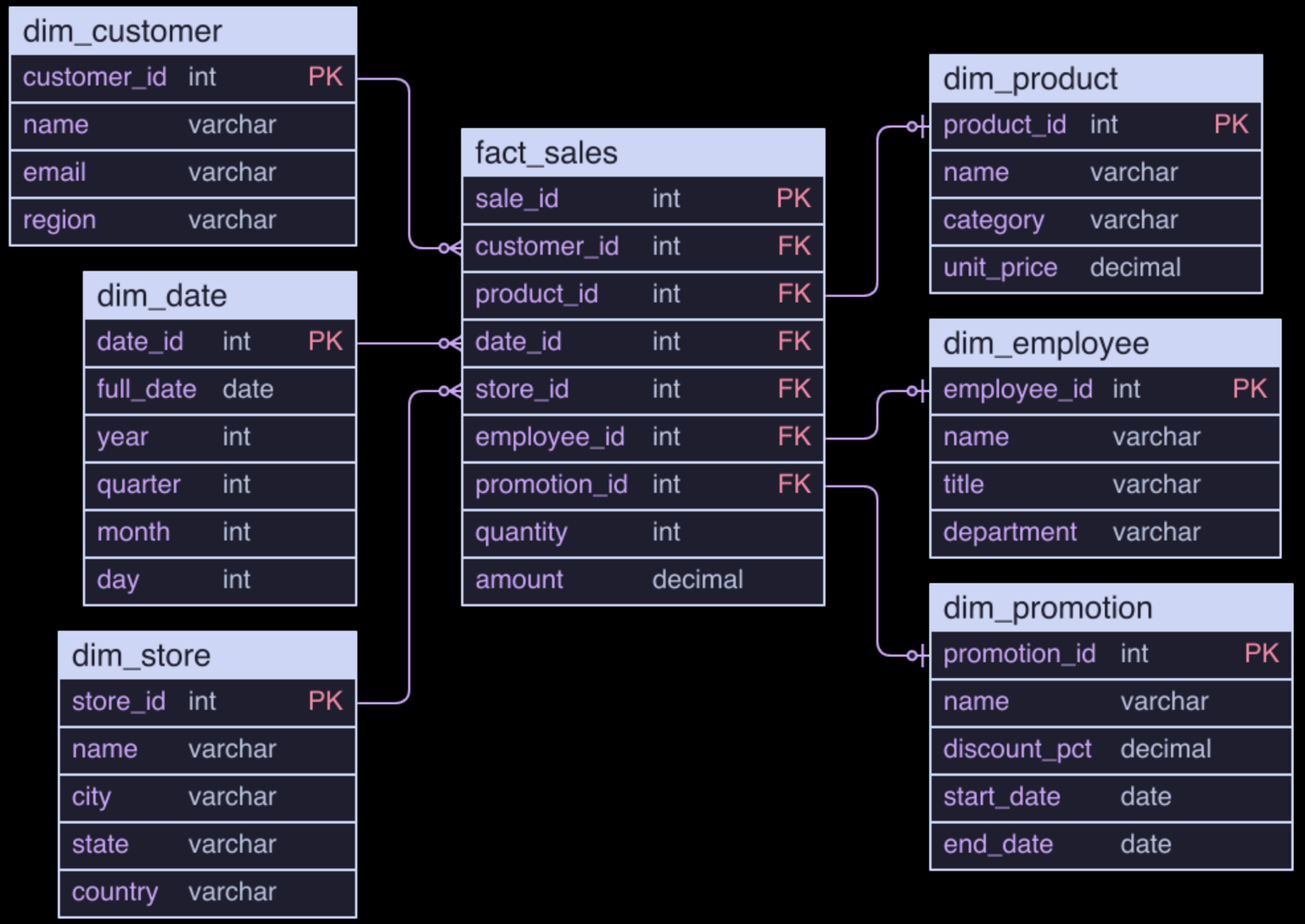
Join Statistics

Corey Huinker
pgconf.dev 2026

A Familiar Problem - Star Join Optimization



The Query Plan

Looking at `EXPLAIN ANALYZE`

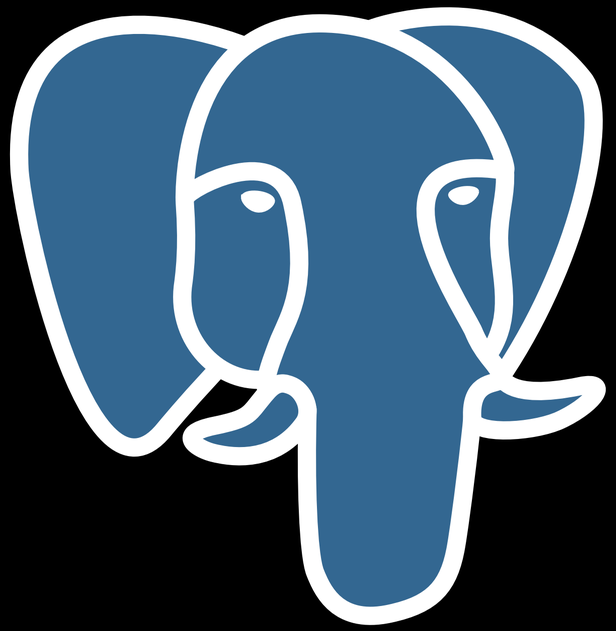
```
Nested Loop Left Join
    (cost=1.57..6750780.94 rows=192773866 width=5154)
    (actual time=1012.018..1012.057 rows=4 loops=1)
```

That estimate is a bit off…

…by 7.68 orders of magnitude

How Big Is That Overestimate?

Base Unit of Measure



Distance: 🌍 → 🌒

Diameter: 🪐

Weight (smaller): 📎

The Execution

Is quick

```
(actual time=1012.018..1012.057 rows=4 loops=1)

 1012.057 - 1012.018 = 0.035ms
```

but the startup is heavy

```
Planning Time: 0.602 ms
JIT:
  Functions: 38
  Options: Inlining true, Optimization true,
           Expressions true, Deforming true
  Timing: Generation 3.415 ms (Deform 2.285 ms),
          Inlining 12.590 ms, Optimization 519.319 ms,
          Emission 479.985 ms, Total 1015.310 ms
Execution Time: 1015.665 ms
```

1015.665ms for 4 rows ☹

Execution Time without JIT

```
Nested Loop Left Join
    (cost=1.57..6750780.94 rows=192773866 width=5154)
    (actual time=0.078..0.113 rows=4 loops=1)
...
Execution Time: 0.602 ms
```

