crunchy data

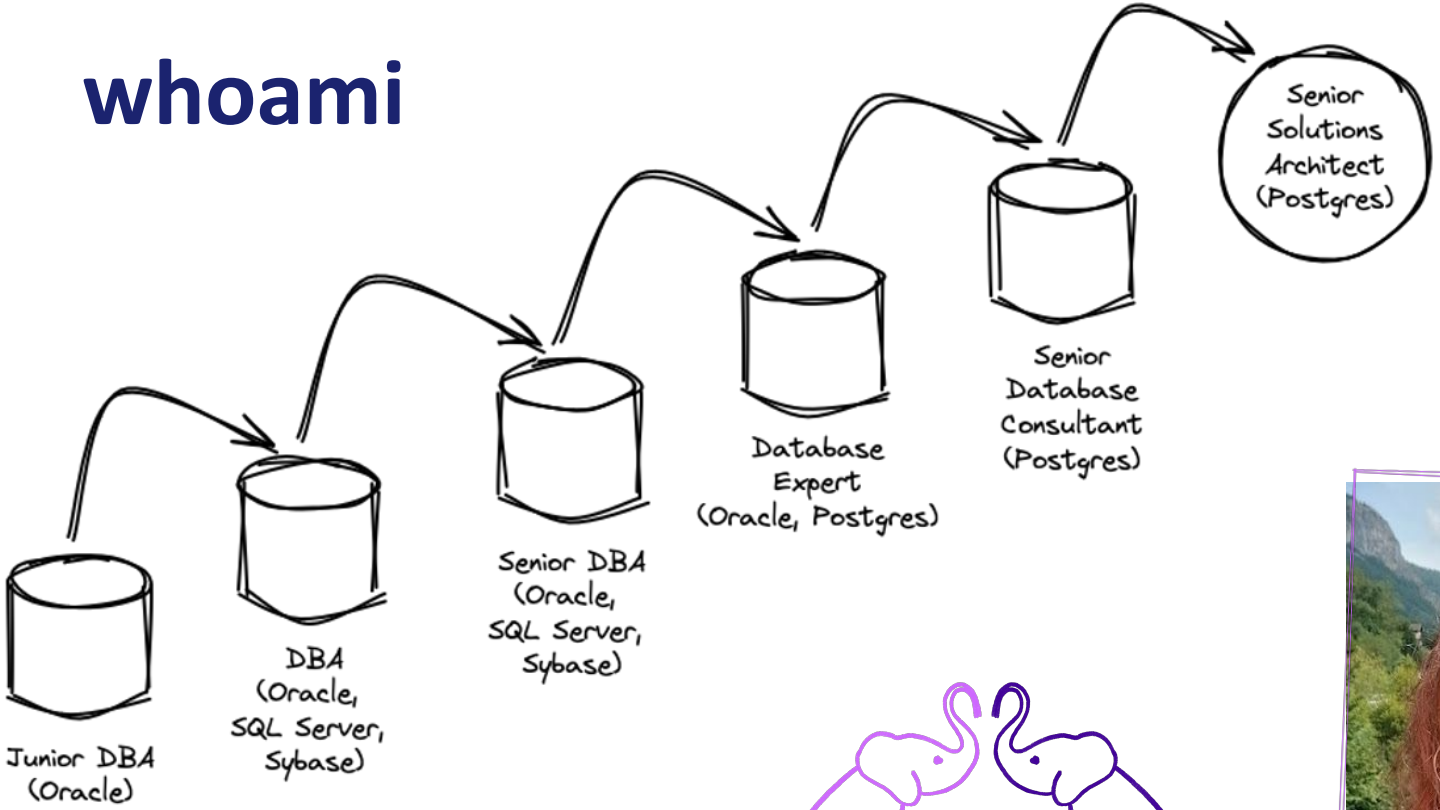
Optimising your Database for Analytics

Karen Jex | Senior Solutions Architect @ Crunchy Data

FOSDEM PostgreSQL Devroom | February 2025

Introduction

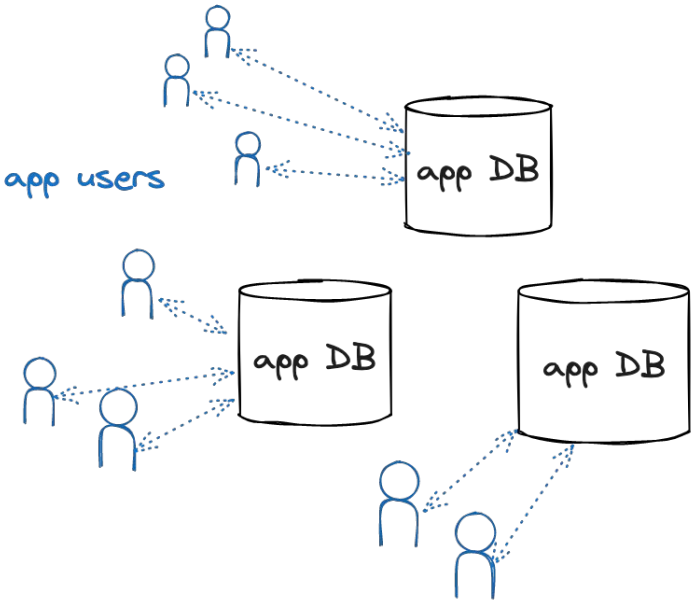
whoami



PostgreSQL Europe Diversity Task Force

Types of Workload

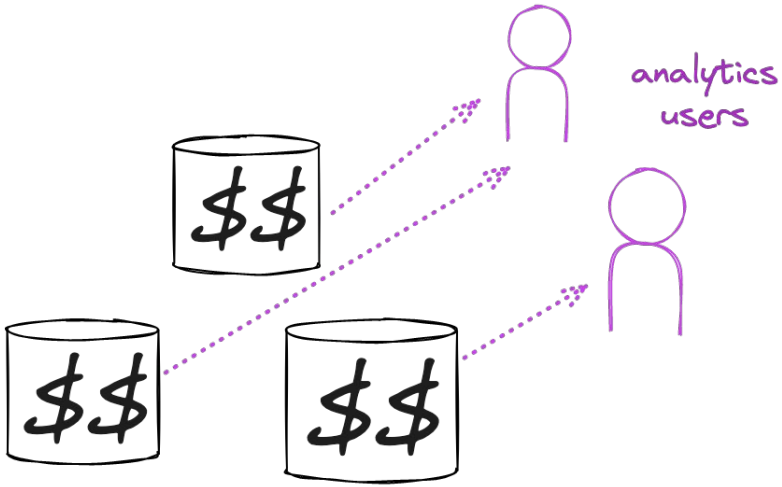
OLTP



