

# Automating Index Selection Using Constraint Programming

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
4. Advanced Use Cases  
(Per-Scan Rules, HOT Updates, Consolidation)



# 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 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...

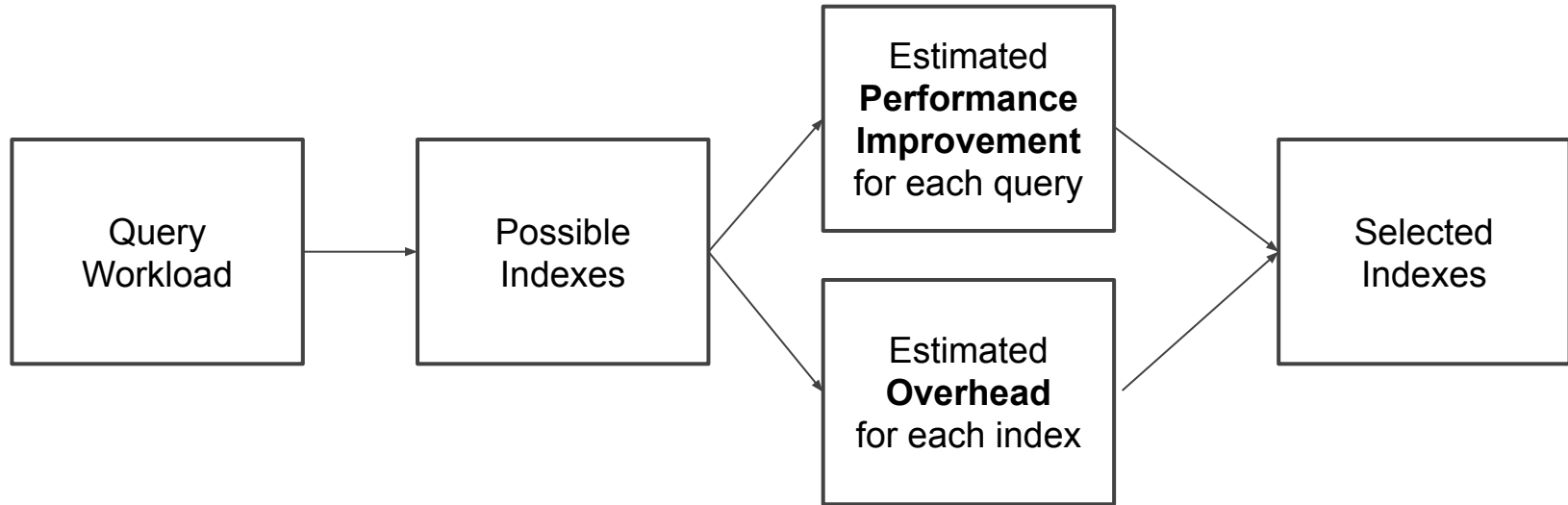
How to create indexes for a **whole workload**, not just a single query?

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

---



# Queries, Scans and Costs

---

- 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

# Queries, Scans and Costs

---

- Use Postgres planner **costs** to estimate performance improvement (they are cheap to calculate for hypothetical indexes using HypoPG)
- **“Costs are arbitrary units.** They do not represent milliseconds or any other unit of time. Instead, **they are anchored to a single read of a sequential page, which costs 1.0 unit.”**
  - [Tadeáš Peták](#), paraphrasing the Postgres documentation

