STORMATICS

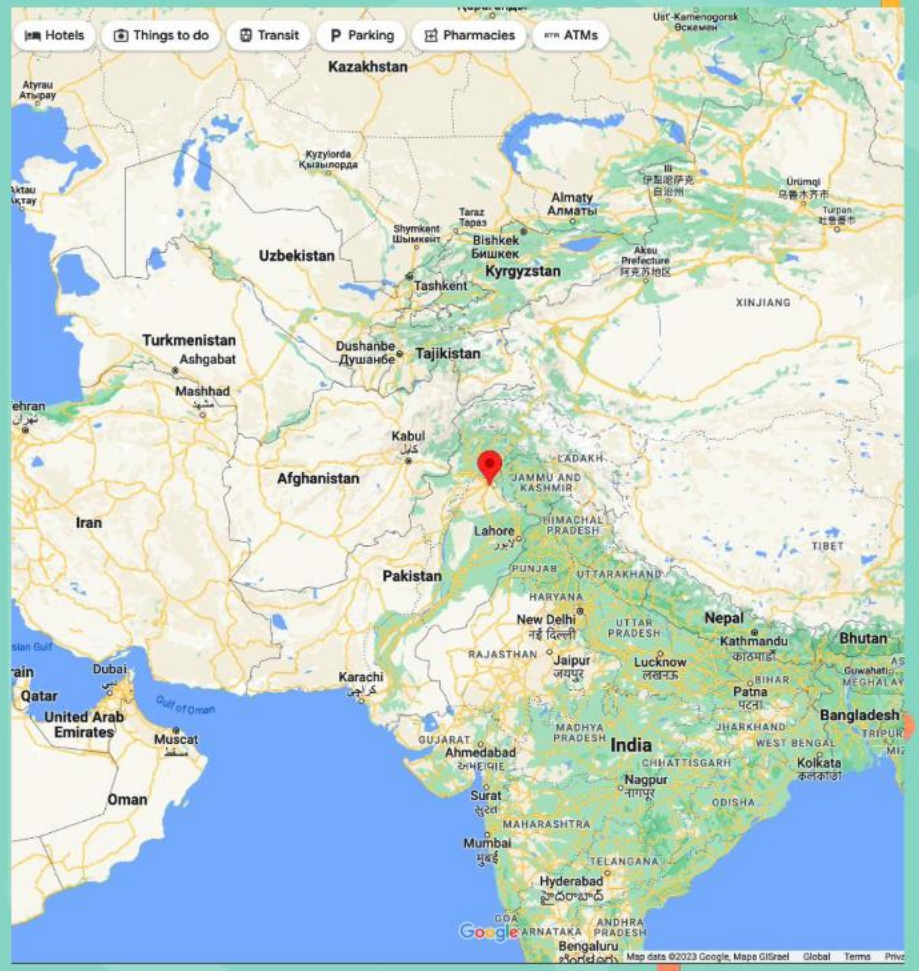
PostgreSQL Performance for Application Developers

Because There is no Magic Button

PGBootCamp.ru 16-April-2024

STORMATICS

```
postgres=# select * from umair;

-[ RECORD 1 ]-------------------------------
name        | Umair Shahid
description | 20+ year PostgreSQL veteran
company     | Stormatics
designation | Founder
location    | Islamabad, Pakistan
family      | Mom, Wife & 2 kids
kid1        | Son, 17 year old
kid2        | Daughter, 14 year old
```



OpenSCG™ PostgreSQL, Java & Linux Experts

2ndQuadrant® PostgreSQL

PERCONA

STORMATICS

STORMATICS

Professional Services for PostgreSQL

Our mission is to help businesses scale PostgreSQL reliably for mission-critical data