- Online Transaction Processing
- Operational

- Concurrent users
- Short queries
- Multiple selects & updates

Value from Data - OLAP



- Online Analytical Processing
- Reporting
- Decision Support (DS)
- Business Intelligence (BI)

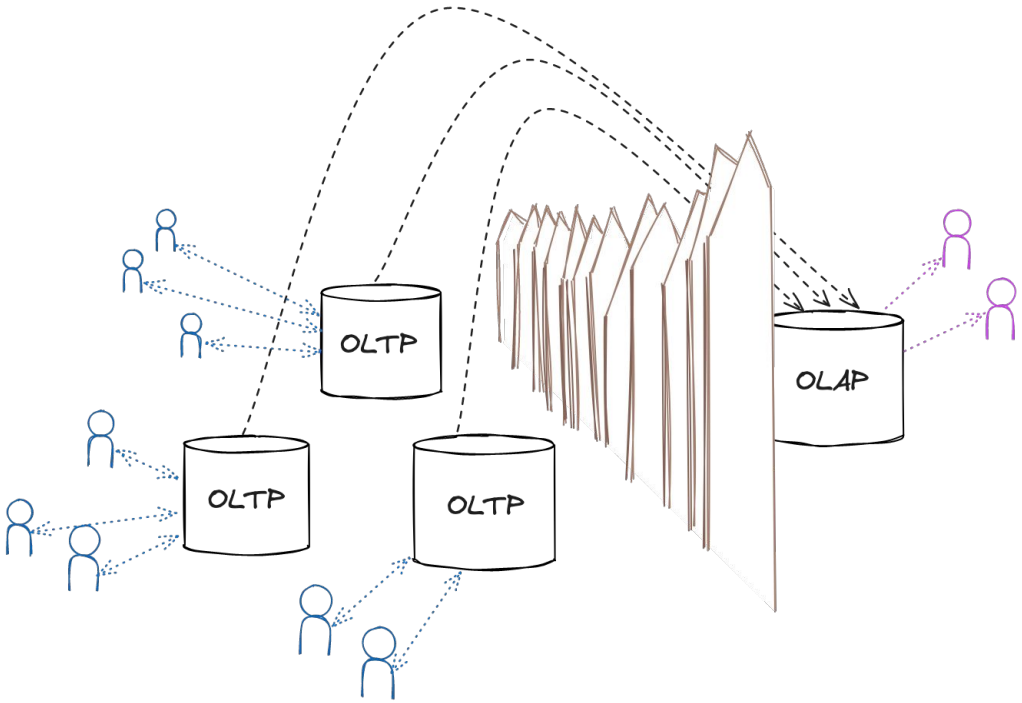
- Complex Queries
- Multiple tables
- Large data sets
- High resource utilisation