# 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

	<b>IWO</b>
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

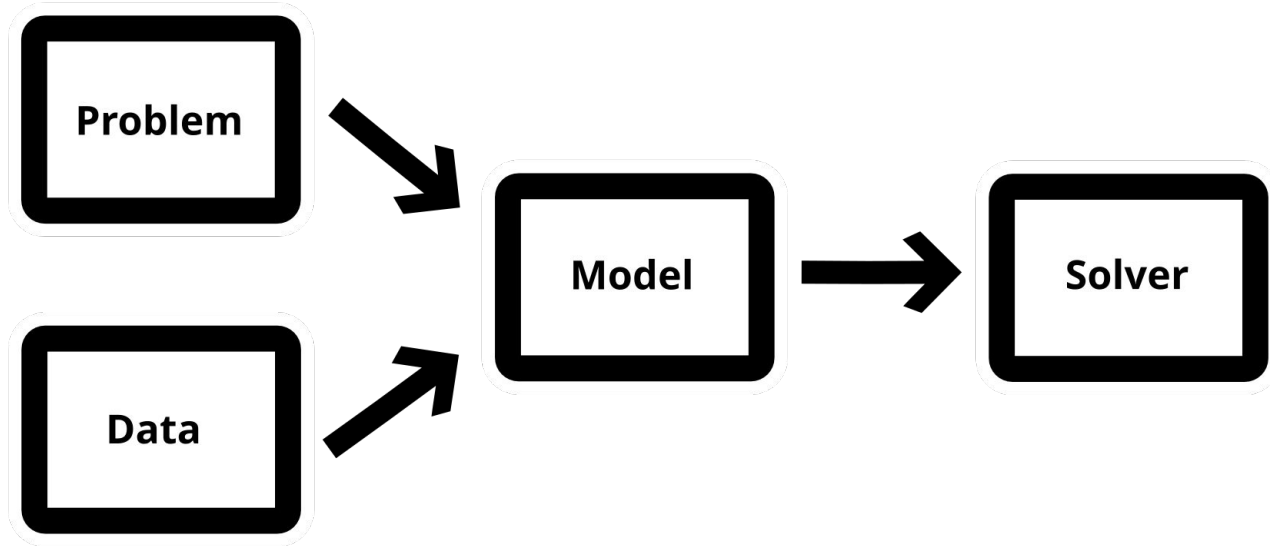
# Optimization

---

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

# Optimization

---



# Optimization

---

Make the index write overhead small



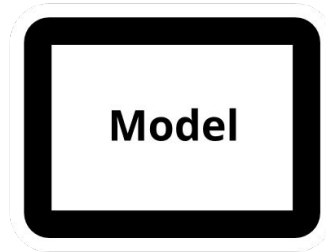
$$\min \sum_{i \in \mathcal{I}} x_i w_i^p + \sum_{k \in \mathcal{E}} y_k w_k^e$$



```
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

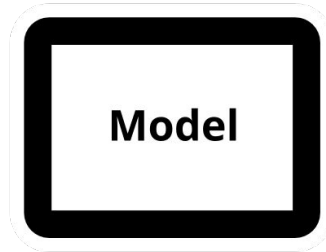
---



- 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
- $\epsilon$ -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 cannot be higher than X
3. Second goal: Minimize the IWO

# Conflicting Goals



Sort the goals by preference:

1. First goal: Minimize the costs
2. New rule: The costs cannot be higher than ~~X~~ than X + 10%
3. Second goal: Minimize the IWO

**tolerance**  
↙

# Conflicting Goals



Sort the goals by preference:

1. First goal: Minimize the costs
2. New rule: The costs cannot be higher than ~~than X~~ than X + 10%
3. Second goal: Minimize the IWO

**tolerance**  
↙

*“Costs can be 10% worse than whatever the lowest possible costs are.  
Which selection of indexes gives me that for as little IWO as possible?”*

# Example

-

1. Minimize costs (10% tolerance)

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 (10% tolerance)

Indexes:  $I_1, I_2, I_4$

Costs:  $4 + 2 + 4 = 10$

IWO:  $3 + 3 + 1 = 7$

## 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 (10% tolerance)

Indexes:  $I_1, I_2, I_4$

Costs:  $4 + 2 + 4 = 10$

IWO:  $3 + 3 + 1 = 7$

2. Minimize IWO + rule: costs  $\leq 11$

10% worse than

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 (10% tolerance)

Indexes:  $I_1, I_2, I_4$

Costs:  $4 + 2 + 4 = 10$

IWO:  $3 + 3 + 1 = 7$

## 2. Minimize IWO + rule: costs $\leq 11$

Indexes:  $I_1, I_3, I_4$

Costs:  $4 + 2 + 5 = 11$

IWO:  $3 + 1 + 1 = 5$

10% worse than

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



# Utilizing The Index Selection Model In Practice

# Per-Table Index Insights

## Current (Per-Index)

Issue #13272: Missing Index

**Overview**

Severity: Info | Check Frequency: Daily | Last Updated: 2023-11-13 06:01:58pm PST | State: Triggered [Acknowledge](#)

**Description**  
Missing index on `public.users btree (email)`

**Guidance**