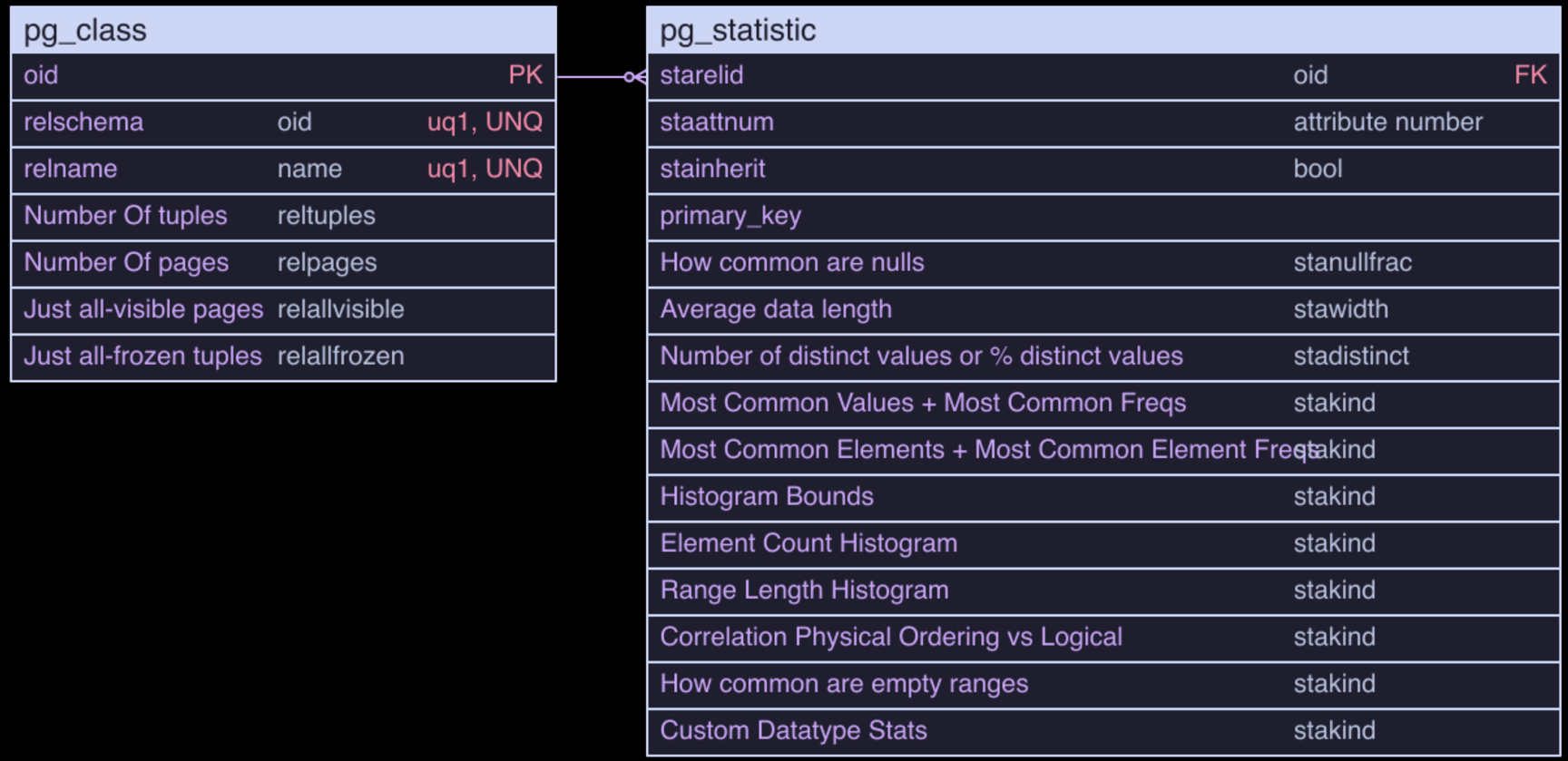
How much slower?

```
1015.665ms / 0.602 ms ~= 5000x
```

Is JIT bad? No, the row estimate is bad.

Why is the estimate bad?

Regular Statistics are Limited



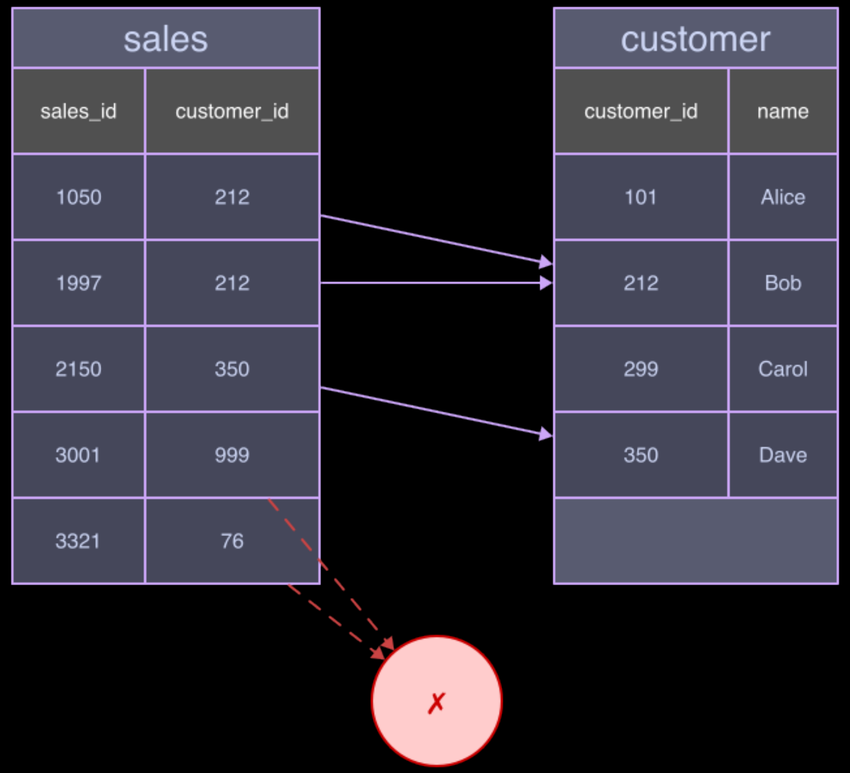
Extended Statistics

Many combinations

Allows expressions

Can only reference one table

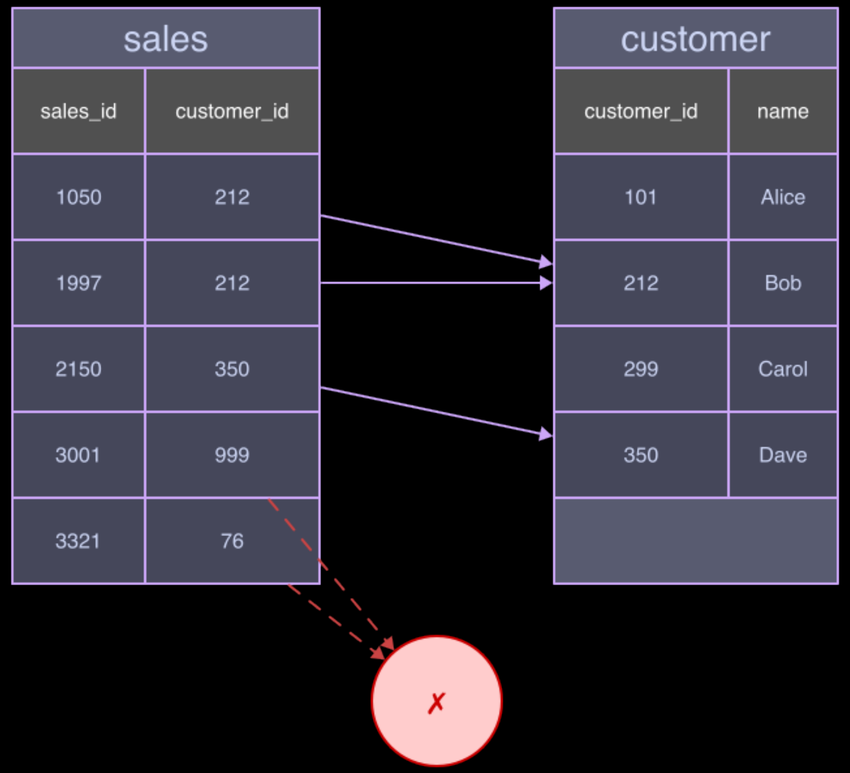
Transitive Statistics?