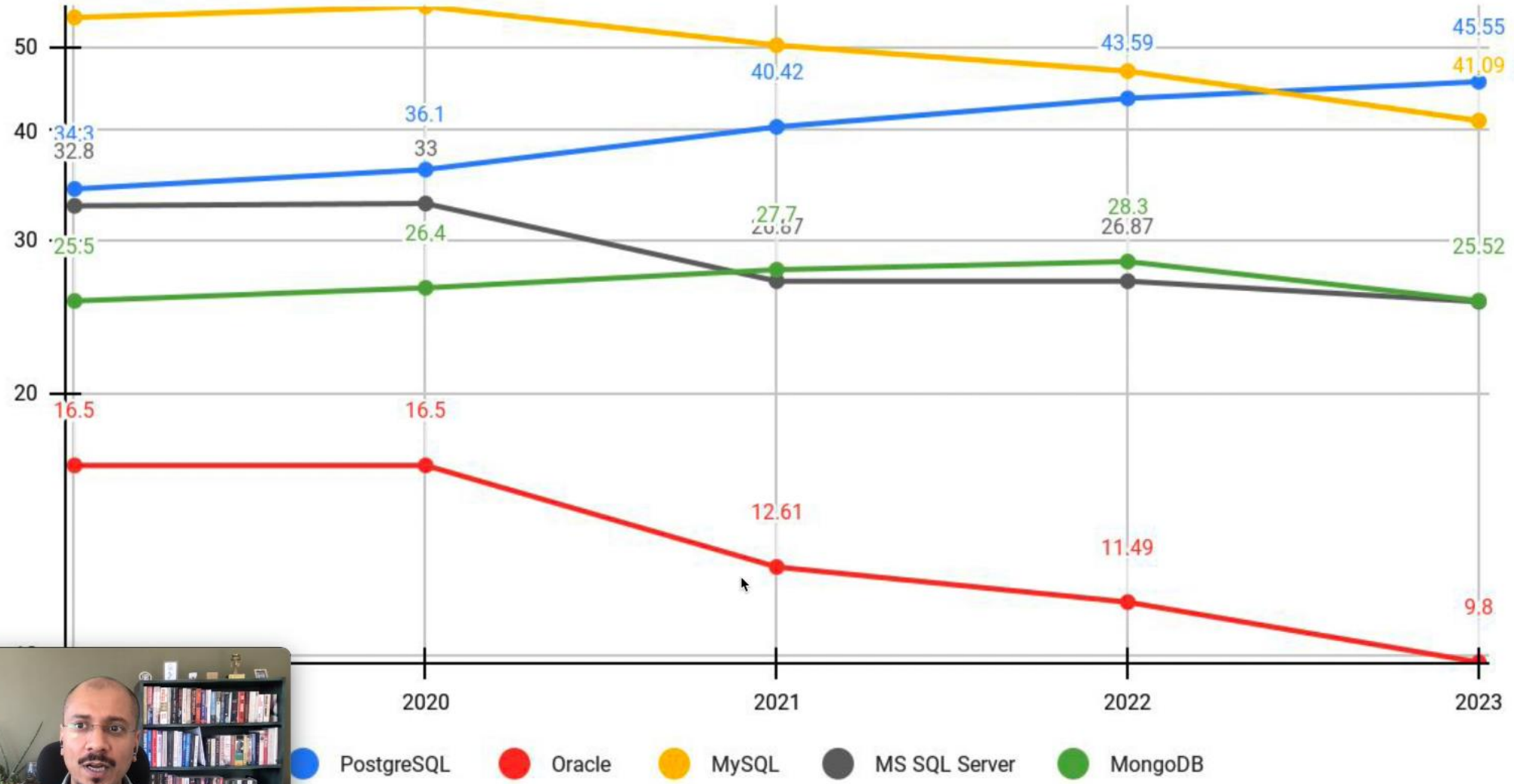
STORMATICS

Abstract

- PostgreSQL adoption is exploding and the move to the cloud is fueling it
- The difference between kicking things off and scaling in production
- The four areas of focus for scaling PostgreSQL
  - Query & SQL Optimization
  - Performance Features
  - Architectural Improvements
  - Parameter Tuning