```
CREATE INDEX CONCURRENTLY ON public.users USING btree (email);
```

[Copy CREATE INDEX command](#)

Cost Improvement: **150x**  
Weighted Improvement: **150x**  
Index Write Overhead: **+0.06**

We recommend testing insights in pre-production or staging environments first before deploying changes to production. If possible, it is advisable to use a copy of the production database for your tests, otherwise you may not see a representative performance improvement or query plan change.

> How to test the insight

**Scans (2)**

SCAN EXPRESSION	COST	EST. SCANS / MIN	COST IMPROVEMENT
> <code>((email)::text = \$n) AND (auth_organization_id IS NULL)</code>	1,207.70	0.0290	150x
> <code>((email)::text = \$n)</code>	1,207.70	0.5320	150x

**Affected Queries (2)**

QUERY ID	QUERY	AVG TIME (MS)	CALLS / MIN	% ALL RUNTIME
#4383...	<code>SELECT users.* FROM users WHERE users.email = \$1 LIMIT \$2</code>	3.81ms	0.5320	0.00%
#4385...	<code>SELECT users.* FROM users WHERE ... ORDER BY users.id ASC LIMIT \$2</code>	7.98ms	0.0290	0.00%

**Existing Indexes (4)**

NAME	DEFINITION	CONSTRAINT	VALID?	FIRST SEEN	SIZE
<code>index_users_unique_auth_org_id_lower_email</code>	<code>btree (COALESCE(auth_organization_id, '00000000-0-...))</code>		VALID	Over 30 days ago	3.1 MB
<code>index_users_on_auth_uid_provider_organiza...</code>	<code>btree (auth_uid, auth_provider, COALESCE(auth_org...</code>		VALID	Over 30 days ago	2 MB



## Upcoming (Per-Table)

Table: `public.users`

Statistics | Partitions | Queries | Columns | Indexes | Index Advisor | Constraints | VACUUM/ANALYZE Activity

**Summary** [Change preset](#)

Preset: `Balanced`

Scans on Table Covered <b>41/58</b> 4 scans with improved cost estimate	Total Indexes <b>4</b> 2 existing, 2 new, 1 removed	Index Write Overhead <b>+0.11</b> 0.21 total	Update Overhead <b>+0.73</b> 12.45 total
---	---	--	--

> All Scans on Table (58)

**Next Steps** [Expand all steps](#)

- Add `btree (last_sign_in_at)` index

Improves cost of 3 scans and increases Index Write Overhead by 0.14 and Update Overhead by 10.73.

```
CREATE INDEX CONCURRENTLY ON public.users USING btree (last_sign_in_at)
```

[Copy command](#)

[View covered scans details](#)
- Add `btree (email)` index

Improves cost of 1 scan and increases Index Write Overhead by 0.05 and does not affect Update Overhead.

```
CREATE INDEX CONCURRENTLY ON public.users USING btree (email)
```

[Copy command](#)

[View covered scans details](#)
- Remove `btree (login_method)` index

Decreases Index Write Overhead by 0.08 and total index size decreases by 1 GB. Index was last used 43 days ago.

```
DROP INDEX CONCURRENTLY idx_users_login_method
```

[Copy command](#)

[View covered scans details](#)

"Method": "Greedy" // Greedy set-covering algorithm used by pganalyze Index Advisor today



## Summary

Total runtime: 4.91 s, model runtime: 0.00 s

## Result

<b>Indexes Used:</b>	19 (12 existing, 7 to be added)	<b>Scan Coverage:</b>	38 covered (38 by existing, 34 by to be added), 1 uncovered
<b>Index Write Overhead:</b>	1.72 (0.98 existing, 0.74 to be added)	<b>Scan Cost:</b>	1,300,000 total, 887,600 maximum per-scan
<b>Update Overhead:</b>	Up to 101.12 non-HOT updates / min	<b>Scan Impact:</b>	33,650,000,000 total, 33,620,000,000 maximum per-scan

## Missing Indexes to Add

