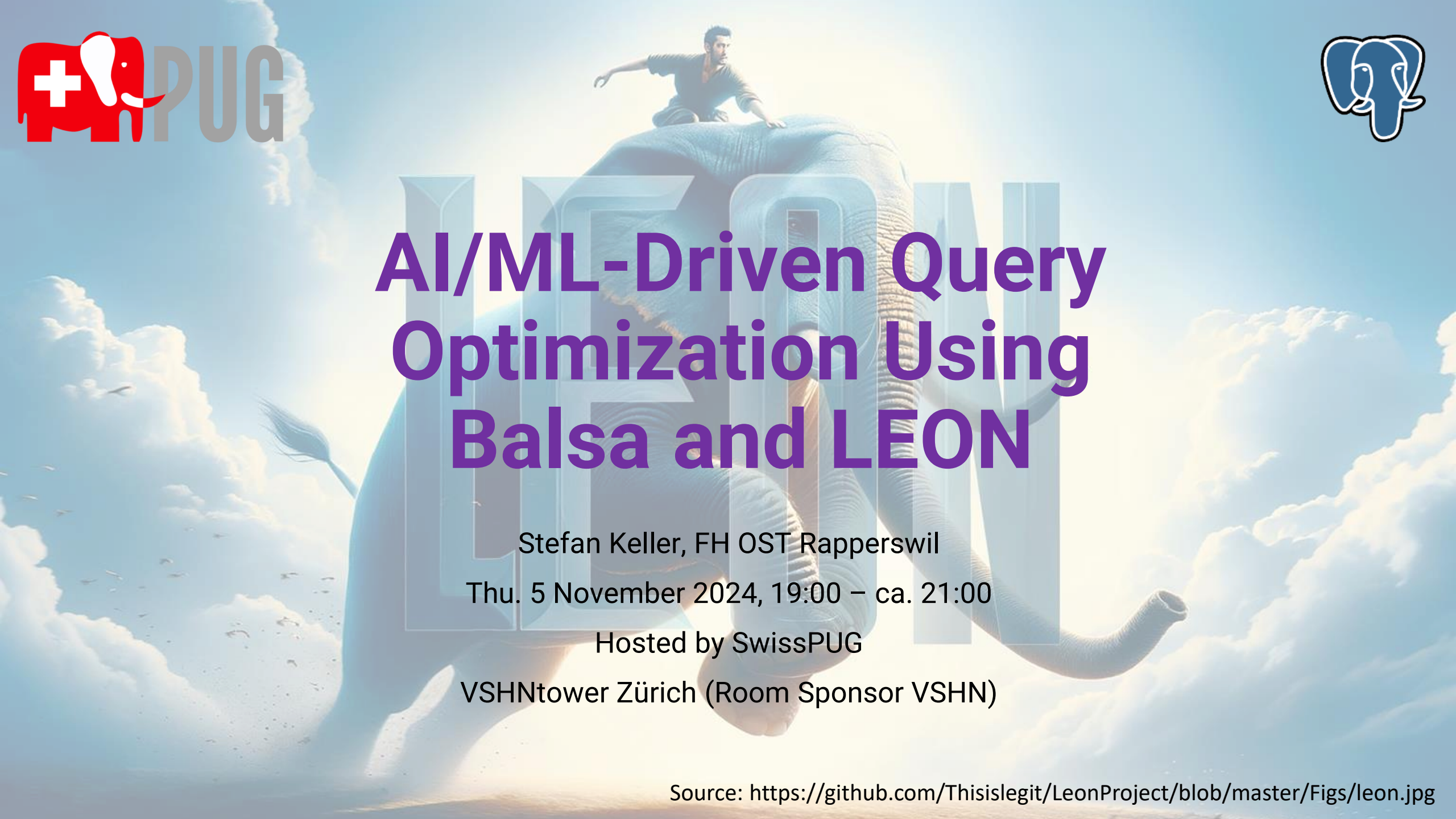


AI/ML-Driven Query Optimization Using Balsa and LEON

Stefan Keller, FH OST Rapperswil

Thu. 5 November 2024, 19:00 – ca. 21:00

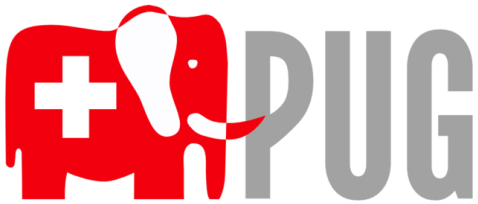
Hosted by SwissPUG

VSHNtower Zürich (Room Sponsor VSHN)