Introduction

OLTP & OLAP

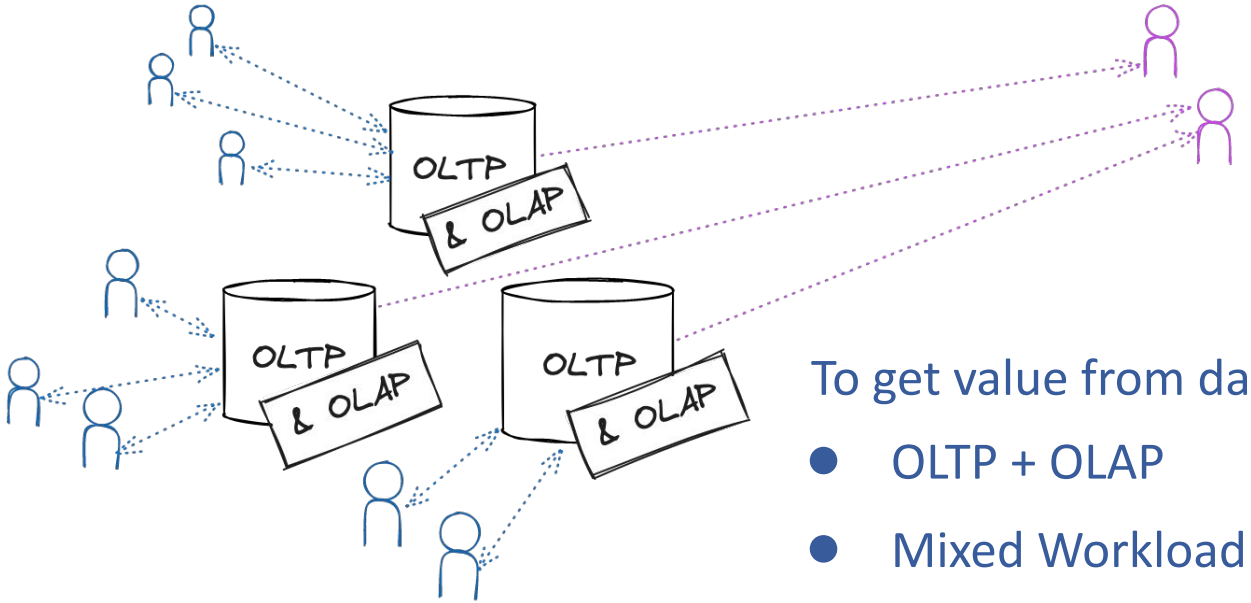
In the past:

- OLAP databases separate
- Loaded from OLTP systems
- Nightly or weekly batch processes



Introduction

OLTP & OLAP



To get value from data where it is, in real time

- OLTP + OLAP
- Mixed Workload
- Hybrid Workload

“Hybrid transaction/analytical processing (HTAP) is an emerging application architecture that "breaks the wall" between transaction processing and analytics. It enables more informed and "in business real time" decision making.”

Gartner: https://en.wikipedia.org/wiki/Hybrid_transactional/analytical_processing

What’s the problem?

The mixed workload CH-benCHmark

Dagstuhl "Robust Query Processing"
Breakout Group "Workload Management"[1]

ABSTRACT

While standardized and widely used benchmarks address either operational or real-time Business Intelligence (BI) workloads, the lack of a hybrid benchmark led us to the definition of a new, complex, mixed workload benchmark, called mixed workload CH-benCHmark. This benchmark bridges

focus on providing high availa
large number of users. OLAP sy
of business data to support str
computing sales revenue of a
regions and time, or fraud detc
port fast scans and complex da

Cole, Richard et al. "The Mixed Workload CH-BenCHmark." Proceedings of the Fourth International Workshop on Testing Database Systems (2011): n. pag. Web.