```
117 CREATE INDEX CONCURRENTLY ON public.issues USING btree ("check", server_id, updated_at);
129 CREATE INDEX CONCURRENTLY ON public.issues USING btree (database_id, created_at);
134 CREATE INDEX CONCURRENTLY ON public.issues USING btree (database_id, server_id, organization_id);
158 CREATE INDEX CONCURRENTLY ON public.issues USING btree (grouping_key, server_id);
181 CREATE INDEX CONCURRENTLY ON public.issues USING btree (organization_id, "check", state, updated_at);
183 CREATE INDEX CONCURRENTLY ON public.issues USING btree (organization_id, server_id);
100 CREATE INDEX CONCURRENTLY ON public.issues USING btree (server_id, severity);
```

[Copy CREATE INDEX commands](#)

We recommend testing insights in pre-production or staging environments first before deploying changes to production. If possible, it is advisable to use a copy of the production database for your tests, otherwise you may not see a representative performance improvement or query plan change.

```
"Method": "CP",
"Options": {
  "Goals": [ { "Name": "Minimize Total Cost", "Tolerance": 0 },
             { "Name": "Minimize Index Write Overhead", "Tolerance": 0 } ],
}
```



## Summary

Total runtime: 5.05 s, model runtime: 0.08 s

## Goals

1. **Minimize Total Cost** achieved minimum 1,245,000, no tolerance
2. **Minimize Index Write Overhead** achieved minimum 2.03, no tolerance

## Result

<b>Indexes Used:</b>	23 (12 existing, 11 to be added)	<b>Scan Coverage:</b>	38 covered (38 by existing, 34 by to be added), 1 uncovered
<b>Index Write Overhead:</b>	2.03 (0.98 existing, 1.05 to be added)	<b>Scan Cost:</b>	1,245,000 total, 887,600 maximum per-scan
<b>Update Overhead:</b>	Up to 101.12 non-HOT updates / min	<b>Scan Impact:</b>	33,660,000,000 total, 33,620,000,000 maximum per-scan

## Missing Indexes to Add

```
116 CREATE INDEX CONCURRENTLY ON public.issues USING btree ("check", server_id, state, updated_at);
120 CREATE INDEX CONCURRENTLY ON public.issues USING btree ("check", updated_at);
129 CREATE INDEX CONCURRENTLY ON public.issues USING btree (database_id, created_at);
132 CREATE INDEX CONCURRENTLY ON public.issues USING btree (database_id, server_id);
141 CREATE INDEX CONCURRENTLY ON public.issues USING btree (database_id, updated_at);
158 CREATE INDEX CONCURRENTLY ON public.issues USING btree (grouping_key, server_id);
181 CREATE INDEX CONCURRENTLY ON public.issues USING btree (organization_id, "check", state, updated_at);
182 CREATE INDEX CONCURRENTLY ON public.issues USING btree (organization_id, "check", updated_at);
183 CREATE INDEX CONCURRENTLY ON public.issues USING btree (organization_id, server_id);
1100 CREATE INDEX CONCURRENTLY ON public.issues USING btree (server_id, severity);
1104 CREATE INDEX CONCURRENTLY ON public.issues USING btree (server_id, updated_at);
```

[Copy CREATE INDEX commands](#)

