate performance improvement
  (they are cheap to calculate for hypothetical indexes using HypoPG)
- Make it easier to reason about complex queries,
  split them up into scans by table (**scan = Index Scan using idx** on table tbl)
- For each table, and each scan:
  - Get sequential scan cost (tiny tables don’t need indexes!)
  - Get existing index scan costs
  - Get possible index scan costs

Splitting up queries into scans


```
WITH slow_queries AS (
 SELECT qs.database_id, qs.fingerprint, qs.postgres_role_id, SUM(qs.total_time) / SUM(qs.calls) AS avg_time,
SUM(qs.shared_blks_read) / SUM(qs.calls) AS avg_blks_loaded, SUM(qs.calls) AS total_calls
   FROM query_stats_3dd qs
  WHERE qs.database_id IN (
   SELECT id FROM databases
    WHERE server_id = $4 AND NOT hidden
 ) AND qs.collected_at >= $5
  GROUP BY 1, 2, 3 HAVING SUM(qs.calls) > $6 AND SUM(qs.total_time) / SUM(qs.calls) > $7
)
SELECT q.id, (
 SELECT MAX(runtime_ms) FROM query_samples_7d qs
  WHERE qs.database_id = qfp.database_id AND qs.query_fingerprint = qfp.fingerprint AND qs.postgres_role_id =
qfp.postgres_role_id AND qs.occurred_at >= $1
) AS max_time
 FROM slow_queries JOIN query_fingerprints qfp USING (database_id, fingerprint, postgres_role_id) JOIN queries q
ON (qfp.query_id = q.id)
WHERE q.statement_types && ARRAY[$2,$3]
```

Splitting up queries into scans

public.databases	(NOT hidden) AND (server_id = $n) AND (id = $n)	Bitmap Heap Scan
	WHERE clause: (NOT hidden) AND (server_id = $n)	
	JOIN clause: (id = $n)	
public.queries	((statement_types && (ARRAY[$n])::text[]) OR (statement_types && ...	Bitmap Heap Scan
	WHERE clause: ((statement_types && (ARRAY[$n])::text[]) OR (statement_types && (ARRAY[$n])::text[]))	
	JOIN clause: (id = $n)	
public.query_samples_7d	(occurred_at >= $n) AND (database_id = $n) AND (query_fingerprint...	Append
	WHERE clause: (occurred_at >= $n) AND (database_id = $n) AND (query_fingerprint = $n) AND (postgres_role_id = $n)	
	JOIN clause: -	
public.query_stats_35d	(collected_at >= $n) AND (database_id = $n)	Seq Scan
	WHERE clause: (collected_at >= $n)	
	JOIN clause: (database_id = $n)	

Estimated Overhead for each index

How to we measure the fact that each index has a cost?

Historically, approaches have used estimated **storage size** of a given index (e.g. as calculated by HypoPG in the case of Postgres).

However, in practice, and especially in the cloud, **I/Os** are often more expensive and problematic, than storage space.

Our Approach - Index Write Overhead (IWO)

Index Write Overhead = the estimated size of an index write (in bytes), based on the index definition, divided by the size of the average table row.

table

- col1 text, avg_width = 30 bytes
- col2 bigint, avg_width = 8 bytes
- col3 uuid, avg_width = 16 bytes