M2Bench: A Database Benchmark for Multi-Model Analytic Workloads

Bogyeong Kim
Seoul National University
Seoul, Republic of Korea
bgkim@dbs.snu.ac.kr

Kyoseung Koo
Seoul National University
Seoul, Republic of Korea
koo@dbs.snu.ac.kr

Undraa Enkhbat
Seoul National University
Seoul, Republic of Korea
undraae@dbs.snu.ac.kr

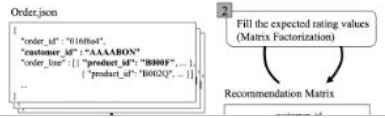
Sohyun Kim
Seoul National University
Seoul, Republic of Korea
chloek409@dbs.snu.ac.kr

Juhun Kim
Seoul National University
Seoul, Republic of Korea
johnjhkim@dbs.snu.ac.kr

Bongki Moon
Seoul National University
Seoul, Republic of Korea
bkmoon@snu.ac.kr

ABSTRACT

As the world becomes increasingly data-centric, the tasks dealt with by a database management system (DBMS) become more complex and diverse. Compared with traditional workloads that typically require only a single data model, modern-day computational tasks often involve multiple data sources and rely on more



Bogyeong Kim, Kyoseung Koo, Undraa Enkhbat, Sohyun Kim, Juhun Kim, and Bongki Moon. M2Bench: A Database Benchmark for Multi-Model Analytic Workloads. PVLDB, 16(4): 747 - 759, 2022. doi:10.14778/3574245.3574259

What's the Problem?

- Database optimised for OLTP activity
- Analytics queries: complex, long-running, resource-intensive
- Inefficient analytics queries
- Slow down existing application

Demo environment

Demo Environment

Table & Data

```
CREATE TABLE coffee_shop_sales (
   transaction_id NUMERIC,
   transaction_date DATE,
   transaction_time TIME,
   transaction_qty NUMERIC,
   store_id NUMERIC,
   store_location TEXT,
   product_id NUMERIC,
   unit_price NUMERIC,
   product_category TEXT,
   product_type TEXT,
   product_detail TEXT,
   sales_tax_pct NUMERIC);
```

https://www.kaggle.com/datasets/ahmedabbas757/coffee-sales

Demo Environment: Create and load temp table

```
CREATE TABLE coffee_shop_sales_temp
(transaction_id text,
 transaction_date text,
 transaction_time text,
 transaction_qty text,
 store_id text,
 store_location text,
 product_id text,
 unit_price text,
 product_category text,
 product_type text,
 product_detail text);

COPY coffee_shop_sales_temp
FROM '/Users/karen/coffee_shop/CoffeeShopSales.csv' WITH (FORMAT 'csv');
```

Demo Environment: Create and load temp table

```
CREATE TABLE coffee_shop_sales
(transaction_id numeric PRIMARY KEY,
 transaction_date date,
 transaction_time time,
 transaction_qty numeric,
 store_id numeric,
 store_location text,
 product_id numeric,
 unit_price numeric,
 product_category text,
 product_type text,
 product_detail text,
 sales_tax_pct numeric);
```

```
INSERT INTO coffee_shop_sales
SELECT CAST(transaction_id AS integer),
       TO_DATE(transaction_date, 'DD/MM/YYYY'),
       CAST(transaction_time AS time),
       CAST(transaction_qty AS integer),
       CAST(store_id AS integer),
       store_location,
       CAST(product_id AS integer),
       CAST(unit_price AS decimal),
       product_category,
       product_type,
       product_detail,
       CASE
         WHEN product_category
           IN ('Coffee','Drinking Chocolate','Tea') THEN 8
         WHEN product_category IN ('Bakery') THEN 9
         ELSE 7
       END
FROM coffee_shop_sales_temp;
```

Demo Environment: Generate extra data

```
-- mock data 12 months older than original data

INSERT INTO coffee_shop_sales
SELECT CAST(transaction_id AS integer)+149456,
       TO_DATE(transaction_date, 'DD/MM/YYYY') - interval '12 months',
       CAST(transaction_time AS time),
       CAST(transaction_qty AS integer),
       CAST(store_id AS integer),
       store_location,
       CAST(product_id AS integer),
       CAST(unit_price AS decimal),
       product_category,
       product_type,
       product_detail,
       CASE WHEN product_category in ('Coffee','Drinking Chocolate','Tea') THEN 8
            WHEN product_category IN ('Bakery') THEN 9
            ELSE 7
       END
FROM coffee_shop_sales_temp;

-- Repeat this step to add other data as required
```

Demo Environment: Gather Statistics

```
-- I kept adding data until I had 26.5 million rows:

SELECT COUNT(*) FROM coffee_shop_sales;
-- 26531424

-- gather stats on coffee_shop_sales table
ANALYZE coffee_shop_sales;
```