```
"Method": "CP",
"Options": {
  "Goals": [ { "Name": "Minimize Total Cost", "Tolerance": 0.10 },
             { "Name": "Minimize Index Write Overhead", "Tolerance": 0 } ],
}
```



## Summary

Total runtime: 5.01 s, model runtime: 0.11 s

### Goals

1. **Minimize Total Cost** achieved minimum 1,245,000, tolerance 0.1 allows up to 1,369,500
2. **Minimize Index Write Overhead** achieved minimum 0.98, no tolerance

### Result

<b>Indexes Used:</b>	12 (12 existing, 0 to be added)	<b>Scan Coverage:</b>	38 covered (38 by existing, 0 by to be added), 1 uncovered
<b>Index Write Overhead:</b>	0.98 (0.98 existing, 0.00 to be added)	<b>Scan Cost:</b>	1,329,000 total, 887,600 maximum per-scan
<b>Update Overhead:</b>	Up to 101.12 non-HOT updates / min	<b>Scan Impact:</b>	33,720,000,000 total, 33,620,000,000 maximum per-scan

No changes recommended

With the current configuration, the Indexing Engine did not find any missing indexes to recommend. Try changing the index selection configuration settings for different trade-offs.

[Learn more in documentation](#)

# Total Cost: 1,245,000 vs 1,329,000



# Advanced Use Cases

# Per-Scan Optimization

---

129 btree (database\_id, created\_at)

+0.05 Index Write Overhead ⓘ

SCAN EXPRESSION

> S32 (created\_at >= \$n) AND (created\_at <= \$n) AND (database\_id = \$n)

EST. COST

EST. NEW COST

EST. SCANS/MIN

1,776.06

12.33

0.06

**But what if we care about individual scans?**

**Total Cost: 1,245,000 vs 1,329,000**

```
"Method": "CP",
"Options": {
  "Goals": [ { "Name": "Minimize Total Cost", "Tolerance": 0.10 },
             { "Name": "Minimize Index Write Overhead", "Tolerance": 0 } ],
  "Rules": { "Maximum Per-Scan Cost (Normal)": 100 }
}
```



## Summary

Total runtime: 5.22 s, model runtime: 0.08 s

## Goals

1. **Minimize Total Cost** achieved minimum 1,245,000, tolerance 0.1 allows up to 1,369,500
2. **Minimize Index Write Overhead** achieved minimum 1.29, no tolerance

## Result

<b>Indexes Used:</b>	16 (12 existing, 4 to be added)	<b>Scan Coverage:</b>	38 covered (38 by existing, 33 by to be added), 1 uncovered
<b>Index Write Overhead:</b>	1.29 (0.98 existing, 0.31 to be added)	<b>Scan Cost:</b>	1,303,000 total, 887,600 maximum per-scan
<b>Update Overhead:</b>	Up to 101.12 non-HOT updates / min	<b>Scan Impact:</b>	33,680,000,000 total, 33,580,000,000 maximum per-scan

## Missing Indexes to Add

```
114 CREATE INDEX CONCURRENTLY ON public.issues USING btree ("check", server_id);
129 CREATE INDEX CONCURRENTLY ON public.issues USING btree (database_id, created_at);
132 CREATE INDEX CONCURRENTLY ON public.issues USING btree (database_id, server_id);
183 CREATE INDEX CONCURRENTLY ON public.issues USING btree (organization_id, server_id);
```

[Copy CREATE INDEX commands](#)

Now our scan is indexed better.

We recommend testing insights in pre-production or staging environments first before deploying changes to production. If possible, it is advisable to use a copy of the production database for your tests, otherwise you may not see a representative performance improvement or query plan change.

# HOT Updates

---

- HOT = Heap Only Tuples
- Reduces individual UPDATE overhead by not updating index entries
- Reduces future autovacuum effort by enabling on-access HOT pruning (which can happen on a per-page basis, only for Heap Only Tuples)
- **If you index a previously unindexed column\*, any UPDATE statement involving that column could be impacted\*\***

\* Postgres 16+ allows BRIN indexes to not interfere with HOT updates

\*\* Updates that don't actually change the column value may still use HOT

# HOT Updates



Columns						
Name	Type	Null % ⓘ	Avg. Size ⓘ	Est. Updates / Min ⓘ	HOT? ⓘ	Modifiers
id	integer	0.00%	4	n/a	No ⓘ	NOT NULL DEFAULT nextval('public.issues_id_seq'::regclass)
database_id	bigint	2.93%	8	n/a	No ⓘ	
check	character varying(255)	0.00%	29	n/a	No ⓘ	NOT NULL
description_template	text	0.00%	81	21.95	Yes ✓	
reference_id	text	96.70%	11	n/a	No ⓘ	
reference_type	character varying(255)	96.70%	8	n/a	No ⓘ	
created_at	timestamp without time zone	0.00%	8	n/a	Yes ✓	NOT NULL
updated_at	timestamp without time zone	0.00%	8	93.63	No ⚠	NOT NULL
severity	integer	0.00%	4	0.08	No ⚠	NOT NULL
state	integer	0.00%	4	9.00	No ⚠	NOT NULL
server_id	uuid	0.00%	16	n/a	No ⓘ	NOT NULL
organization_id	uuid	0.00%	16	n/a	No ⓘ	NOT NULL
details	jsonb	0.00%	91	63.27	Yes ✓	NOT NULL DEFAULT '{}'::jsonb
grouping_key	jsonb	0.00%	74	n/a	No ⓘ	NOT NULL DEFAULT '{}'::jsonb
grouping_key_labels	jsonb	0.00%	49	<0.01	Yes ✓	NOT NULL DEFAULT '{}'::jsonb
query_text	text	55.10%	92	n/a	Yes ✓	