avg row size = 54 bytes

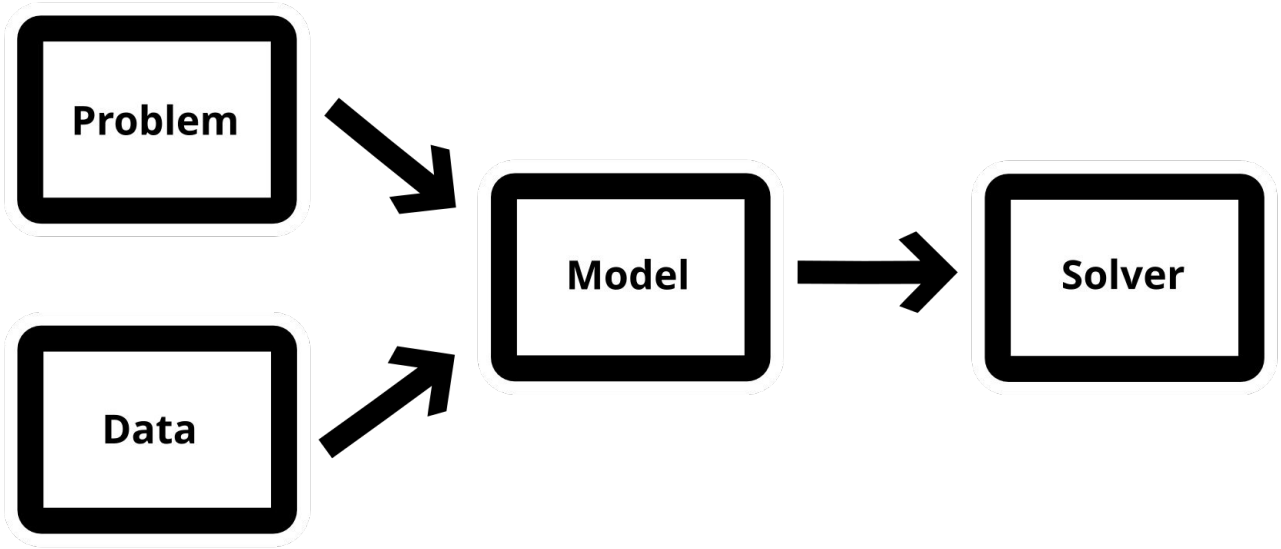
	IWO
idx1 (col2)	8/54 = 0.14
idx2 (col2, col1)	38/54 = 0.70
idx3 (col3)	16/54 = 0.29

A Constraint Programming Model For Index Selection

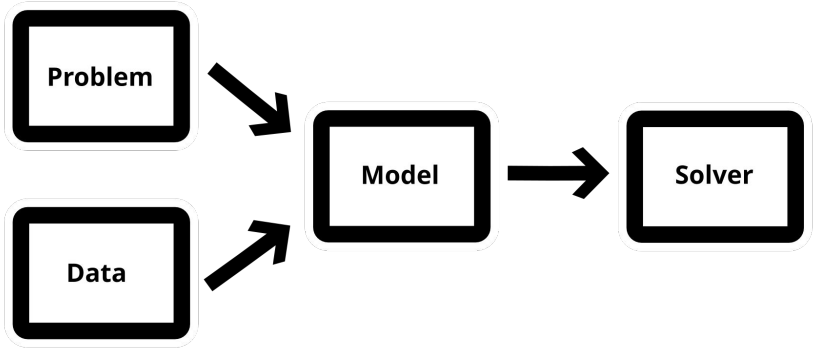
Optimization

- Find a good solution to a problem
- How?
  - Heuristics
  - Exact methods (MIP, CP, etc.)

Optimization



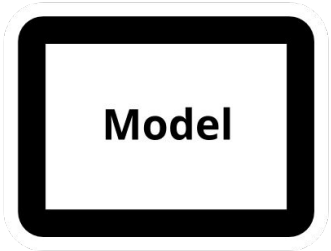
Optimization



$$\min \sum_{i \in \mathcal{I}} x_i w_i^p + \sum_{k \in \mathcal{E}} y_k w_k^e$$

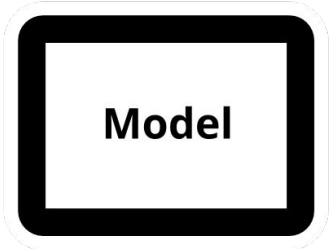
```
model.Add(model.objective ==
          cp_model.LinearExpr.WeightedSum(model.x, model.pind_iwo) +
          cp_model.LinearExpr.WeightedSum(model.y, model.eind_iwo))
model.Minimize(model.objective)
```

Declarative Model



- Variables: What we want to find
- Constraints: Rules we must follow
- Objectives: Goals we want to achieve

Declarative Model



- Variables: What we want to find ← **solution**
- Constraints: Rules we must follow
- Objectives: Goals we want to achieve

The solver finds a solution (the best?) to the model.

Index Selection Model

- Variables: Which indexes to select
- Constraints: User-defined rules
- Objectives: User-defined goals

The index selection model will find a suitable selection of indexes.