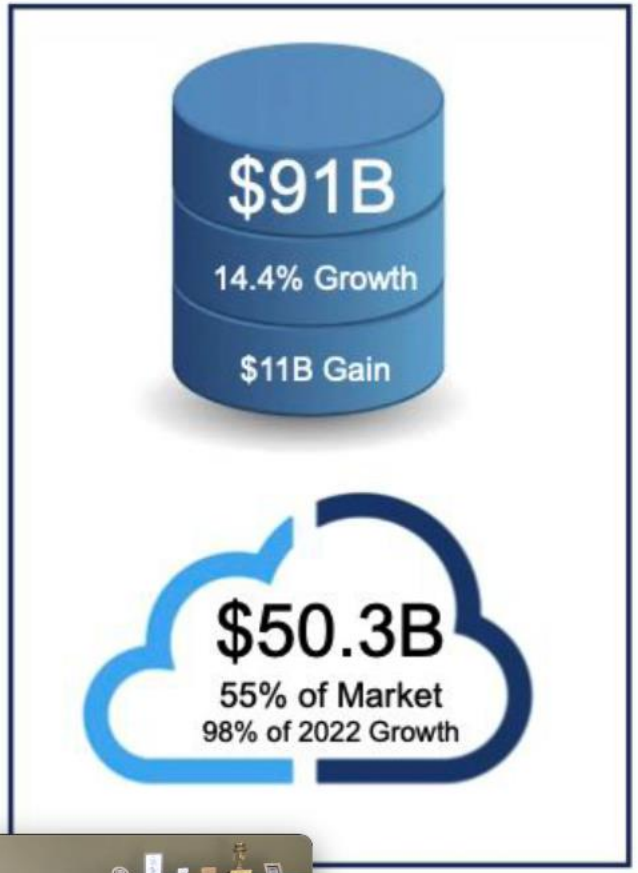
Popularity

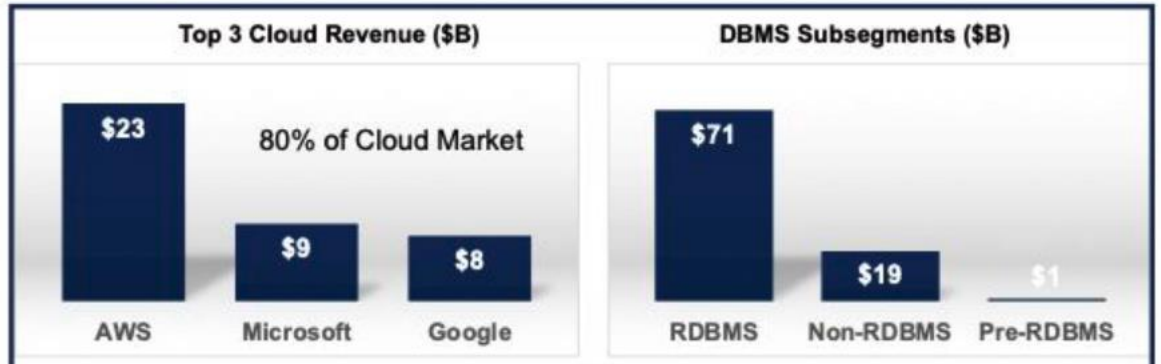
stackoverflow



STORMATICS

2022 DBMS Market Snapshot





Top 10 (Top 5 = 81% of Market)			Fastest Growing	
AWS	New #1!	$23,023	Snowflake	83.2%
Microsoft		$21,970	Cockroach Labs	81.5%
Oracle		$16,875	Databricks	76.3%
Google		$7,616	EnterpriseDB	58.3%
IBM		$4,587	TigerGraph	49.3%
SAP		$3,611	MongoDB	48.7%
Alibaba		$1,939	Google	46.2%
Huawei		$1,251	Tencent	41.3%
Snowflake	New!	$1,223	Redis	40.7%
MongoDB	New!	$1,205	Singlestore	40.6%

Gartner

STORMATICS

So - what do you do when you need to scale PostgreSQL in the cloud?

STORMATICS

Scale by credit card!

STORMATICS

Well, not really ...

You are only delaying the inevitable

STORMATICS

You tested your application here

STORMATICS

… and this is what production looks like

STORMATICS

If you find this button, please let me know!

Scaling PostgreSQL

Some tips for developers

- Query & SQL Optimization
- Performance Features
- Architectural Improvements
- Parameter Tuning

Scaling PostgreSQL

Some tips for developers

- **Query & SQL Optimization**
- Performance Features
- Architectural Improvements
- Parameter Tuning

STORMATICS

pg_stat_statement is your friend

- PostgreSQL extension, included in distribution
  - Provides a view in PostgreSQL
- Logs statistics about SQL statements
- Easy stats to watch out for
  - Long running (`mean_exec_time`)
  - Most frequent (`calls`)
  - Standard deviation in execution time (`stddev_exec_time`)
  - I/O intensive (`blk_read_time`, `blk_write_time`)

STORMATICS

What to watch out for

- pg_stat_statements is `off` by default
- The data is aggregated with time windows not available
- Small ~4% CPU overhead

STORMATICS

Some tooling for stats and visualization

- pgAdmin
- pgBadger
- Prometheus with Grafana
- Commercial tools
  - Datadog
  - DBeaver
  - New Relic

STORMATICS

EXPLAIN plan is your friend

- ‘Cost’ based query planner of PostgreSQL
  - Configuration parameters
  - Database statistics
- Key metrics in the EXPLAIN plan
  - Cost
  - Rows
  - Width
- Key outputs of the EXPLAIN plan
  - Scan
  - Join
  - Aggregate & grouping
  - Sort
- EXPLAIN vs EXPLAIN ANALYZE