Applying stats from one table filtered through stats of another

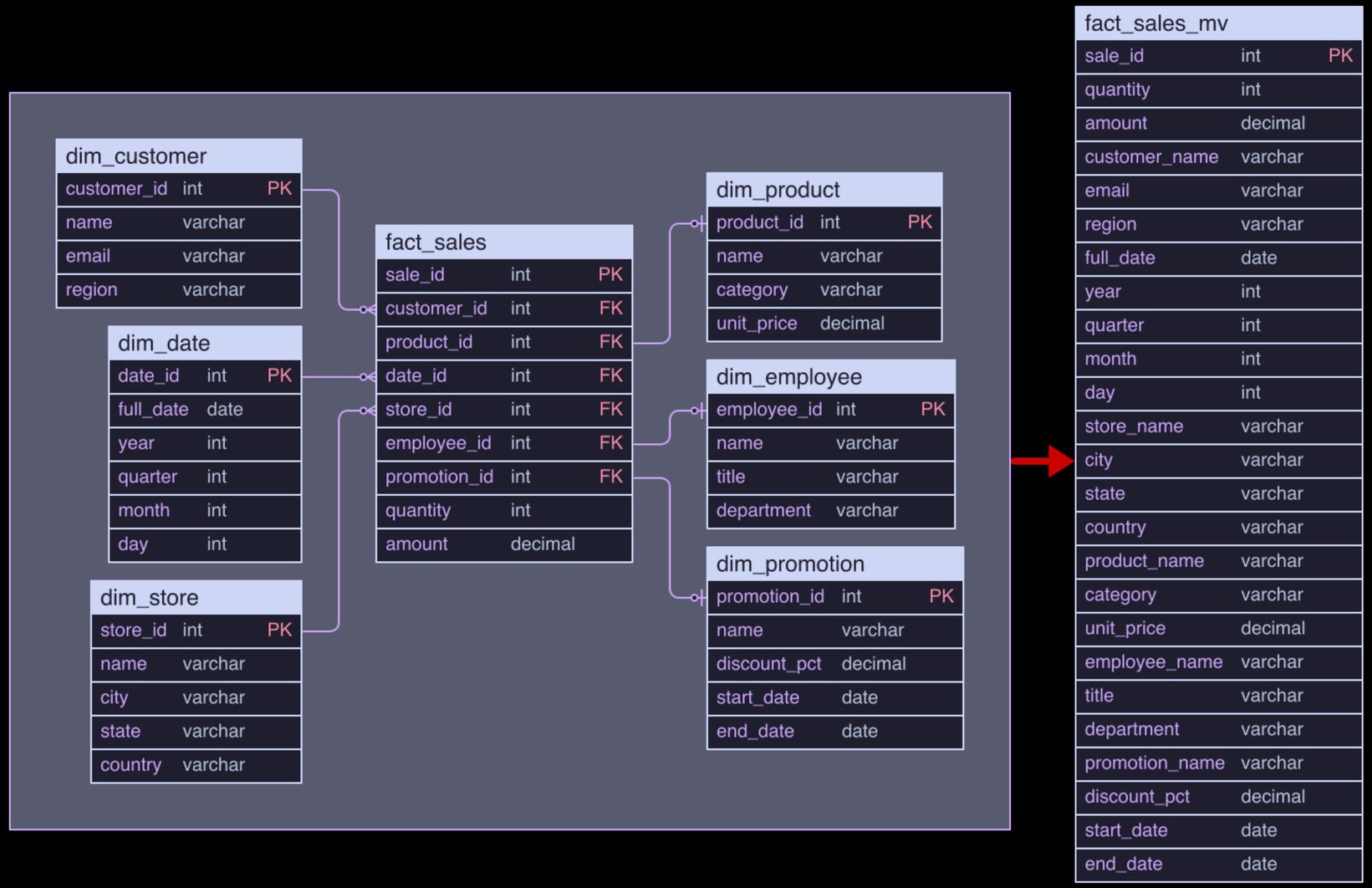
f(name = 'Bob') → f(customer_id = 212)

Transitive Statistics - Problems

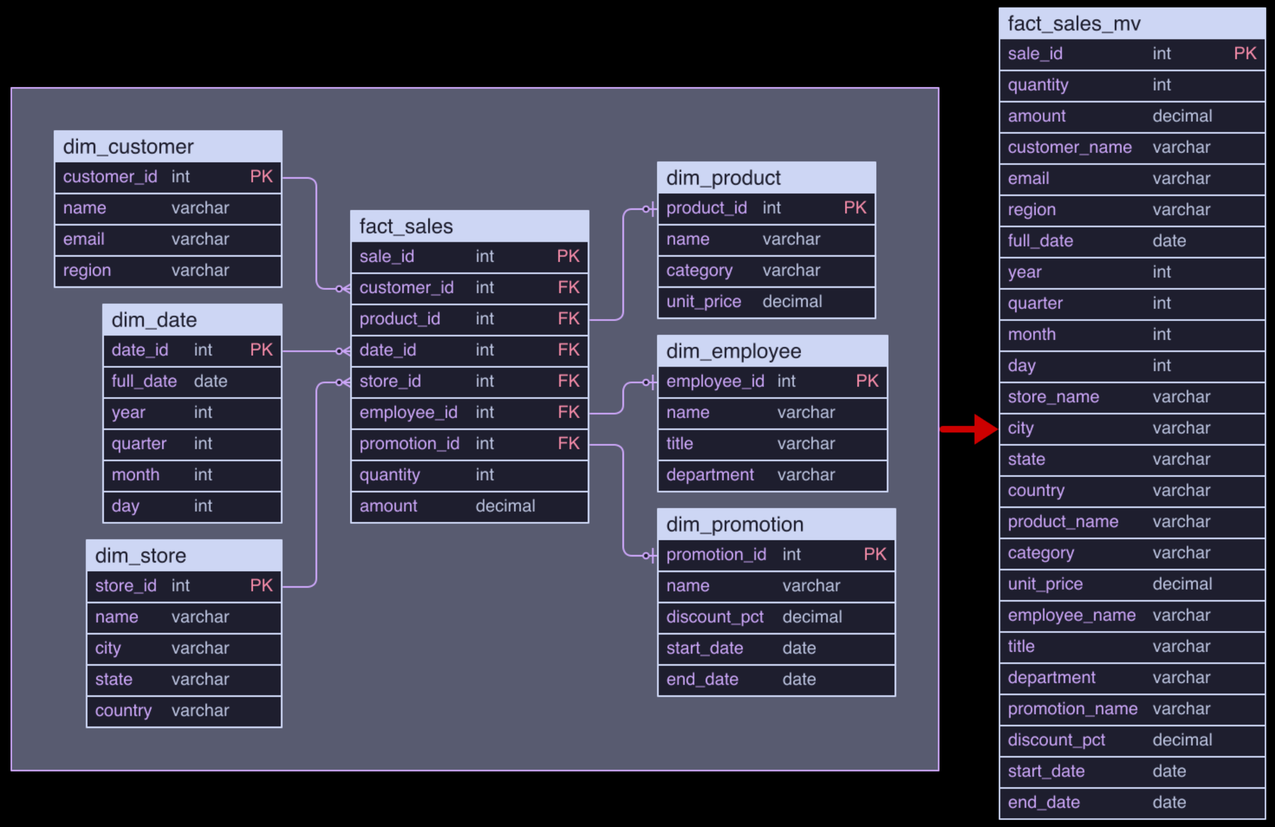


- ❌ Composite keys
- ❌ Samples have impossible misses
- ❌ Not one-to-one (values repeat)
- ❌ Stats can’t be re-weighted