Example: *“Select the indexes that minimize the costs and the IWO.”*

Single and Multiple Goals

Single goal:

- Minimize the costs: Easy! Use more indexes
- Minimize the IWO: Easy! Use fewer indexes

Single and Multiple Goals

Single goal:

- Minimize the costs: Easy! Use more indexes
- Minimize the IWO: Easy! Use fewer indexes

Multiple goals:

- Minimize the costs and the IWO: ???

conflict

Conflicting Goals

Multi-objective methods:

- Weighted sum method
- ϵ-constraint method
- Lexicographic method
- **Hierarchical optimization method**

Conflicting Goals

Sort the goals by preference:

1. First goal: Minimize the costs
2. New rule: The costs must not be worse than X
3. Second goal: Minimize the IWO

Conflicting Goals

Sort the goals by preference:

strictness ↙

1. First goal: Minimize the costs
2. New rule: The costs must not be worse ~~than X~~ than 90% of X
3. Second goal: Minimize the IWO

Conflicting Goals

Sort the goals by preference:

strictness ↙

1. First goal: Minimize the costs
2. New rule: The costs must not be worse ~~than X~~ than 90% of X
3. Second goal: Minimize the IWO

“I want to be within 90% of whatever the lowest possible costs are. Which selection of indexes allows me to have that for as little IWO as possible?”

User-Defined Goals

- Minimize total scan cost
- Minimize IWO
- Minimize worst cost
- Minimize the number of indexes used
- Consider impact (scan cost weighted by frequency)
- Target specific scans
- And more

User-Defined Goals

- Minimize total scan cost
- Minimize IWO
- Minimize worst cost
- Minimize the number of indexes used
- Consider impact (scan cost weighted by frequency)
- Target specific scans
- And more

If you can put a number on something, it can be optimized.

User-Defined Options

pganalyze

User-defined rules:

- Limit the number of indexes selected
- Limit the total IWO/storage/etc
- Priority scans (e.g., web app queries)

Other options:

- Ignore specific scans
- Allow replacing existing indexes
  - Specific goals
  - Specific rules

Example

–

1. Minimize costs (90% strictness)

2. Minimize IWO

IWO ↓		S_1	S_2	S_3
3	I_1	4	3	
3	I_2		3	4
1	I_3	8		5
1	I_4	7	2	8

Example

–

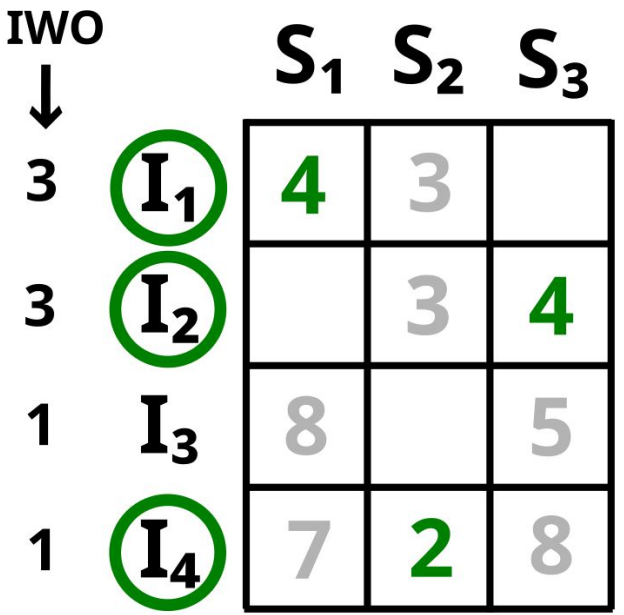
1. Minimize costs (90% strictness)