```
"Method": "CP",
"Options": {
  "Goals": [ { "Name": "Minimize Total Cost", "Tolerance": 0.10 },
             { "Name": "Minimize Index Write Overhead", "Tolerance": 0 } ],
  "Rules": { "Maximum Per-Scan Cost (Normal)": 100 }
}
```



## Summary

Total runtime: 1.74 s, model runtime: 0.07 s

### Goals

1. **Minimize Total Cost** achieved minimum 59,280, tolerance 0.1 allows up to 65,208
2. **Minimize Index Write Overhead** achieved minimum 0.30, no tolerance

### Result

<b>Indexes Used:</b>	4 (3 existing, 1 to be added)	<b>Scan Coverage:</b>	28 covered (27 by existing, 1 by to be added), 4 uncovered
<b>Index Write Overhead:</b>	0.30 (0.26 existing, 0.04 to be added)	<b>Scan Cost:</b>	61,700 total, 15,450 maximum per-scan
<b>Update Overhead:</b>	Up to 283.16 non-HOT updates / min	<b>Scan Impact:</b>	1,100,000 total, 329,300 maximum per-scan

### Missing Indexes to Add

```
115 CREATE INDEX CONCURRENTLY ON public.servers USING btree (last_snapshot_id);
```

[Copy CREATE INDEX commands](#)

We recommend testing insights in pre-production or staging environments first before deploying changes to production. If possible, it is advisable to use a copy of the production database for your tests, otherwise you may not see a representative performance improvement or query plan change.