STORMATICS

Example

We want the top 5 products by total sales in 2023 from a database with tables: orders, order_details, & products

```
SELECT p.product_name, SUM(od.quantity * od.unit_price) AS
total_sales
  FROM products p
  JOIN order_details od ON p.product_id = od.product_id
  JOIN orders o ON od.order_id = o.order_id
 WHERE o.order_date >= '2023-01-01' AND o.order_date <= '2023-12-31'
 GROUP BY p.product_name
 ORDER BY total_sales DESC
 LIMIT 5;
```

STORMATICS

Example

```
Limit  (cost=1000.43..1000.45 rows=5 width=40)
  ->  Sort  (cost=1000.43..1005.43 rows=2000 width=40)
        Sort Key: (sum((od.quantity * od.unit_price))) DESC
        ->  HashAggregate  (cost=950.00..975.00 rows=2000 width=40)
              Group Key: p.product_name
              ->  Hash Join  (cost=500.00..900.00 rows=10000 width=20)
                    Hash Cond: (od.product_id = p.product_id)
                    ->  Hash Join  (cost=250.00..600.00 rows=10000 width=16)
                          Hash Cond: (od.order_id = o.order_id)
                          ->  Seq Scan on order_details od  (cost=0.00..300.00 rows=10000 width=16)
                          ->  Hash  (cost=225.00..225.00 rows=2000 width=8)
                                ->  Seq Scan on orders o  (cost=0.00..225.00 rows=2000 width=8)
                                      Filter: (order_date >= '2023-01-01' AND order_date <= '2023-12-31')
                    ->  Hash  (cost=150.00..150.00 rows=5000 width=12)
                          ->  Seq Scan on products p  (cost=0.00..150.00 rows=5000 width=12)
```

STORMATICS

Helping the query planner do its job

- Up to date database statistics
  - Run ANALYZE periodically
- Accurate cost calculations
  - Tune database parameters

STORMATICS

Watch out for locks!

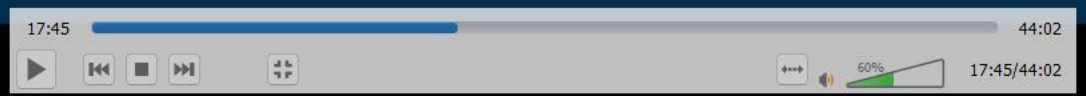
Session 1

```
BEGIN;

UPDATE foo SET … WHERE id = 1;

UPDATE foo SET … WHERE id = 2;

UPDATE foo SET … WHERE id = 3;

COMMIT;
```

Session 2

```
UPDATE foo SET … WHERE id = 1;
```

(waits)



STORMATICS

Query & SQL Optimization

Key takeaways:

- Monitor your queries
- Analyze the execution
- Code to avoid locks

Scaling PostgreSQL

Some tips for developers

- Query & SQL Optimization
- **Performance Features**
- Architectural Improvements
- Parameter Tuning

STORMATICS

Indexes

- B-Tree - *default index*
  - Default index that structure data into a balanced binary tree
  - Composite - *multi column*
    - Useful in cases where queries filter based on multiple columns
  - Partial - *conditional index on subset of data*
    - Defined by a conditional expression, making the index smaller in size
  - Covering - *includes an additional column*
    - Index-only lookup, created using an INCLUDE clause
- Hash - *equality checks*
  - Uses hash of the key for very fast equality access
  - Creates a hash table with O(1) complexity
- BRIN (block range index) - *space efficient for sorted tables*
  - Stores min and max values of a block only

STORMATICS

Indexes - Not a one-size-fits-all!

Don’t use indexes when:

- You need all or most of the data any ways
- Your workload is WRITE or UPDATE heavy with little READs
- Data bloat is a concern (‘over’ indexing)
- Your table is too small

STORMATICS

Many performance features ‘just work’

A few examples …

- Parallel queries
  - The query planner decides if it can use multiple CPU cores to execute a single query
  - There are tuning parameters that you can adjust
- Heap-Only Tuples (HOT)
  - Avoids index updates if changes don’t impact an indexed column
- Incremental sort
  - Don’t start from scratch, sort only what is not yet sorted
- Autovacuum
  - Gets rid of dead tuples to clear out the table bloat
  - [illegible]are tuning parameters that you can adjust

STORMATICS

Performance Features

Key takeaways:

- Indexes are a powerful ally
- … but you shouldn’t overuse them
- Let PostgreSQL do its job

Scaling PostgreSQL

Some tips for developers

- Query & SQL Optimization
- Performance Features
- **Architectural Improvements**
- Parameter Tuning