   Indexes: I_1, I_2, I_4

   Costs: 4 + 2 + 4 = 10

   IWO: 3 + 3 + 1 = 7

2. Minimize IWO



Example

–

1. Minimize costs (90% strictness)

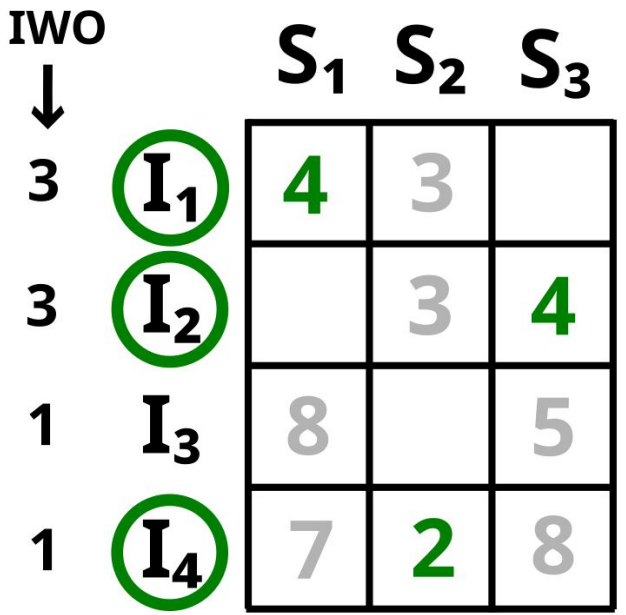
   Indexes: I_1, I_2, I_4

   Costs: 4 + 2 + 4 = 10

   IWO: 3 + 3 + 1 = 7

   90% as good as

2. Minimize IWO + rule: costs ≤ 11



Example

–

1. Minimize costs (90% strictness)

   Indexes: I_1, I_2, I_4

   Costs: 4 + 2 + 4 = 10