Materialized Views



Materialized Views



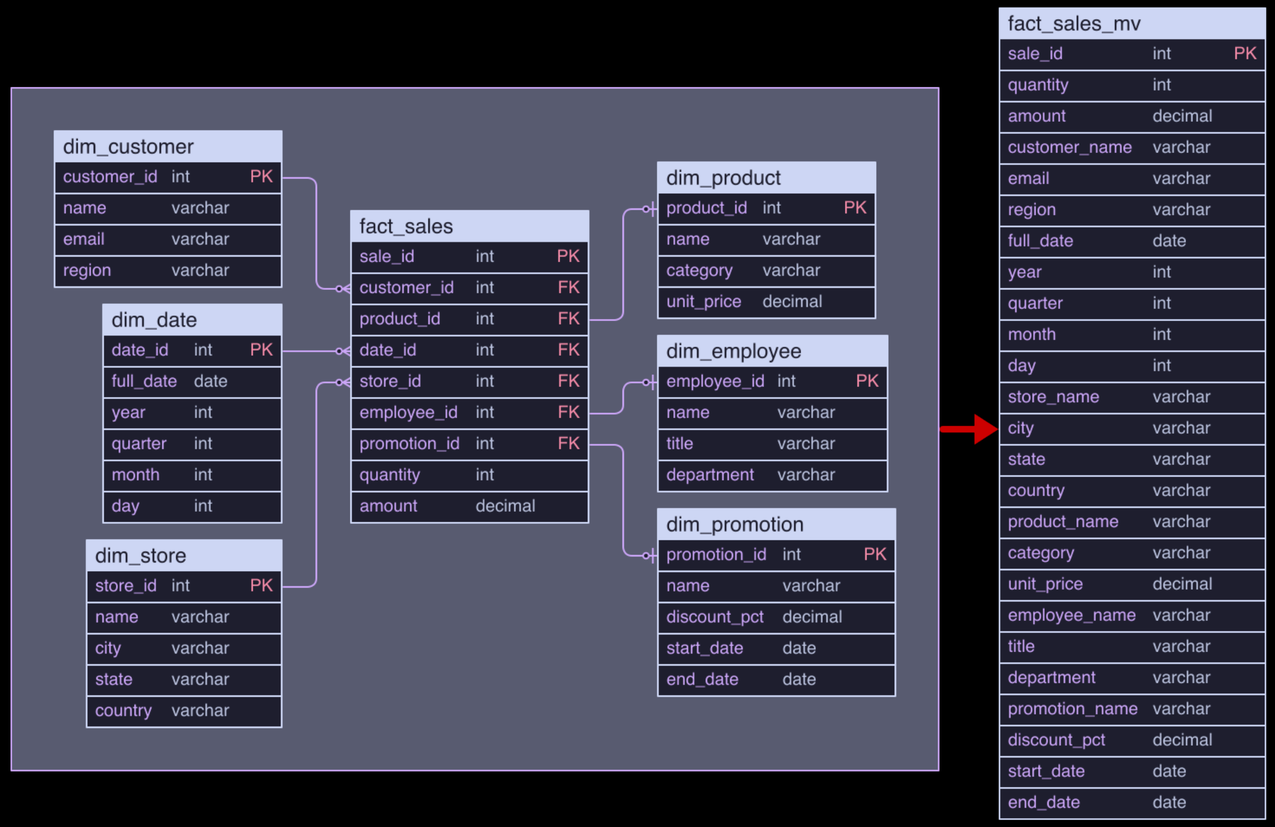
✅ Ubiquitous

✅ Simple

✅ Extended Statistics

❌ Update Sync

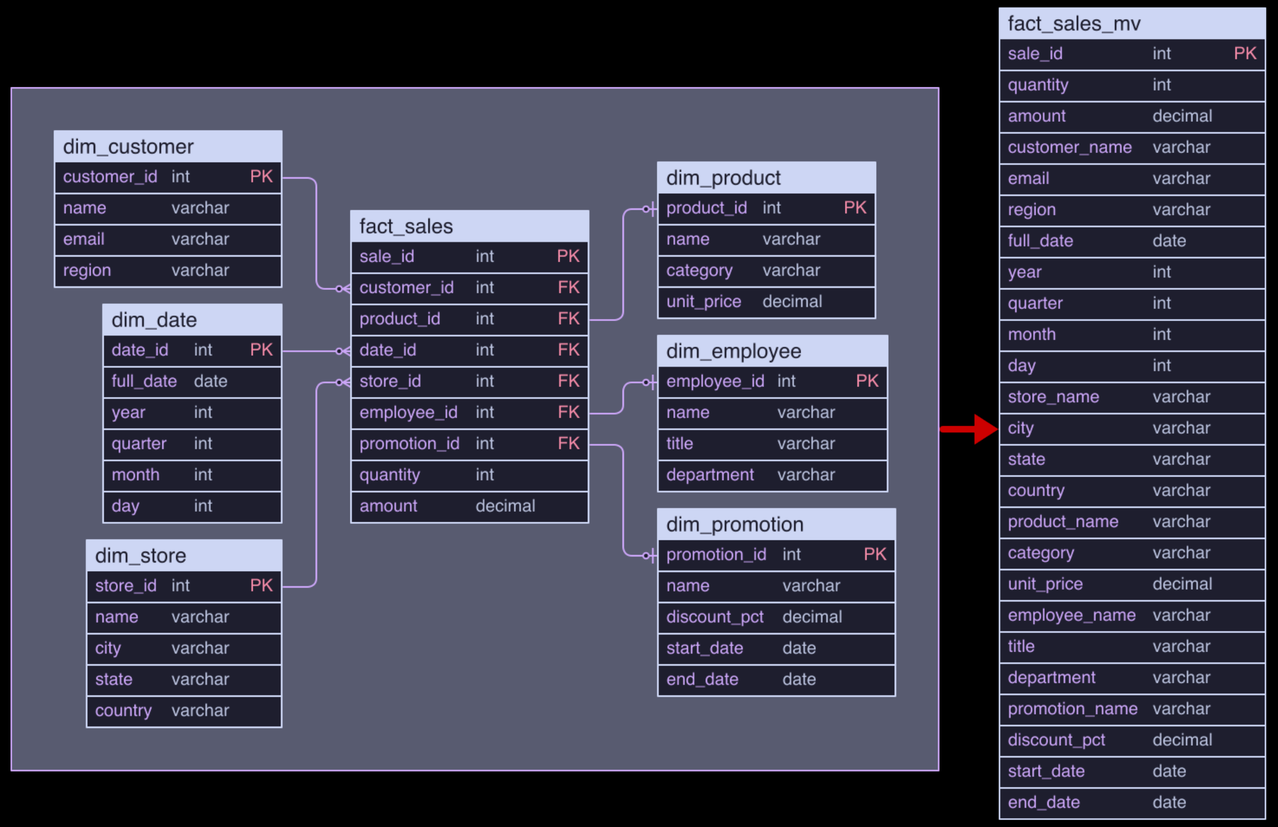
Materialized View Problems



fact_sales_mv		
sale_id	int	PK
quantity	int	
amount	decimal	
customer_name	varchar	
email	varchar	
region	varchar	
full_date	date	
year	int	
quarter	int	
month	int	
day	int	
store_name	varchar	
city	varchar	
state	varchar	
country	varchar	
product_name	varchar	
category	varchar	
unit_price	decimal	
employee_name	varchar	
title	varchar	
department	varchar	
promotion_name	varchar	
discount_pct	decimal	
start_date	date	
end_date	date	

- ❌ Storage Requirement
- ❌ Continuous maintenance
- ❌ Must invoke directly
- ❌ Or planner needs to see that join matches the MV