```
"Method": "CP",
"Options": {
  "Goals": [ { "Name": "Minimize Update Overhead", "Tolerance": 0 },
             { "Name": "Minimize Total Cost", "Tolerance": 0.10 } ],
             { "Name": "Minimize Index Write Overhead", "Tolerance": 0 } ]
}
```



## Summary

Total runtime: 1.36 s, model runtime: 0.04 s

### Goals

1. **Minimize Update Overhead** achieved minimum 0.00, no tolerance
2. **Minimize Total Cost** achieved minimum 77,220, tolerance 0.1 allows up to 84,942
3. **Minimize Index Write Overhead** achieved minimum 0.26, no tolerance

### Result

<b>Indexes Used:</b>	3 (3 existing, 0 to be added)	<b>Scan Coverage:</b>	27 covered (27 by existing, 0 by to be added), 5 uncovered
<b>Index Write Overhead:</b>	0.26 (0.26 existing, 0.00 to be added)	<b>Scan Cost:</b>	77,230 total, 15,540 maximum per-scan
<b>Update Overhead:</b>	Up to 0.00 non-HOT updates / min	<b>Scan Impact:</b>	1,100,000 total, 328,900 maximum per-scan

No changes recommended

With the current configuration, the Indexing Engine did not find any missing indexes to recommend. Try changing the index selection configuration settings for different trade-offs.

[Learn more in documentation](#)

# Consolidating Indexes

---

- What if we've over indexed in the past, and want help reducing indexes?
- Optionally (and very much still experimental),  
**pganalyze can suggest removing existing indexes**
- This may cause new indexes to be created to consolidate existing ones
- pganalyze automatically excludes primary keys and unique constraints
- Depending on the situation, removing indexes may cause significant slowdowns. **Caution and testing on database clones is advised!**

# We have 12 indexes. Can we go to $\leq 5$ ?

— — —

Indexes						
Total Index Size	2.9 GB		Index Write Overhead ⓘ	0.59		
Name	Definition	Constraint	Valid?	First Seen	Index Size	Index Write Overhead
<a href="#">index_issues_on_check</a>	btree ("check")		VALID	Over 30 days ago	164.8 MB	0.08
<a href="#">index_issues_on_database_id</a>	btree (database_id)		VALID	Over 30 days ago	158 MB	0.03
<a href="#">index_issues_on_database_id_and_check</a>	btree (database_id, "check")		VALID	Over 30 days ago	169.6 MB	0.10
<a href="#">index_issues_on_database_id_and_severity</a>	btree (database_id, severity) <b>WHERE</b> (state <> 2)		VALID	Over 30 days ago	12.8 MB	0.00
<a href="#">index_issues_on_organization_id_and_check</a>	btree (organization_id, "check")		VALID	Over 30 days ago	173.9 MB	0.11
<a href="#">index_issues_on_reference_type_and_reference_id</a>	btree (reference_type, reference_id)		VALID	Over 30 days ago	590.8 MB	0.03
<a href="#">index_issues_on_server_id</a>	btree (server_id)		VALID	Over 30 days ago	166 MB	0.05
<a href="#">index_issues_on_server_id_and_check</a>	btree (server_id, "check")		VALID	Over 30 days ago	174.5 MB	0.11
<a href="#">index_issues_on_server_id_and_check_and_grouping_key</a>	btree (server_id, "check", grouping_key) <b>WHERE</b> (state <> 2)		VALID	Over 30 days ago	168.1 MB	0.03
<a href="#">index_issues_on_server_id_and_severity</a>	btree (server_id, severity) <b>WHERE</b> (state <> 2)		VALID	Over 30 days ago	12.2 MB	0.01
<a href="#">index_issues_on_server_id_and_updated_at</a>	btree (server_id, updated_at) <b>WHERE</b> (state = 2)		VALID	Over 30 days ago	685.8 MB	0.01
<a href="#">issues_pkey</a>	btree (id)	<b>PRIMARY KEY</b> (id)	VALID	Over 30 days ago	484.5 MB	0.03

```

"Method": "CP",
"Options": {
  "Goals": [ { "Name": "Minimize Total Cost", "Tolerance": 0 },
             { "Name": "Minimize Index Write Overhead", "Tolerance": 0 } ],
  "Rules": { "Maximum Number of Indexes": 5 }
}

```



## [x] Allow consolidation/removal of indexes

### Summary

Total runtime: 5.44 s, model runtime: 0.45 s

### Goals

1. **Minimize Total Cost** achieved minimum 1,427,000, no tolerance
2. **Minimize Index Write Overhead** achieved minimum 0.59, no tolerance

### Result

<b>Indexes Used:</b>	5 (2 existing, 3 to be added)	<b>Scan Coverage:</b>	38 covered (6 by existing, 35 by to be added), 1 uncovered
<b>Index Write Overhead:</b>	0.59 (0.27 existing, 0.32 to be added)	<b>Scan Cost:</b>	1,427,000 total, 887,600 maximum per-scan
<b>Update Overhead:</b>	Up to 11.61 non-HOT updates / min	<b>Scan Impact:</b>	33,470,000,000 total, 33,370,000,000 maximum per-scan

### Missing Indexes to Add

```

133 CREATE INDEX CONCURRENTLY ON public.issues USING btree (database_id, server_id, "check");
191 CREATE INDEX CONCURRENTLY ON public.issues USING btree (reference_id, reference_type);
198 CREATE INDEX CONCURRENTLY ON public.issues USING btree (server_id, organization_id, "check");

```

[Copy CREATE INDEX commands](#)

### Existing Indexes to Remove