   IWO: 3 + 3 + 1 = 7

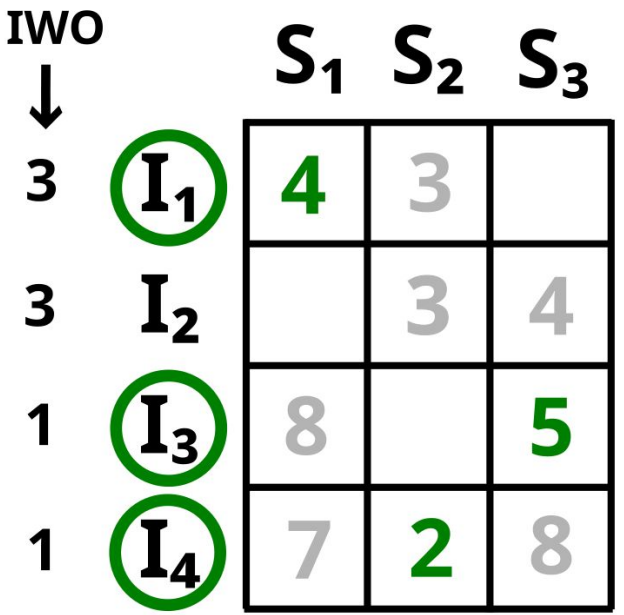
90% as good as

2. Minimize IWO + rule: costs ≤ 11

   Indexes: I_1, I_3, I_4

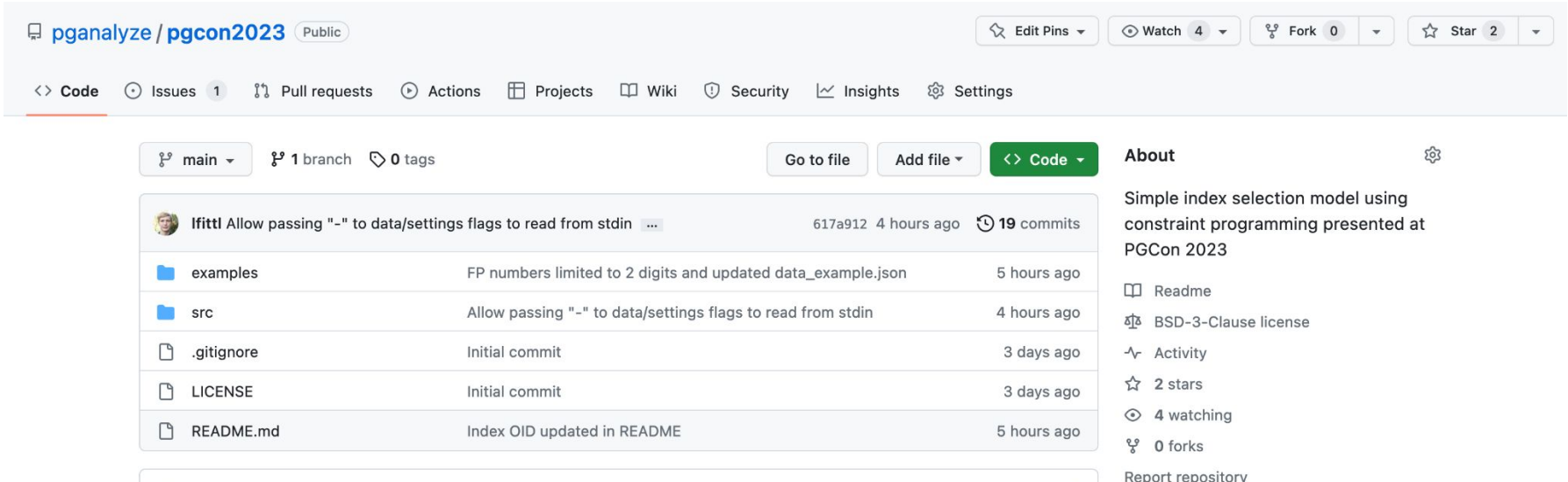
   Costs: 4 + 2 + 5 = 11

   IWO: 3 + 1 + 1 = 5



Try out the Index Selection Model

github.com/pganalyze/pgcon2023



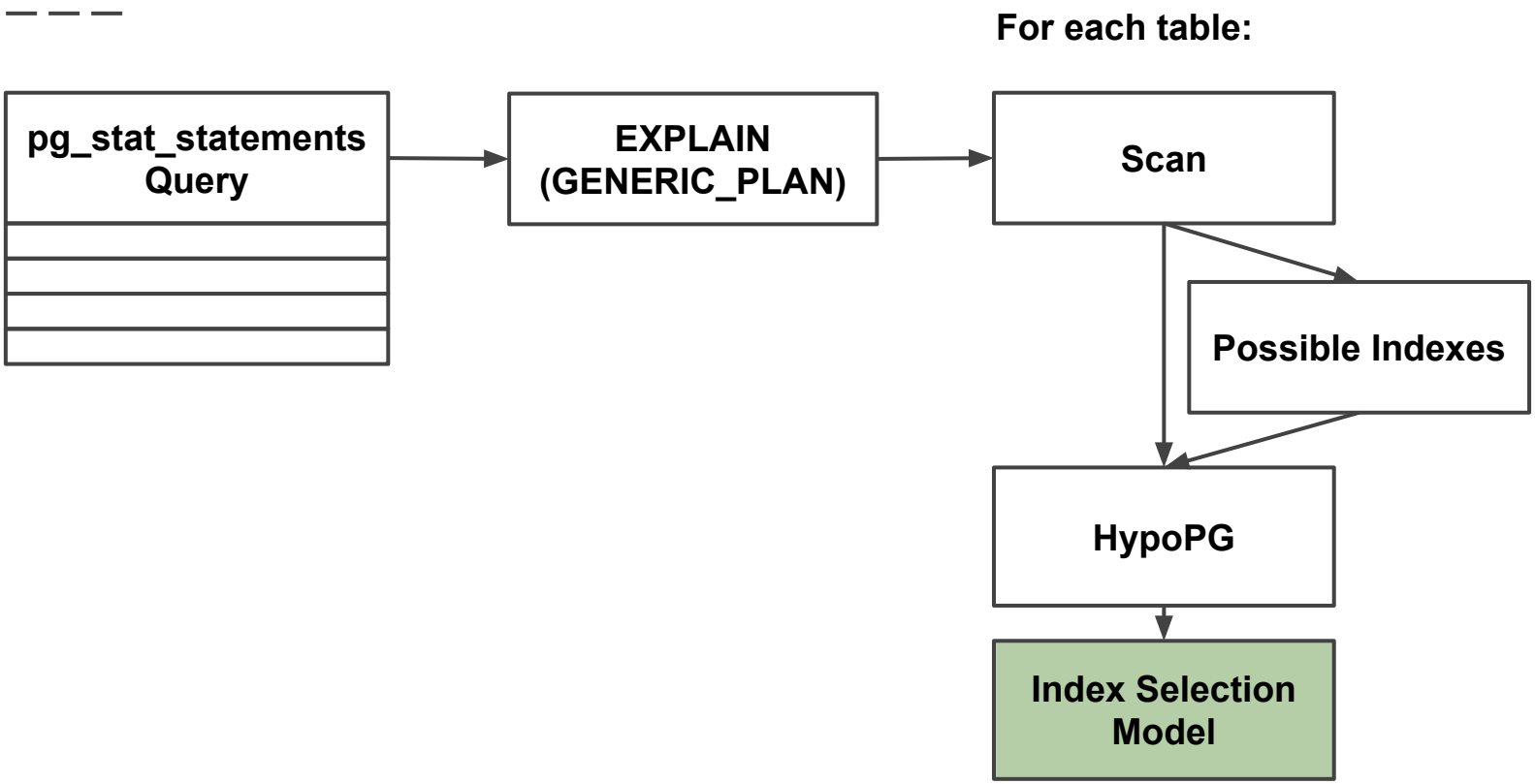
Utilizing The Index Selection Model In Practice