Source: https://github.com/Thisislegit/LeonProject/blob/master/Figs/leon.jpg

AI/ML-Driven Query Optimization Using Balsa and LEON

- Presented by
  - Prof. Stefan Keller, Leiter Institut für Software
  - Ostschweizer Fachhochschule, Campus Rapperswil
  - ost.ch/ifs
- Hosted by
  - Swiss PostgreSQL Users Group (SwissPUG)
  - swisspug.ch
- Sponsored by
  - VSHN AG Zurich Office
  - vshn.ch



Content Overview

AI/ML-Driven Query Optimization Using Balsa and LEON

1. What is Optimization and Tuning in Databases?
2. Optimization and Tuning Tools for PostgreSQL
3. Introduction to Query Optimization
4. LEON and Balsa
5. Comparison and Outlook

Q & A and “Ask-us-Anything”

This Presentation …

… is based on own preliminary short studies.

- The goals of this study were
  - to get to know the possibilities and limits of AI/ML query optimization with PostgreSQL,
  - esp. with regard to next seminar in our Computer and Data Science Master's 2025
  - (and also for my own learning about PostgreSQL and AI)
- The goal of this study was *not*
  - to focus on monitoring for db admins, nor database tuning in general,
  - nor to get the most out of the current PostgreSQL (e.g. with LEON and Balsa),
  - nor to be an exhaustive overview of the PG tools ecosystem (→ feedback welcome!)

The Problem

- No automation: DBMS tuning and query optimization traditionally require manual intervention and decades of expertise.
- Lack of DB statistics: Maintaining full-featured statistics in the system catalog on user data on an ongoing basis is impractical due to computational overhead.
- Unmatched defaults: PostgreSQL is a strong candidate for tuning because it comes configured "like a toaster controller".
- Complexity: As DBMSs evolve, it becomes increasingly difficult for human experts to anticipate their complexity.

The Need and the Opportunities

- The growing needs:
  - DB-as-a-Service (cloud) providers have amplified issues
  - PostgreSQL being the "database of choice" (Source: db-engines 2024, Stack Overflow Developer Survey 2024) the user base is growing

- The emerging opportunity:
  - Artficial Intelligence and Machine Learning (AI/ML) to implement AI/ML-based tuners and optimizers!

1. What is Optimization and Tuning in Databases?

What is optimization and tuning in Databases?

- Optimization is a subset of database (DBMS) tuning
  - While optimization focuses typically on queries, tuning addresses the performance of the entire database system, including hardware, software, and configuration settings.
- Optimization in databases
  - Process of improving query performance.
  - Uses algorithms to determine the most efficient way to execute queries (optimizer).
  - Minimizes response time, CPU usage, and I/O operations.
  - Includes techniques like index selection, query rewriting, and join optimization.
- Tuning in databases
  - Fine-tuning the database settings to enhance performance.
  - Involves adjusting parameters, memory allocation, and hardware settings.
  - Includes optimizing schema design and partitioning.
  - Ensures efficient resource utilization and scalability.

What are tuning options in PG?

10 options for tuning a database like PG - starting with the plain means of configuration and with options that don't require altering the schema or data:

- 1. Statistics Updating and Database Monitoring
  - Regularly vacuum tables and update statistics to optimize space usage and query planner performance.
    Tools: PostgreSQL's VACUUM, ANALYZE, autovacuum.
  - Monitor database performance and logs to identify slow queries and resource bottlenecks.
  - Tools: pg_stat_statements, pgBadger, pg_stat_activity. Extensions: PoWA.

What can be tuned in PostgreSQL?

- 2. Query Optimization:
  - Refactor inefficient queries, reduce complexity, and avoid “SELECT *”.
  - Just-in-time (JIT) compilation: Process of turning some form of interpreted program evaluation into a native program, at run-time.
  - Tools: EXPLAIN ANALYZE, pg_stat_statements.
- 3. Index Optimization:
  - Add indexes to frequently queried columns, especially for WHERE, JOIN, or ORDER BY clauses.
  - Tools: CREATE INDEX, pg_stat_user_indexes.
- 4. Database Configuration Tuning:
  - Adjust server parameters (with GUC or in postgresql.conf file) such as shared_buffers, work_mem, and maintenance_work_mem for performance tuning, effective_io_concurrency, parallel_query.
  - Tools: pgTune, pg_stat_activity, pg_stat_statements.