Materialized View Variations



SQLServer Indexed Views

Same as Matviews

Oracle Analytic Views

Autogenerated Matviews with GraphQL syntax

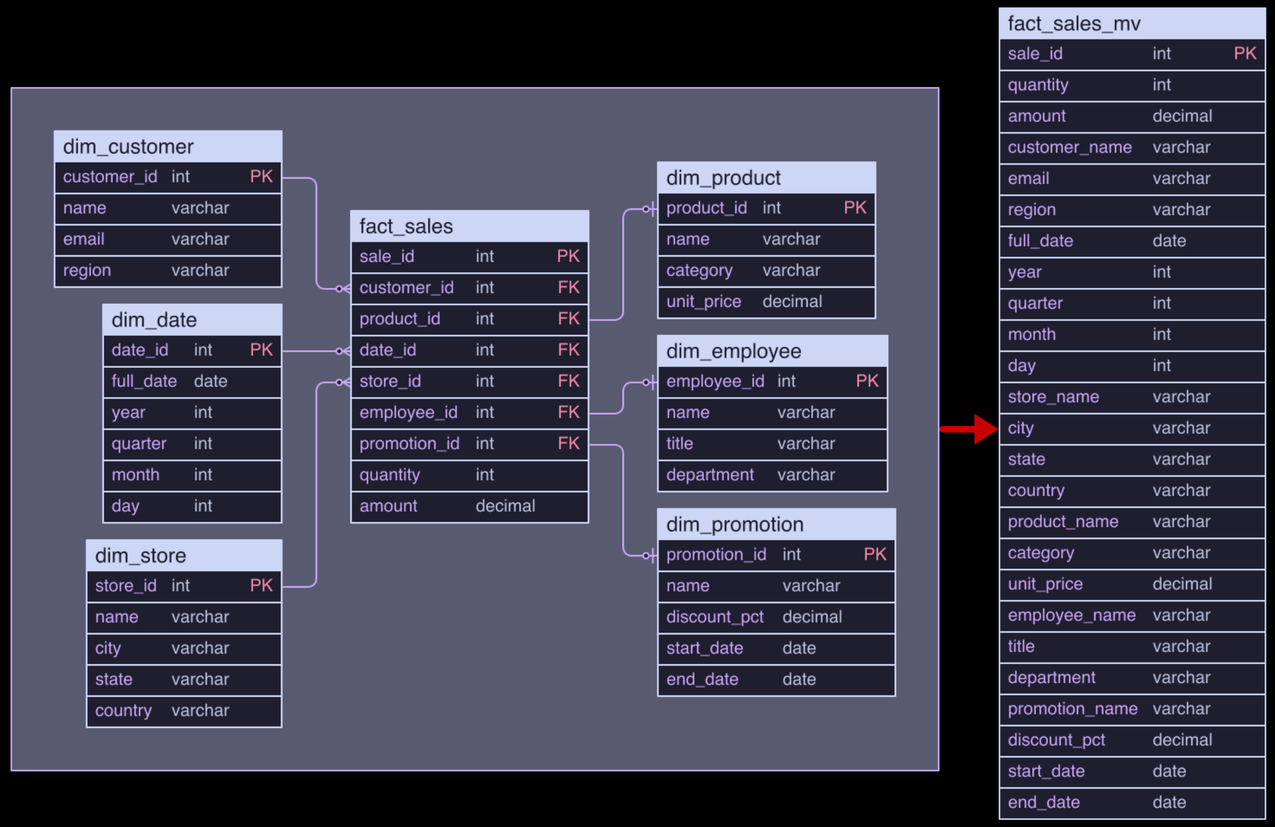
Oracle - Bitmap Join Indexes

To avoid a materialized view, Oracle can make join indexes:

```
CREATE BITMAP INDEX fact_dim_att_bji
ON fact_table(d.attribute_name)
FROM fact_table f, dim_table d
WHERE f.dim_id = d.dim_id
```

- Similar to Postgres GIN indexes
- Best for low cardinality
- Best for AND/OR chaining
- Slow to build
- Can be invalidated by updates
- Static Data Warehouse

Introducing Join Statistics



Storing statistics about columns in table B but associated with table A

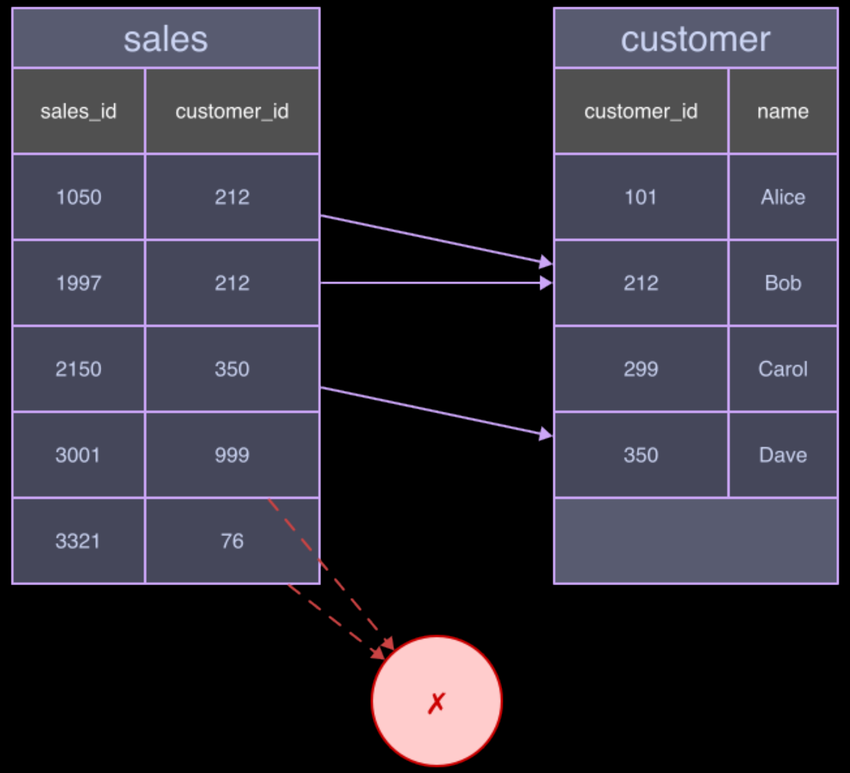
Sampling is cheaper and more flexible than materialization

Good News!

We've already got this

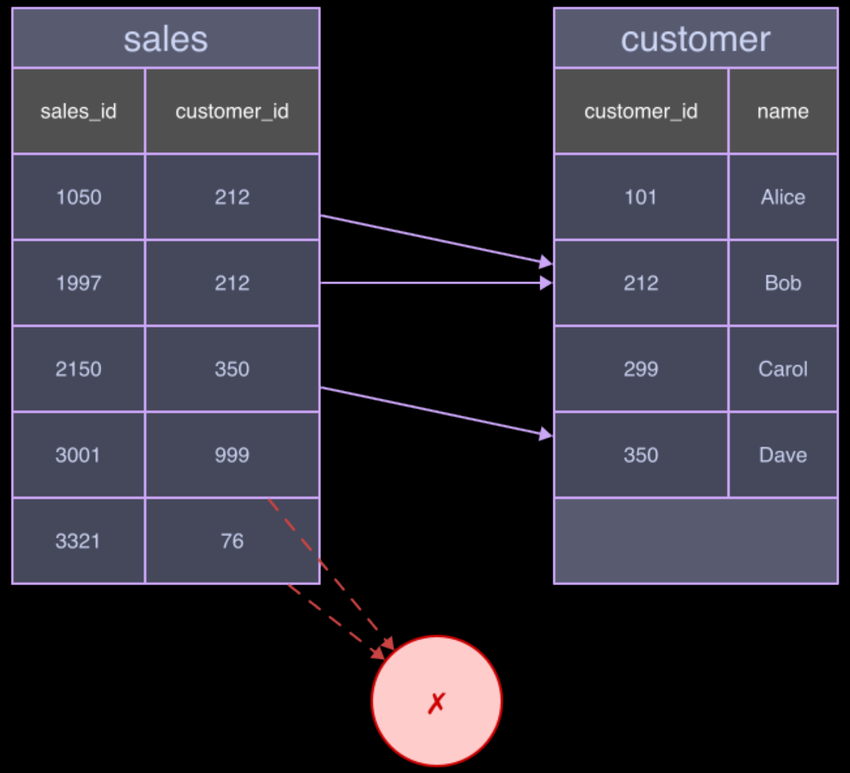
- Existing Proof Of Concept
- Update to patchset in last few weeks
- Author is (probably) in the room
- Closely follows an existing research paper (link in credits)

First Attempt



- Original Demo joined on existing MCV stats
- Join Misses Guaranteed
- Add new stats as planner learns to use them

Second Attempt



- ANALYZE requires index for probe of "far" table
- No more join misses
- Syntax allows N-way join
- Only implements 2-way joins for now

Declaring Join Statistics

Declare with a Join Clause?

```
CREATE STATISTICS my_join_stat(mcv)
ON d1.d1c1, d1.c1c2, d2.c3, f.c4
FROM fact AS f
JOIN dimension_1 AS d1
ON d1.pkey = f.d1key
JOIN dimension_2 AS d1
ON d1.pkey = f.d2key;
```

✅ Commonly understood join syntax

✅ Can map multiple joins

❌ Must Fit Query to Join Definition

Declare with an RI Constraint?

```
CREATE STATISTICS my_join_stat(mcv)
ON d1.d1c1, d1.c1c2, f.c4 -- attributes
FROM fact f
USING ri_constraint_on_dimension_2 d2
```