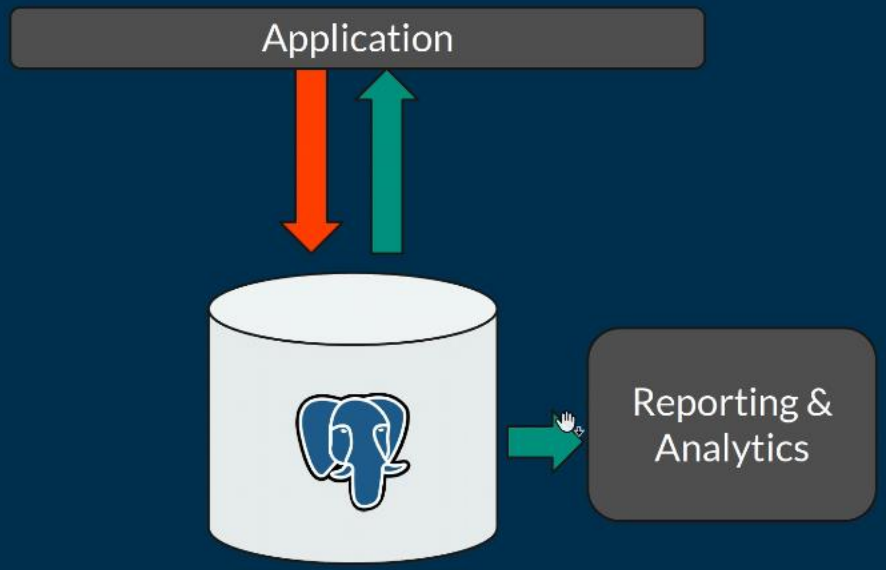
STORMATICS

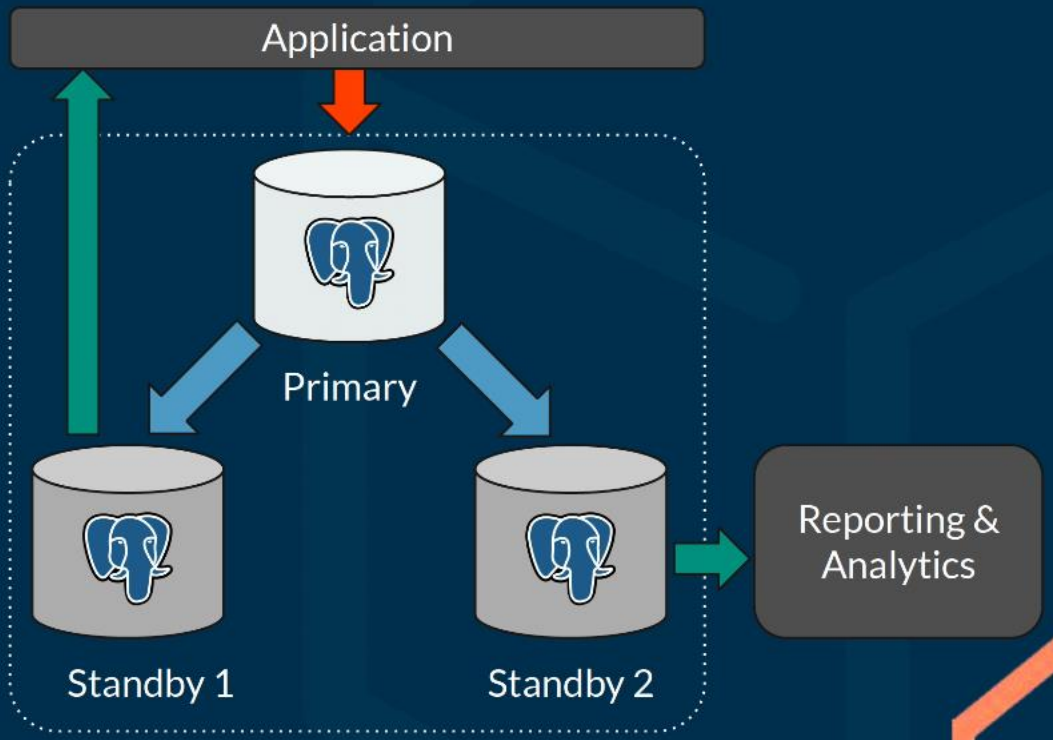
Load Balancing

- Distributing database queries across multiple servers to optimize resource use and improve performance
- Benefits
  - Prevents any single server from becoming a bottleneck.
  - Facilitates horizontal scaling by adding more replicas.
  - Enhances system's resilience against server failures.