What can be tuned in PostgreSQL?

- 5. Planner Hints:
  - Use GUC to guide the optimizer for better query plans when automatic optimization isn't effective. Extend PostgreSQL to planner hints.
  - Tools: PostgreSQL's enable_* options (e.g., enable_seqscan, enable_indexscan).
  - Extensions: pg_hint_plan, DBtune, LEON.
- 6. Parallel Query Execution:
  - Enable and configure parallel query execution to distribute the workload across multiple CPU cores.
  - Tools: PostgreSQL's max_parallel_workers_per_gather, parallel_setup_cost.
- 7. Partitioning:
  - Divide large tables into smaller, more manageable parts.
  - Tools: Native partitioning in PostgreSQL

What can be tuned in PostgreSQL?

- 8. Connection Pooling:
  - Reuse database connections to reduce connection overhead, improving performance.
  - Tools: PgBouncer, pgpool-II.
- 9. Caching:
  - Store frequently queried data in memory to reduce load on the database.
  - Tools: PostgreSQL's built-in shared_buffers, pg_buffercache extension, external tools like pgCache or Redis.
- 10. Hardware Resource Scaling:
  - Improve hardware resources with more CPU, RAM, or faster storage (SSD).
  - Tools: None specific to PostgreSQL but relevant for infrastructure configuration.

Views, modules and extensions

PostgreSQL views, modules and extensions mentioned here:

- Views (from PostgreSQL system catalog):
  - pg_stats, pg_stat_activity, pg_stat_database.
- Contrib. Modules (as part of PostgreSQL delivery):
  - pg_stat_statements, pg_prewarm, …
- Extensions (as modules «external» to PostgreSQL delivery):
  - «Readers»: pg_qualstats, pg_linux_stats, pg_stat_kcache
  - «Writers»: pg_hint_plan

2. Optimization and Tuning Tools for PostgreSQL

Tuning tools for PostgreSQL («.com»)

- What are commercial tuning tools for PostgreSQL (with focus on tuning rather than query optimization)?
- EDB Postgres Tuner
  - This extension provides safe recommendations for PG settings (e.g. memory) that maximize the use of available resources (…?)
  - https://www.enterprisedb.com/docs/pg_extensions/pg_tuner/
- DBtune
  - AI-powered optimizer which tunes PG database configuration.
  - https://www.dbtune.com/
- OtterTune
  - A database tuning service for PG and MySQL start-up out of Carnegie Mellon University.
  - Ended 2024 saying "we got screwed over by a Private Equity PostgreSQL company on an acquisition offer.«
  - https://ottertune.com/

Tuning tools for PostgreSQL (FOSS)

- GPTuner:
  - A documentation-text-reading database tuning system using Bayesian Optimization guided by GPT models for PG and MySQL. No license indicated. Last commit Aug. 2024, 70 stars, 3 watching, 17 forks, written in Python. https://github.com/SolidLao/GPTuner
- UDO "Universal Database Optimization"
  - Utilizes Deep Reinforcement Learning to optimize transactions, index selections and database system parameters for PG and MySQL. MIT license. Last commit March 2024, 70 stars, 3 watching, 17 forks, written in Python. https://github.com/jxiw/UDO
- DB-BERT:
  - A documentation-text-reading AI-driven tool that utilizes Reinforcement Learning and GenAI to optimize PG configurations, enhancing performance by analyzing workload patterns. Apache License 2.0. Last commit Aug. 2023, 56 stars, 4 watching, 13 forks, written mainly in Python. https://github.com/itrummer/dbbert
- pg_tuner:
  - An automated PG database parameter tuning tool that utilizes Bayesian Optimization techniques. It simulates realistic workloads and captures comprehensive statistics to optimize database configurations. By Hironobu Suzuki; uses pg_linux_stats. No license (PostgreSQL license?). Last commit Aug. 2024, 37 stars, 1 watching, 1 fork, written in Python. https://github.com/s-hironobu/pg_tuner
- AutoDBA:
  - A - yet to be released - AI agent designed to manage PG databases by ensuring reliability, efficiency, scalability, and security. It connects to existing PostgreSQL databases and takes necessary actions to maintain optimal performance. Apache-2.0 license. Last commit Nov. 2024, 11 stars, 2 watching, 0 forks, written mainly in Go. https://github.com/crystaldb/autodba