Demo Environment

Analytics Query

“Which store had the highest total sales for each month of the previous calendar year, taking into account only transactions with a value greater than 20?”

```
WITH ranked_sales AS (
  SELECT
    store_id::text||': '||store_location AS store,
    DATE_TRUNC('month', transaction_date) AS month,
    SUM(unit_price*(1+sales_tax_pct/100)*transaction_qty) AS total_monthly_sales,
    RANK() OVER (PARTITION BY DATE_TRUNC('month', transaction_date)
      ORDER BY SUM(unit_price*(1+sales_tax_pct/100)*transaction_qty) DESC) AS sales_rank
  FROM coffee_shop_sales
  WHERE date_trunc('year',transaction_date) = date_trunc('year',now()) - interval '1
year'
  AND unit_price*(1+sales_tax_pct/100)*transaction_qty>20
  GROUP BY 1, 2)
SELECT
  to_char(month, 'YYYY-MM') AS month,
  store,
  round(total_monthly_sales,2) AS total_monthly_sales
FROM ranked_sales
WHERE sales_rank = 1
ORDER BY month;
```

Demo Environment

Query Results

```
  month   |       store        | total_monthly_sales
----------+--------------------+---------------------
 2023-01  | 5: Lower Manhattan |            92228.86
 2023-02  | 5: Lower Manhattan |            74252.86
 2023-03  | 5: Lower Manhattan |            49454.54
 2023-04  | 8: Hell's Kitchen  |            84161.92
 2023-05  | 8: Hell's Kitchen  |           102180.72
 2023-06  | 3: Astoria         |          1908647.57
 2023-07  | 8: Hell's Kitchen  |            71506.82
 2023-08  | 8: Hell's Kitchen  |            40446.00
 2023-09  | 5: Lower Manhattan |            49454.54
 2023-10  | 3: Astoria         |          1383157.99
 2023-11  | 8: Hell's Kitchen  |           102180.72
 2023-12  | 8: Hell's Kitchen  |           128999.20
(12 rows)
```

Demo Environment

Execution Plan

```
EXPLAIN ANALYZE
WITH ranked_sales AS (
  SELECT
    store_id::text||': '||store_location AS store,
    DATE_TRUNC('month', transaction_date) AS month,
    SUM(unit_price*(1+sales_tax_pct/100)*transaction_qty) AS total_monthly_sales,
    RANK() OVER (PARTITION BY DATE_TRUNC('month', transaction_date)
      ORDER BY SUM(unit_price*(1+sales_tax_pct/100)*transaction_qty) DESC) AS
sales_rank
  FROM coffee_shop_sales
...
```

```
        QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=893953.86..893953.96 rows=40 width=96) (actual time=2948.391..2948.417 rows=12 loops=1)
   Sort Key: (to_char(ranked_sales.month, 'YYYY-MM'::text))
   Sort Method: quicksort  Memory: 25kB
   ->  Subquery Scan on ranked_sales  (cost=893569.08..893952.80 rows=40 width=96) (actual time=2948.344..2948.383 rows=12 loops=1)
        Filter: (ranked_sales.sales_rank = 1)
        ->  WindowAgg  (cost=893569.08..893851.67 rows=8074 width=80) (actual time=2948.338..2948.373 rows=12 loops=1)
                Run Condition: (rank() OVER (?) <= 1)
                ->  Sort  (cost=893569.08..893589.27 rows=8074 width=72) (actual time=2948.324..2948.351 rows=36 loops=1)
                        Sort Key: (date_trunc('month'::text, (coffee_shop_sales.transaction_date)::timestamp with time zone)),
(sum(((coffee_shop_sales.unit_price * ('1'::numeric + (coffee_shop_sales.sales_tax_pct / '100'::numeric))) * coffee_shop_sales.transaction_qty))) DESC
                        Sort Method: quicksort  Memory: 27kB
                        ->  Finalize GroupAggregate  (cost=890207.19..893045.12 rows=8074 width=72) (actual time=2927.807..2948.320 rows=36 loops=1)
                        Group Key: ((((coffee_shop_sales.store_id)::text || ': '::text) || coffee_shop_sales.store_location)),
(date_trunc('month'::text, (coffee_shop_sales.transaction_date)::timestamp with time zone))
                        ->  Gather Merge  (cost=890207.19..892661.61 rows=16148 width=72) (actual time=2927.677..2948.280 rows=108 loops=1)
                                Workers Planned: 2
                                Workers Launched: 2
                                ->  Partial GroupAggregate  (cost=889207.17..889797.70 rows=8074 width=72) (actual time=2891.938..2912.869 rows=36
loops=3)
                                        Group Key: ((((coffee_shop_sales.store_id)::text || ': '::text) || coffee_shop_sales.store_location)),
(date_trunc('month'::text, (coffee_shop_sales.transaction_date)::timestamp with time zone))
                                        ->  Sort  (cost=889207.17..889253.23 rows=18425 width=56) (actual time=2891.713..2895.854 rows=62219 loops=3)
                                                Sort Key: ((((coffee_shop_sales.store_id)::text || ': '::text) || coffee_shop_sales.store_location)),
(date_trunc('month'::text, (coffee_shop_sales.transaction_date)::timestamp with time zone))
                                                Sort Method: external merge  Disk: 3232kB
                                                Worker 0:  Sort Method: external merge  Disk: 3056kB
                                                Worker 1:  Sort Method: external merge  Disk: 3336kB
                                                ->  Parallel Seq Scan on coffee_shop_sales  (cost=0.00..887901.81 rows=18425 width=56) (actual
time=2.044..2874.587 rows=62219 loops=3)
                                                Filter: ((((unit_price * ('1'::numeric + (sales_tax_pct / '100'::numeric))) * transaction_qty) >
'20'::numeric) AND (date_trunc('year'::text, (transaction_date)::timestamp with time zone) = (date_trunc('year'::text, now()) - '1 year'::interval)))
                                                Rows Removed by Filter: 8781589
 Planning Time: 0.674 ms
 Execution Time: 2949.244 ms
```