Demo

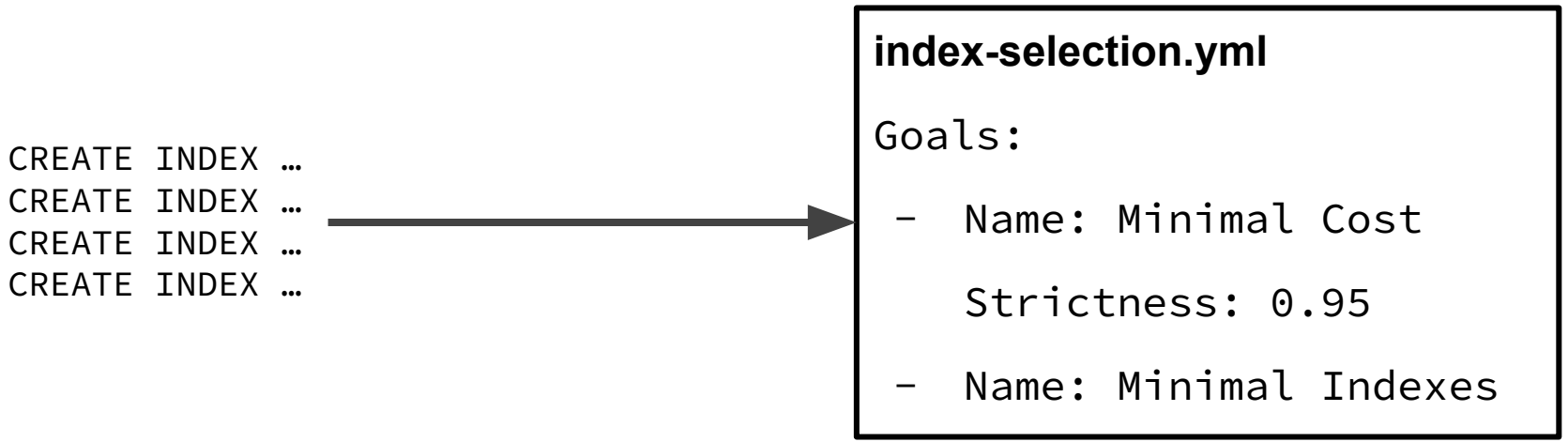
For a table:

1. Give me the missing indexes
2. Give me just one index, but the best one
3. Give me a trade-off between costs and index write overhead

How does this work?



index-selection.yml gives developer control



In Summary

- Our goal is to (semi-)automate index selection based on **application developer & data team** intent
- Provide explanations why a particular index was chosen, and **make it easy to introspect/override the logic**
- Offer a configurable system that supports choosing multiple, conflicting objectives (e.g. make queries fast, but keep overhead low)
- We've started by **checking for missing indexes first** (e.g. to catch a change early that adds new queries but forgets the index)

Possible improvements in Postgres

1. EXPLAIN: Append nodes should have an Alias field
2. Debug function to get RelOptInfo baserestrictinfo/joininfo
3. Track parameterized index scan choices
4. pg_stat_statements: Track search path + param types
5. How could we make something like pg_qualstats work better in practice? (how do we track selectivity outliers?)
6. Ability to have planner use hypothetical table sizes

thanks!

Email us to talk more about this:

team@pganalyze.com

Try out the code:

github.com/pganalyze/**pgcon2023**

github.com/pganalyze/**lint**