✅ RI constraints easily/cheaply discoverable

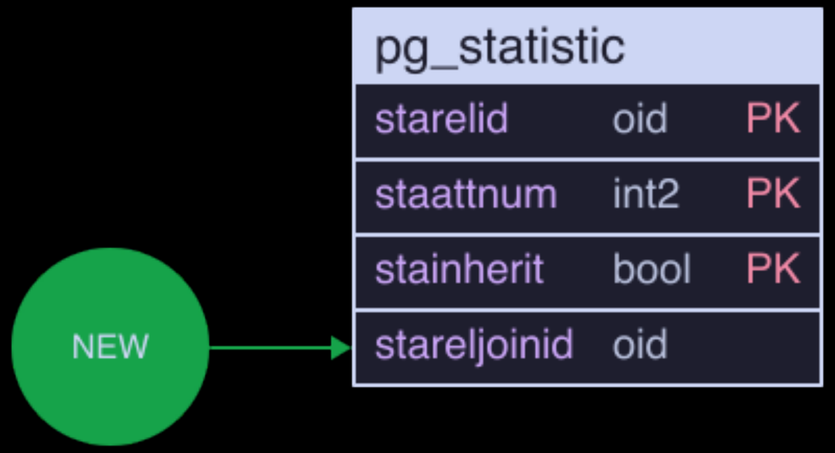
✅ Guarantees an indexed lookup bypassing SQL/SPI

❌ Nobody remembers the names of RI constraints

❌ Rules out Foreign Tables

Where to put these new stats?

Add a relation id



Use `InvalidOid` for regular stats

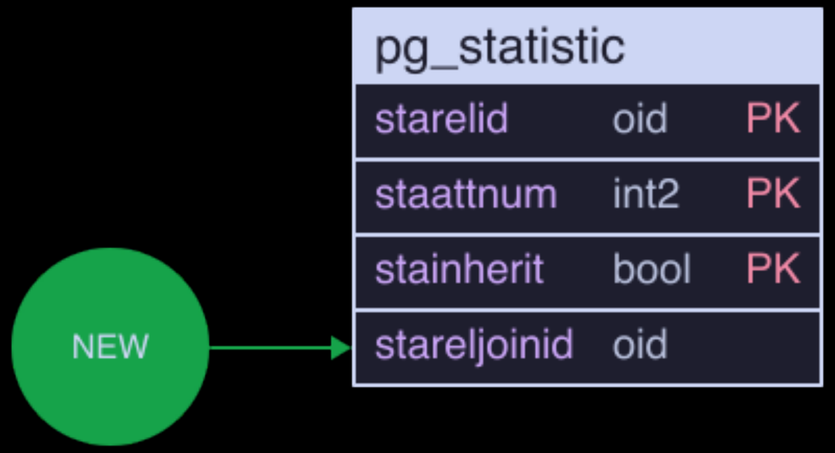
✅ allows regular stats for remote columns

❌ burdens regular statistics

❌ ambiguous if multiple joins to same table

❌ does not guarantee an index

Add a constraint id



✅ regular stats allowed

❌ burdens regular stats

✅ multiple joins allowed

✅ guarantees an index

❌ Requires an index

❌ No FDW joins possible

New Table - pg_statistic_join

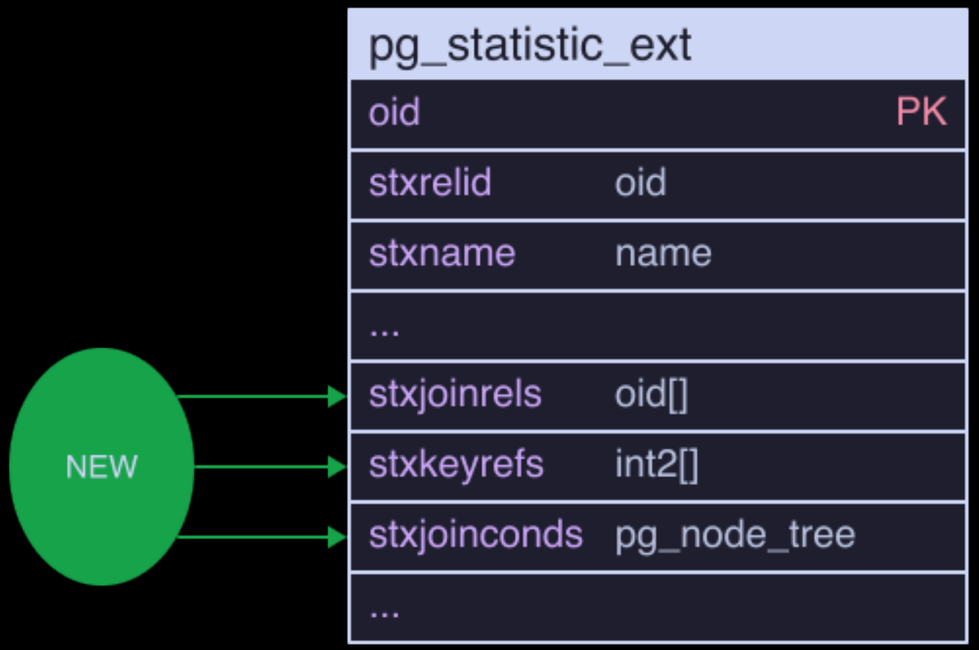
pg_statistic_join		
starelid	oid	PK
staattnum	int2	PK
stainherit	bool	PK
staconsjoinid	oid	

✅ Avoids `pg_statistic` friction

❌ planner has to go fishing in a third stats table

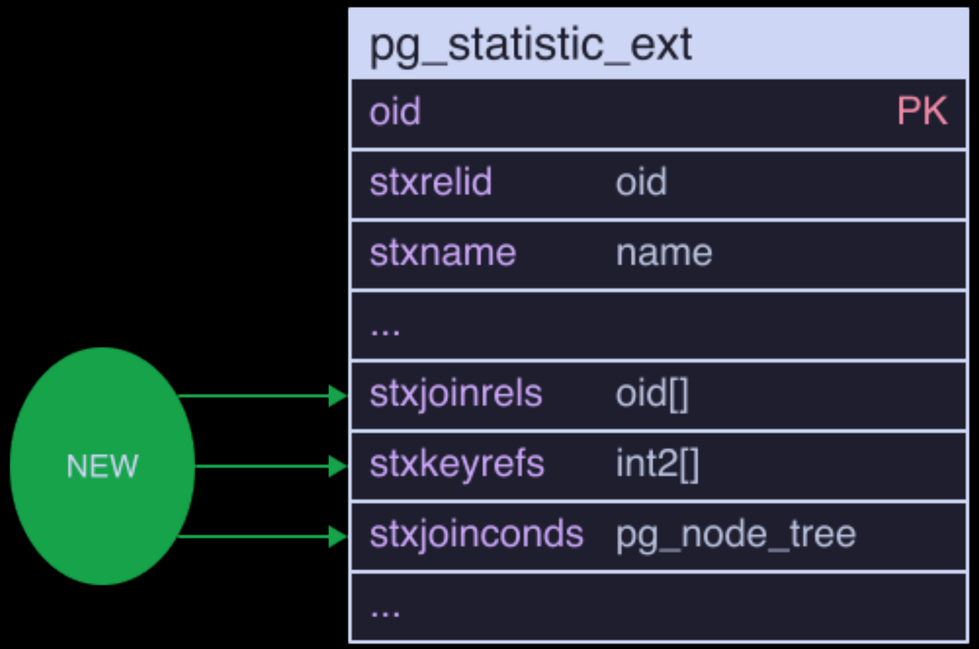
❌ new function `pg_restore_*_stats()`

Modify pg_statistic_ext



- ✅ It's where declared stats go now
- ✅ Fairly compact representation
- ✅ Multiple tables possible
- ❌ ~~Can't do multiple joins to same table~~
- ❌ Does not guarantee an index