Demo Environment

Execution Plan

(Parallel) Full Table Scan: `Parallel Seq Scan on coffee_shop_sales`

Sort on Disk: `Sort Method: external merge  Disk: 3232kB`

Execution Time: `2949.244 ms`

Demo Environment

OLTP Activity: pgbench

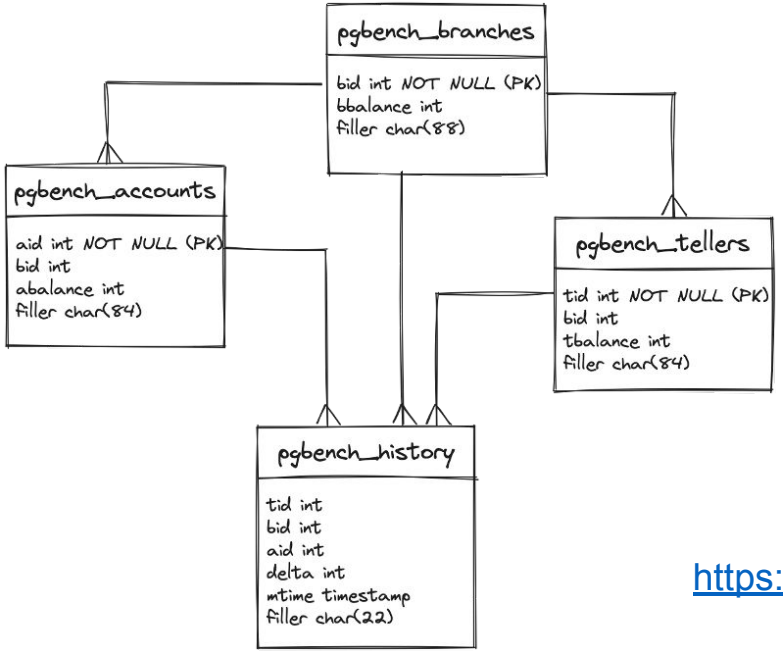


table	# of rows
pgbench_branches	1
pgbench_tellers	10
pgbench_accounts	100000
pgbench_history	0

https://www.postgresql.org/docs/current/pgbench.html

```
BEGIN;

 UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;

 SELECT abalance FROM pgbench_accounts WHERE aid = :aid;

UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;

 UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;

 INSERT INTO pgbench_history (tid, bid, aid, delta, mtime)
     VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);

END;
```

What can you do?

What can you do?

Configuration Parameters

Tuning PostgreSQL to Work Even Better

https://www.youtube.com/watch?v=7CnqVoMxoeo

What can you do?