STORMATICS

Load balancing





Write
Read
Replicate

STORMATICS

Load balancing

Single Node SELECTs

```
transaction type: <builtin: select only>
scaling factor: 10
query mode: simple
number of clients: 25
number of threads: 1
maximum number of tries: 1
duration: 60 s
number of transactions actually processed: 19139
number of failed transactions: 0 (0.000%)
latency average = 67.215 ms
initial connection time = 8620.897 ms
tps = 371.939402 (without initial connection time)
```

Load Balanced 3-node Cluster

```
transaction type: <builtin: select only>
scaling factor: 10
query mode: simple
number of clients: 25
number of threads: 1
maximum number of tries: 1
duration: 60 s
number of transactions actually processed: 24885
number of failed transactions: 0 (0.000%)
latency average = 51.449 ms
initial connection time = 8896.110 ms
tps = 485.918972 (without initial connection time)
```

+30%

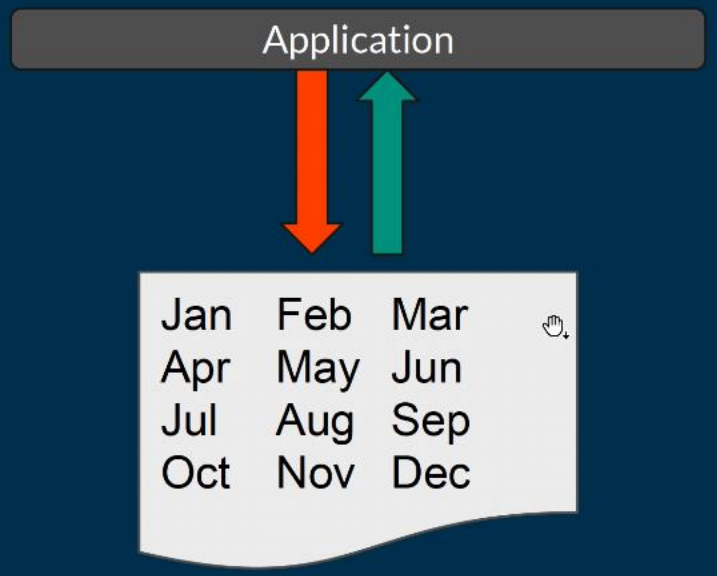
STORMATICS

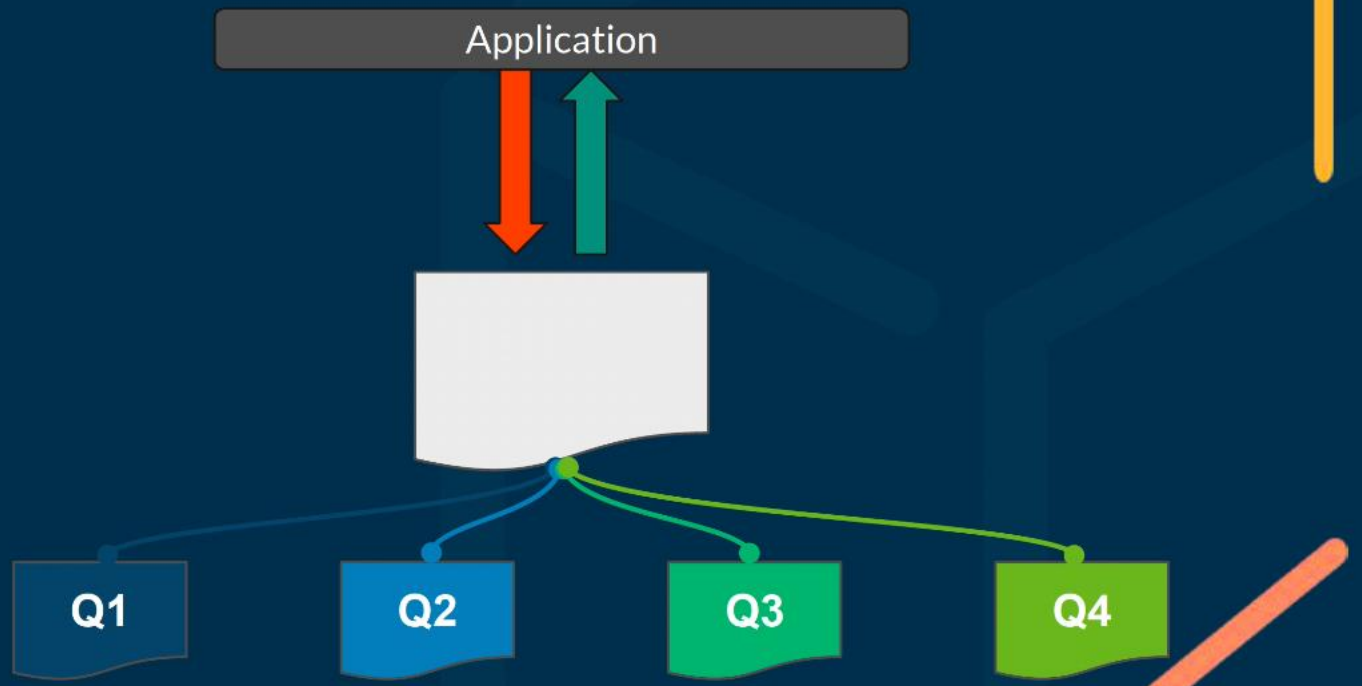
Partitioning

- Dividing a large table into smaller, more manageable pieces, called partitions, based on certain criteria (e.g., date ranges, geographic location).
- Benefits
  - Performance: Queries that access only a subset of data can run faster.
  - Maintenance: Operations like backups, deletes, and archiving can be performed on individual partitions.
  - Indexing: Smaller, partition-specific indexes are faster to update and search.