Add constraint array to pg_statistic_ext



- ✅ Allows for multiple joins to the same table
- ✅ Guarantees an index
- ❌ Rules out pg_statistics
- ❌ Rules out FDWs
- ❌ Complex joins hard

DB2 Has Statistical Views

Introduced around v10 (approx 2012)

```
CREATE VIEW myjoinview ...;

ALTER VIEW myjoinview
    ENABLE QUERY OPTIMIZATION;

RUNSTATS myjoinview;
```

- Exactly 2 tables in the view definition
- Star joins require multiple stat views
- Recommends RI constraint
- Does not require RI constraint

DB2 Has Statistical Views - Curiosities

```
CREATE VIEW myjoinview ...;

ALTER VIEW myjoinview
    ENABLE QUERY OPTIMIZATION;

RUNSTATS myjoinview;
```

- Only considers data distribution stats like cardinality.
- Implies that is the only stat they have found useful.
- Does correlative stats on specificially identified pairs
- That's extended stats
- So DB2 effective has regular and extended stats

Can Postgres DB2, Too?

```
CREATE VIEW myjoinview ...;

ALTER VIEW myjoinview
    ENABLE QUERY OPTIMIZATION;

RUNSTATS myjoinview;
```

✅ Can use existing pg_statistic

✅ Can define extended statistics

❌ Planner has to know to look for these views

❌ Planner has to match join tree in query to join tree in view

✅ Proof of concept can already do that!

❌ New syntax on ALTER VIEW

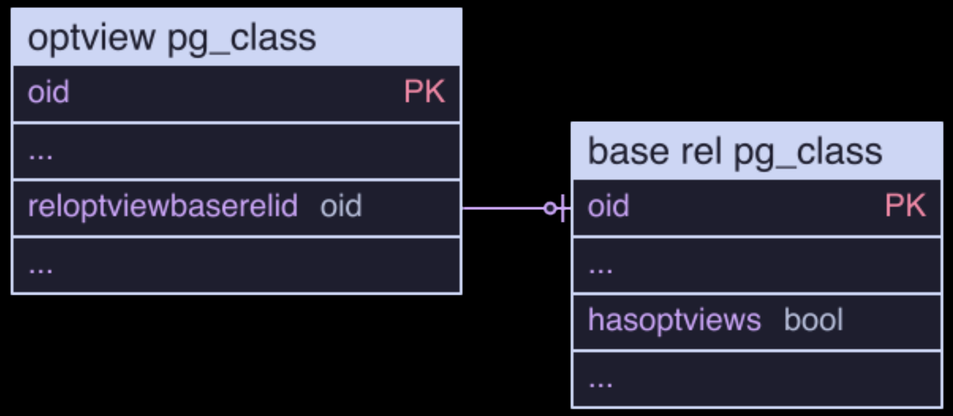
Matching query joins to existing view joins

```
CREATE VIEW myjoinview ...;

ALTER VIEW myjoinview
    ENABLE QUERY OPTIMIZATION;

RUNSTATS myjoinview;
```

- Oracle does this with Materialized Views ("Advanced Query Rewrite")
- DB2 clearly does this
- Proof Of Concept patch already does this!

Make View Matching Go Fast

Add to pg_class



OR

New `pg_constraint`

New `contype = o`

Planner likely scanned for constraints already

Alternative to ALTER VIEW

```
CREATE VIEW myjoinview ...;

ALTER VIEW myjoinview
    ENABLE QUERY OPTIMIZATION;

RUNSTATS myjoinview;
```

Just allow join criteria in `CREATE STATISTICS` and create the view under the hood.

✅ Matches existing paradigms

❌ Can't reference the view in a query

Must the view be updatable?

```
CREATE VIEW opt_view_1 AS
SELECT f.amount, d1.item_name,
       date_trunc('MONTH',
            d2.sales_date) AS sale_month
FROM sales_fact f
JOIN item d1
ON d1.item_id = f.item_id
JOIN time_dimension d2
ON d2.time_id = f.sales_time_id
WHERE d1.discontinued
AND f.amount > 1000
```

- Also known as "Key preserved"
- Means you every row in the view maps to exactly one row in one of the tables

Are stats needed on all columns?

```
CREATE VIEW myjoinview ...;

ALTER VIEW myjoinview
    ENABLE QUERY OPTIMIZATION;

RUNSTATS myjoinview;
```

- Can disable a column's stats with attstattarget = 0
- Existing views can be repurposed as optimizer stats views

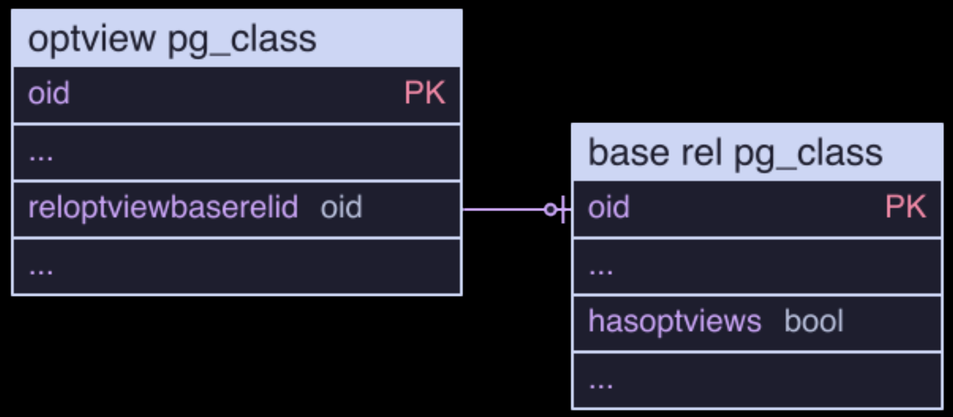
Can The View Have Non-Join Filters?

```
CREATE VIEW opt_view_1 AS
SELECT f.amount, d1.item_name,
       date_trunc('MONTH',
            d2.sales_date) AS sale_month
FROM sales_fact f
JOIN item d1
ON d1.item_id = f.item_id
JOIN time_dimension d2
ON d2.time_id = f.sales_time_id
WHERE d1.discontinued
AND f.amount > 1000
```

- Somewhat like Partial Indexes
- Reduces the Combinatorical Complexity for any extended stats defined