Configuration Parameters

- Entire PostgreSQL instance
- Specific users
- Individual sessions
- Individual queries

Configuration Parameters

Database Connections

- max_connections parameter
  - Entire PostgreSQL instance
  - Default: 100
- Connection pooling
  - Separate pool for analytics connections

Configuration Parameters

work_mem

- Default: 4MB
- Increase for sessions performing large sort / hash operations
- Check for temp_files (set log_temp_files=0)
- Entire PostgreSQL instance or specific sessions
- CAUTION! If setting for entire instance

https://www.postgresql.org/docs/current/runtime-config-resource.html#GUC-WORK-MEM

```
SHOW work_mem;

BEGIN;

SET LOCAL work_mem=20480;

SHOW work_mem;

-- Execute analytics query

END;

SHOW work_mem;
```

```
 work_mem
----------
 4MB
(1 row)

 work_mem
----------
 20MB
(1 row)

 work_mem
----------
 4MB
(1 row)
```

Configuration Parameters

statement_timeout

- Abort statements that run for too long
- Default: 0 (disabled)
- Entire PostgreSQL instance or specific sessions
- CAUTION! if setting for entire instance

https://www.postgresql.org/docs/current/runtime-config-client.html#GUC-STATEMENT-TIMEOUT

Configuration Parameters

log_min_error_statement

- Entire PostgreSQL instance
- Default: ERROR

DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL, PANIC

https://www.postgresql.org/docs/16/runtime-config-logging.html#GUC-LOG-MIN-ERROR-STATEMENT

What can you do?

Indexing Strategy

What can you do?

Indexing Strategy

- Specific indexes for analytics queries
- Not too many

What can you do?

Indexes on Expressions

https://www.postgresql.org/docs/current/indexes-expressional.html

```
CREATE INDEX css_total_sale_amt
ON coffee_shop_sales(
     (unit_price*(1+sales_tax_pct/100)*transaction_qty));
```

```
Bitmap Index Scan on css_total_sale_amt
```

What can you do?

Pre-Calculate Data

What can you do: Pre-Calculate Data

Stored, Generated Columns

```
ALTER TABLE coffee_shop_sales
  ADD COLUMN total_sale_amt numeric
  GENERATED ALWAYS AS
    ((unit_price*(1+sales_tax_pct/100)*transaction_qty)) STORED;
```

https://www.postgresql.org/docs/current/ddl-generated-columns.html

```
WITH ranked_sales AS (
  SELECT
     store_id::text||': '||store_location AS store,
     DATE_TRUNC('month', transaction_date) AS month,
     SUM(total_sale_amt) AS total_monthly_sales,
     RANK() OVER (PARTITION BY DATE_TRUNC('month', transaction_date)
               ORDER BY SUM(total_sale_amt) DESC) AS sales_rank
  FROM coffee_shop_sales
  WHERE date_trunc('year', transaction_date) = date_trunc('year', now()) - interval '1
year'
  AND total_sale_amt>20
  GROUP BY 1, 2)
SELECT
  to_char(month, 'YYYY-MM') AS month,
  store,
  round(total_monthly_sales,2) AS total_monthly_sales
FROM ranked_sales
WHERE sales_rank = 1
ORDER BY month;
```

What can you do: Pre-Calculate Data

Materialized Views

- Physical copy of query results
- Pre-aggregation
- Refresh as needed
- Add Indexes

https://www.postgresql.org/docs/current/sql-creatematerializedview.html

What can you do: Pre-Calculate Data

Materialized Views

```
CREATE MATERIALIZED VIEW ranked_sales_mv AS (
 SELECT
    store_id::text||': '||store_location AS store,
    DATE_TRUNC('month', transaction_date) AS month,
    SUM(unit_price*(1+sales_tax_pct/100)*transaction_qty) AS
total_monthly_sales,
    RANK() OVER (PARTITION BY DATE_TRUNC('month', transaction_date)
      ORDER BY SUM(unit_price*(1+sales_tax_pct/100)*transaction_qty)
DESC) AS sales_rank
  FROM coffee_shop_sales
  WHERE date_trunc('year',transaction_date) = date_trunc('year',now()) -
interval '1 year'
  AND unit_price*(1+sales_tax_pct/100)*transaction_qty>20
  GROUP BY 1, 2);
```

What can you do: Pre-Calculate Data

Materialized Views

```
SELECT
  to_char(month, 'YYYY-MM') AS month,
  store,
  round(total_monthly_sales,2) AS total_monthly_sales
FROM ranked_sales_mv
WHERE sales_rank = 1
ORDER BY month;
```