```

12 DROP INDEX CONCURRENTLY index_issues_on_check; --- btree ("check")
13 DROP INDEX CONCURRENTLY index_issues_on_database_id; --- btree (database_id)
14 DROP INDEX CONCURRENTLY index_issues_on_database_id_and_check; --- btree (database_id, "check")
15 DROP INDEX CONCURRENTLY index_issues_on_database_id_and_severity; --- btree (database_id, severity) WHERE (state <> 2)
16 DROP INDEX CONCURRENTLY index_issues_on_organization_id_and_check; --- btree (organization_id, "check")
17 DROP INDEX CONCURRENTLY index_issues_on_reference_type_and_reference_id; --- btree (reference_type, reference_id)
18 DROP INDEX CONCURRENTLY index_issues_on_server_id; --- btree (server_id)

```

```
"Method": "CP",
"Options": {
  "Goals": [ { "Name": "Minimize Total Cost", "Tolerance": 0 },
             { "Name": "Minimize Index Write Overhead", "Tolerance": 0 } ],
  "Rules": { "Maximum Number of Indexes": 5 }
}
```

Allow consolidation/removal of indexes



## Summary

Total runtime: 5.44 s, model runtime: 0.45 s

### Goals

1. **Minimize Total Cost** achieved minimum 1,427,000, no tolerance
2. **Minimize Index Write Overhead** achieved minimum 0.59, no tolerance

### Result

**Indexes Used:** 5 (2 existing, 3 to be added)

**Index Write Overhead:** 0.59 (0.27 existing, 0.32 to be added)

**Update Overhead:** Up to 11.61 non-HOT updates / min

**Scan Coverage:** 38 covered (6 by existing, 35 by to be added), 1 uncovered

**Scan Cost:** 1,427,000 total, 887,600 maximum per-scan

**Scan Impact:** 33,470,000,000 total, 33,370,000,000 maximum per-scan

### Result

**Indexes Used:** 16 (12 existing, 4 to be added)

**Index Write Overhead:** 1.29 (0.98 existing, 0.31 to be added)

**Update Overhead:** Up to 101.12 non-HOT updates / min

**Scan Coverage:** 38 covered (38 by existing, 33 by to be added), 1 uncovered

**Scan Cost:** 1,303,000 total, 887,600 maximum per-scan

**Scan Impact:** 33,680,000,000 total, 33,580,000,000 maximum per-scan

# The trouble with functional dependencies

## Summary

Total runtime: 5.44 s, model runtime: 0.45 s

## Goals

1. **Minimize Total Cost** achieved minimum 1,427,000, no tolerance
2. **Minimize Index Write Overhead** achieved minimum 0.59, no tolerance

## Result

<b>Indexes Used:</b>	5 (2 existing, 3 to be added)	<b>Scan Coverage:</b>	38 covered (6 by existing, 35 by to be added), 1 uncovered
<b>Index Write Overhead:</b>	0.59 (0.27 existing, 0.32 to be added)	<b>Scan Cost:</b>	1,427,000 total, 887,600 maximum per-scan
<b>Update Overhead:</b>	Up to 11.61 non-HOT updates / min	<b>Scan Impact:</b>	33,470,000,000 total, 33,370,000,000 maximum per-scan

## Missing Indexes to Add

```
133 CREATE INDEX CONCURRENTLY ON public.issues USING btree (database_id, server_id, "check");
191 CREATE INDEX CONCURRENTLY ON public.issues USING btree (reference_id, reference_type);
198 CREATE INDEX CONCURRENTLY ON public.issues USING btree (server_id, organization_id, "check");
```

[Copy CREATE INDEX commands](#)

Columns with functional dependencies cause incorrect Postgres planner estimates.

**CREATE STATISTICS helps!** (and we're working to incorporate that data)

# Future work

---

- Partial indexes for specific parameter values
- More index types (BRIN, Hash)
- Incorporate test data from a database clone (to measure runtime)
- Optionally create GitHub pull requests with database migrations
- Optionally automate index creation on production

thanks!

Email us to talk more about this:

**team@pganalyze.com**

---

**Participate in early access**  
by registering in the post-webinar survey  
(or by emailing us)