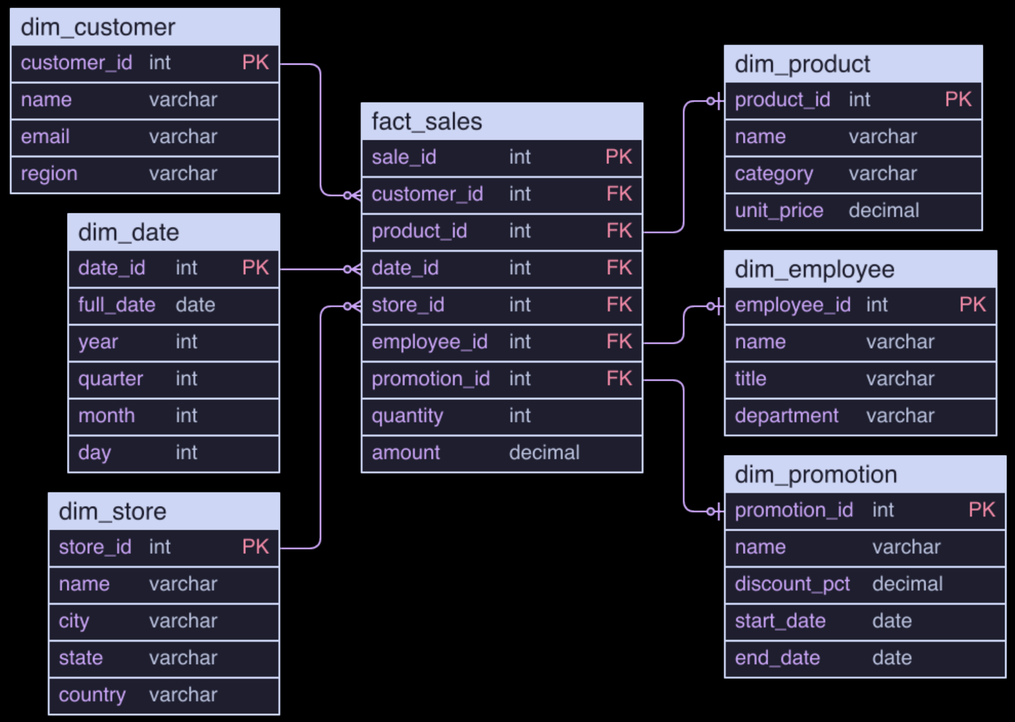
Multi joins

Add to pg_class



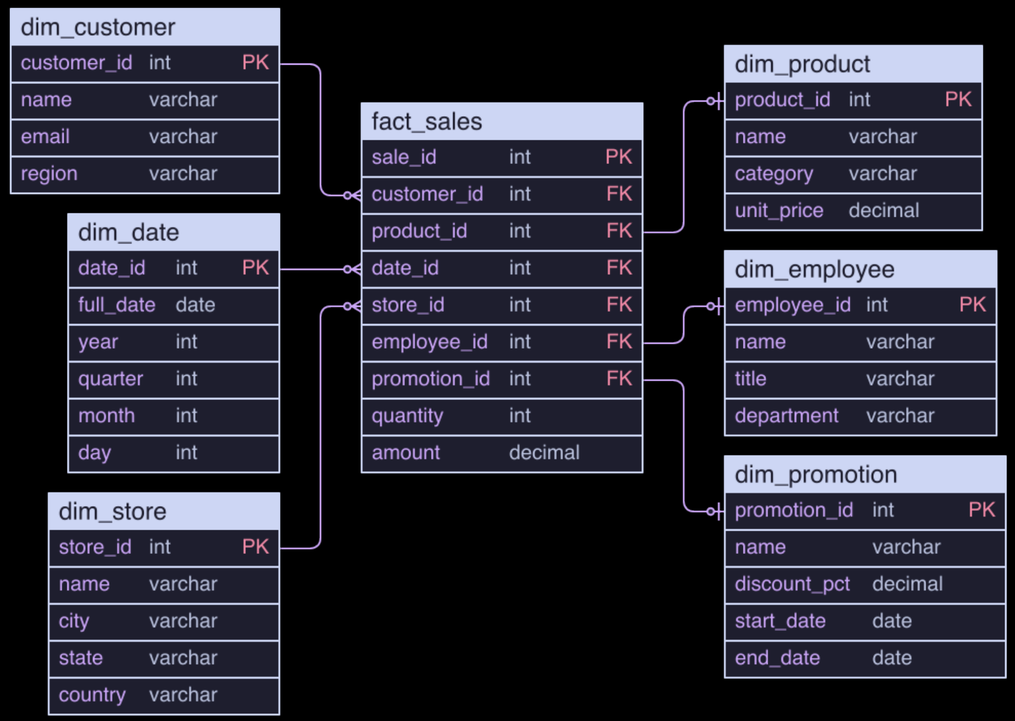
- Would this need an optimizer view for every possible join combination?
- Easier if RI + NOT NULL constraints. Hard otherwise.
- Which is faster, chipping away from multi-view or combining simple views?
- Are statistics reversible?

Actually Collecting the Statistics



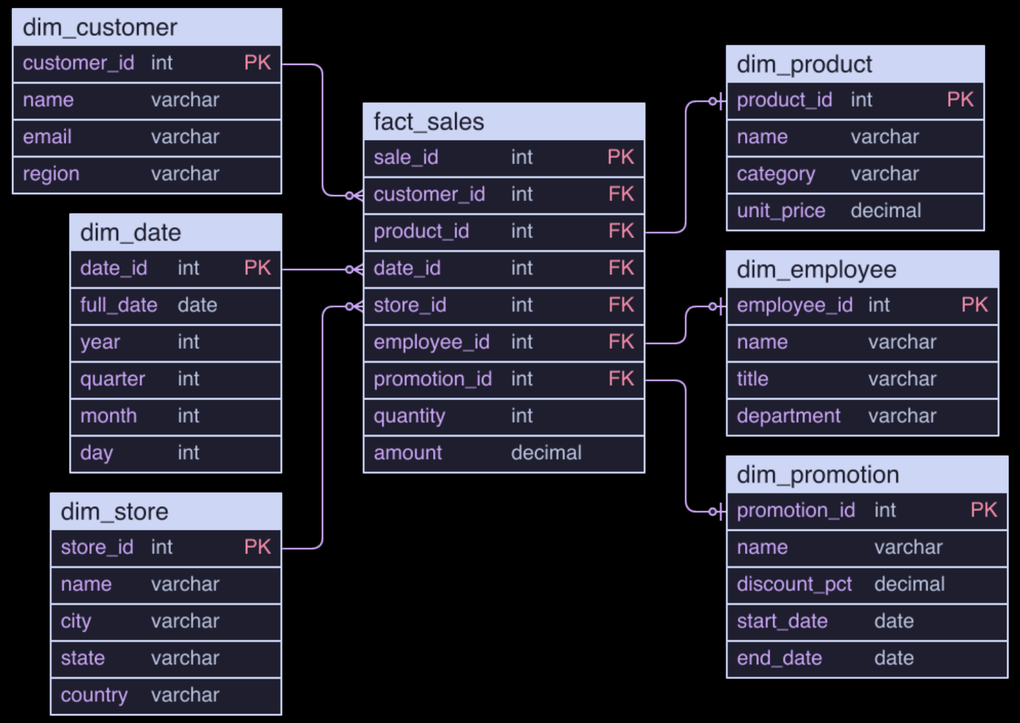
- Done during analyze of fact table.
- That, or re-sample for every view (yuck).
- Can’t rely on existing samples or stats.

Stats Collection: SPI



1. Create an `EphemeralNamedRelation` using the table sample
2. Get the optimizer view definition
3. Swap out the "fact" table with the ENR
4. Query using modified view definition

Stats Collection: RI Indexes



- Collect the table sample for the fact table
- For each table to be joined:
  - Sort the fact table sample by join columns
  - Open the RI constraint index
  - Merge-walk the sorted sample and index

Summary - Catalog

Add feature ALTER VIEW ... ENABLE OPTIMIZER STATISTICS

This operation will fail if view is not suitable.

Add pg_class.optviewbaserelid - "optimizer view base rel id" to all views that are optimizer enabled.

Add pg_class.hasoptviews - "has optimizer views" boolean to referenced relation

Summary - Planner

Only looks for optimizer views if Plan is a join and Base Relation has `hasoptviews` set

OR

Optimizer view is queried directly

Overhead would match what already exists for finding extended stats.

Anticipate Optimizer View usage to be slightly above Extended Statistics Usage.

Idle Musing #1

Enable Optimizer Stats By Default?

If somebody went to the trouble of defining the view, then then might want stats on the view

It would be obvious if the view were valid for optimizer stats at create time anyway.

Generally speaking, customers don't go wild with the number of views.

Idle Musing #2

Capture Join Stats on the fly?

Already doing the heavy lifting every time the view is queried

A hash/NL join that detects no join statistics could collect them as they go

This would require some sort of unique identification of joins

Just limit to equijoins?

Key Challenges For Next Development Cycle

- Agree on data model changes
- Find cheapest way to match query plans to views
- Avoid planning/costing overhead in regular cases

Conclusion

- Join stats are already proven to work
- Recommend following DB2’s Optimizer View pattern
- Could lead to additional ways view stats can be used
- This helps address the problem of Star join optimization

Reference Links

How Good Are Optimizers, Really?

https://www.vldb.org/pvldb/vol9/p204-leis.pdf

IBM DB2 Statistical Views

https://www.ibm.com/docs/en/db2/12.1.x?topic=optimization-statistical-views

Special Thanks

Alexandra Wang

Tomas Vondra

Andrei Lepikhov

Tom Lane

Conference attendees like YOU