What can you do: Pre-Calculate Data

Refreshing Materialized Views

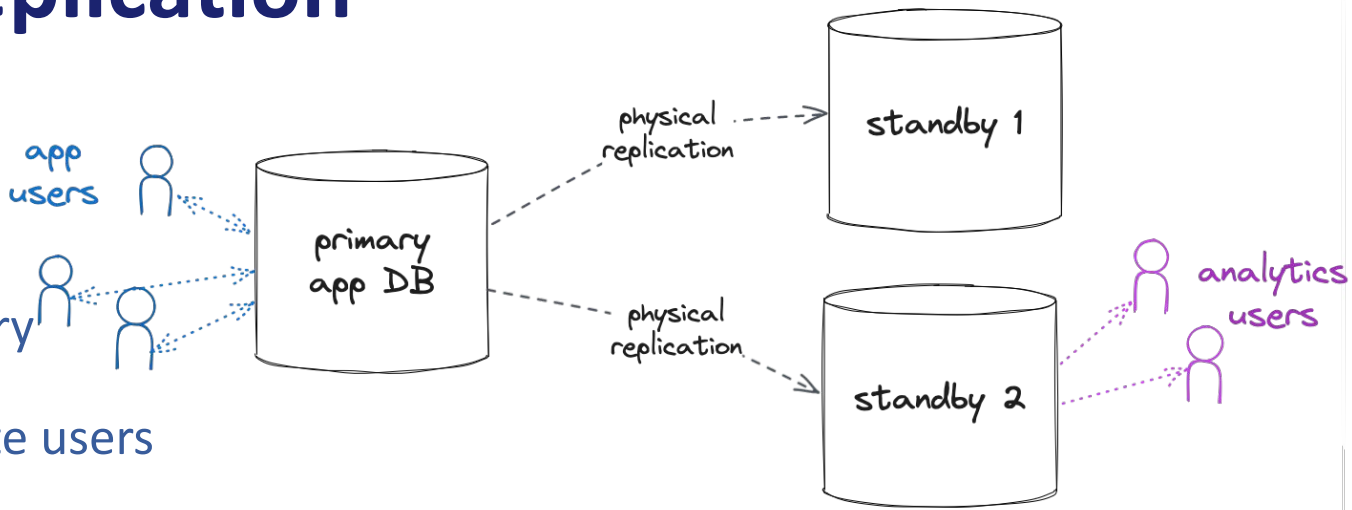
```
SELECT cron.schedule('@hourly',
  'REFRESH MATERIALIZED VIEW CONCURRENTLY ranked_sales_mv');
```

What can you do?

Separate Analytics Database

Separate Analytics Database

Physical Replication



Exact copy of primary

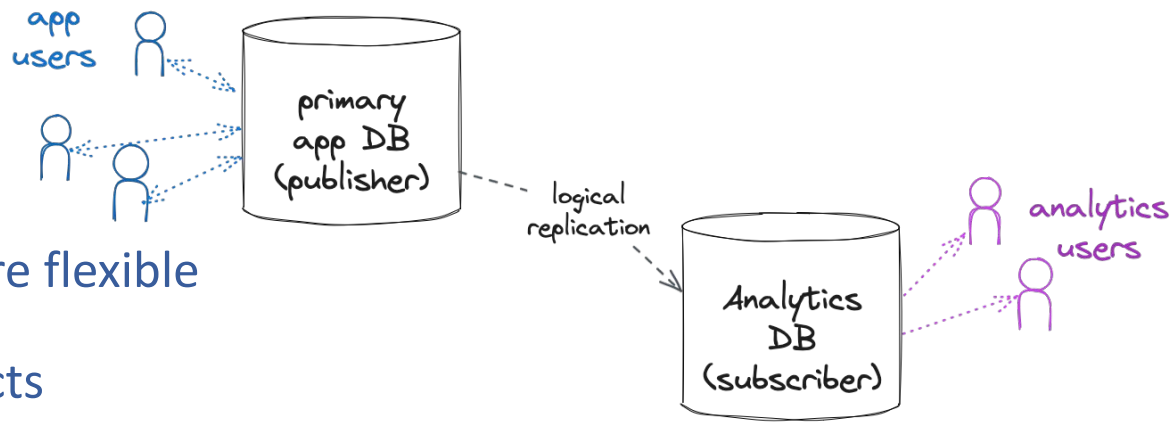
Can’t create separate users

Can’t add indexes

Can’t change layout of schema

Separate Analytics Database

Logical Replication



More complicated but more flexible

Replicate just certain objects

Replicate to multiple targets and/or from multiple sources

Can create additional objects, users etc.

Summary

Summary

- Separate Analytics Database
- Set Configuration Parameters
- Check Indexing Strategy
- Pre-Calculate/Pre-Aggregate Data
- Combine the techniques

Thank You!

@karenhjex | @karenhjex@mastodon.online | https://karenjex.blogspot.com/