STORMATICS

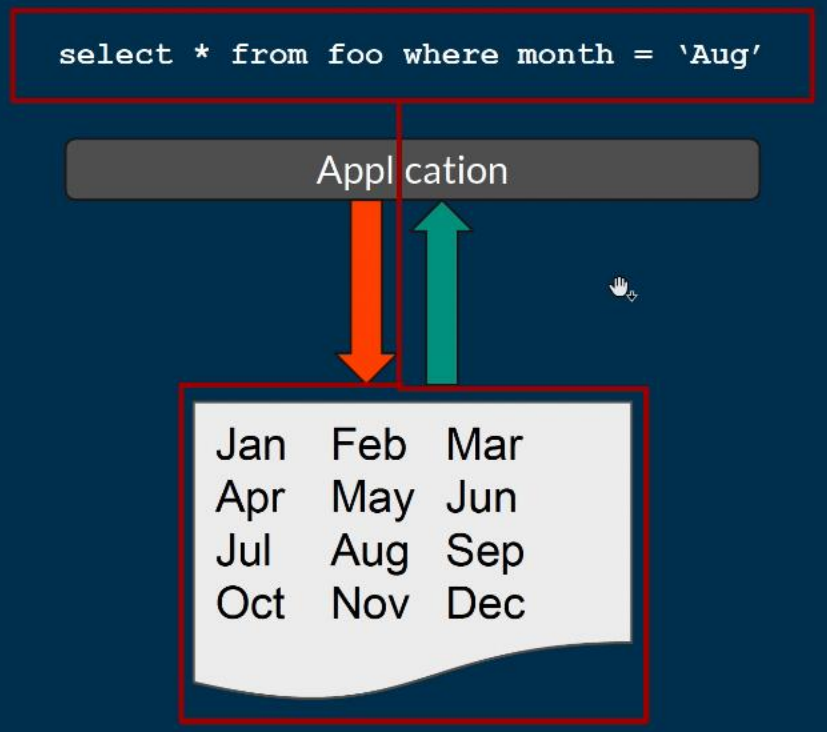
Partitioning

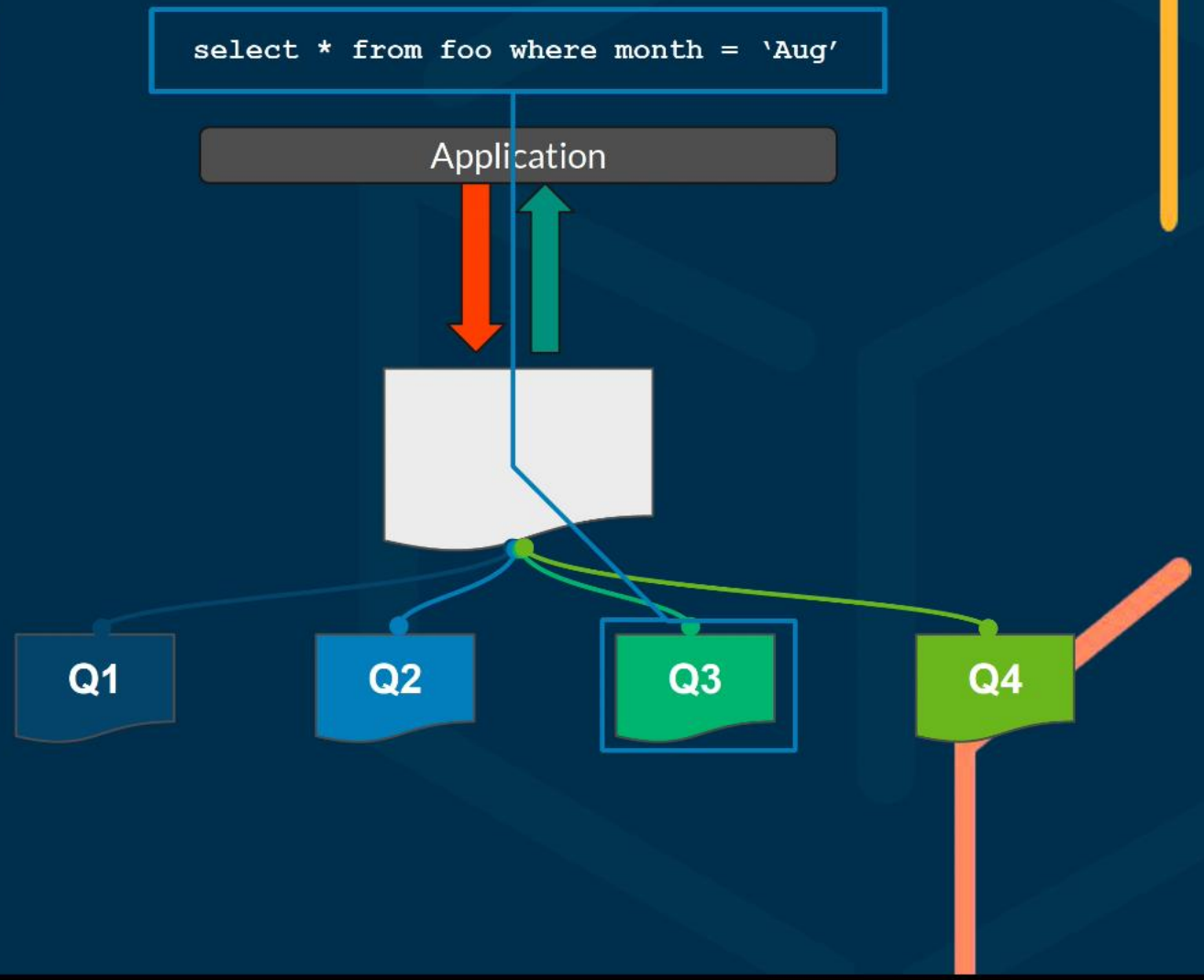




STORMATICS

Partitioning





STORMATICS

Architectural Improvements

Key takeaway:

- Don’t overload a single node!

Scaling PostgreSQL

Some tips for developers

- Query & SQL Optimization
- Performance Features
- Architectural Improvements
- **Parameter Tuning**

STORMATICS

Parameter tuning

- Two broad categories of parameters
  - Allocation parameters
  - Cost definitions

STORMATICS

Easily tuned database parameters - Allocation

- `shared_buffers`
  - Cache for frequently accessed data
  - Default is 128MB
  - Recommended is between 25% and 40% of system memory
- `wal_buffers`
  - Shared memory not yet written to disk
  - Default is 3% of shared_buffers
  - A value of up to 16MB can improve performance in high concurrency commits
- `work_mem`
  - Memory available for a query operation
  - Default is 4MB
  - High I/O activity for a query is an indicator that an increase in work_mem can help
  - Each parallel operation is allowed to use memory up to this value

STORMATICS

Easily tuned database parameters - Costs

- `cpu_tuple_cost`
  - Cost of processing a single row of data, including operations like WHERE and JOIN
  - Default is 0.01
  - Lower values encourage query planner to process more rows, helpful for I/O bound operations
  - Higher values encourage query planner to process less rows, helpful for CPU bound operations
- `random_page_cost`
  - Cost of non-sequential disk page access
  - Default is 4.0
  - Lower values imply low cost for random access, encouraging index scans
  - Higher values imply high cost for random access, encouraging sequential scans
- `effective_cache_size`
  - Expected size of database cache, including shared buffers and OS cache
  - Default is 4GB (this is not an allocation)
  - Higher values imply more data in cache, encouraging index scans
  - Lower values imply less data in cache, encouraging sequential scans

STORMATICS

Parameter Tuning

Key takeaways:

- Tweak configuration parameters based on your hardware and workload
- This will require some experimentation till you nail it down
- Makes an ideal candidate for AI-based tuning

STORMATICS

Conclusion

Database performance involves a lot of variables. Optimize how data is accessed before scaling by credit card!

Questions?

pg_umair

KEEP CALM AND CALL

STORMATICS