Tuning tools for PostgreSQL (FOSS): Draft evaluation

1. GPTuner: 👍 Best overall for features and active maintenance, advanced optimization, but 👎 complex installation.
2. UDO: 👍 Comprehensive optimization features, well-documented, moderate community engagement, but 👎 complex installation.
3. DB-BERT: 👍 Innovative NLP-driven features, good maintenance, but 👎 requires complex installation.
4. pg_tuner: 👍Well-documented, stable and easiest to install, but 👎 lacks advanced features and community engagement.
5. AutoDBA: 👍 Promising goals, minimal dependencies, simple installation, some community engagement but 👎 currently monitoring only, lacking optimization (AI agent) and only working on AWS RDS PostgreSQL.

Optimization tools for PostgreSQL (FOSS)

- LEON:
  - An ML-based framework for enhancing database query optimization by exploring execution plan spaces. Uses pg_stat_statements. License PostgreSQL. Last commit May 2023, 11 stars, 2 watching, 1 fork, written in Python. https://github.com/Thisislegit/LeonProject
- AQO (Adaptive Query Optimizer):
  - A query optimizer for PG that improves plan generation based on past query performance. Uses pg_stat_statements. License AGPL 3.0. Last commit Oct. 2024. 432 stars, 22 watching, 43 forks, written in C and PL/pgSQL. https://github.com/postgrespro/aqo
- pg_plan_advsr:
  - Suggests improved PG query plans using learned optimizations. Uses pg_stat_statements, pg_hint_plan and pg_qualstats. No license (PostgreSQL?). Last commit: May 2024. 99 stars, 18 watching, 12 forks, written in C. https://github.com/ossc-db/pg_plan_advsr
- Not considered
  - IndexAdvisor (Greenplum), PgCuckoo, Peloton, AutoTune.
  - Log analysis tools, like PoWA «PostgreSQL Workload Analyzer»: License PostgreSQL. Uses pg_stat_statements, pg_qualstats, and optionally pg_stat_kcache and hypopg. Last commit Oct. 2024. 770 stars, 37 watching, 57 forks, written in Python. https://github.com/powa-team/powa or pgBadger (Perl) https://github.com/darold/pgbadger/

Optimization tools for PostgreSQL (FOSS): Draft evaluation

1. LEON: 👍 Promising features but 👎 lacks active development, community support and documentation.
2. AQO: 👍 Adaptive and useful for optimization, but 👎 requires familiarity with PG internals for effective use.
3. pg_plan_advsr: 👍 Simple and helpful for advising better query plans but 👎 low community involvement and sporadic updates.

Tuning and query optimization goals and respective open source tool availability

- Optimize resource usage of CPU, memory, and disk I/O.
  - High availability: GPTuner, pg_tuner and LEON (besides PGTune and PGConfigurator).
- Improve throughput/performance by increasing the number of transactions processed.
  - Medium availability: GPTuner and LEON, pg_plan_advsr.
- Reduce latency by minimizing response time for read/write queries.
  - Medium availability: GPTuner, DB-BERT and LEON, AQO.
- Improve overall system efficiency, including DB balancing.
  - Low availability: Besides AutoDBA for promised above goals (and besides PoWA for workload analysis).

3. Introduction to Query Optimization

What are Query Optimizers and where they are used?

- What is a Query Optimizer?
  - A query optimizer is a vital component of a database management system that automatically determines the most efficient way to execute a SQL query.
- What is Query Optimization?
  - Query optimization is the process of finding the best execution strategy—a plan or method to process the data—for a given SQL query, thereby making better use of available resources.

What are Query Optimizers and where they are used? (cont.)

- Where does this fit in?
  - This optimization process happens before any data is retrieved and results in an optimal query execution.
- How is this achieved?
  - It involves analyzing different query execution plans and choosing the most efficient one based on criteria like execution time, resource usage, data volume, index disk I/O.

How the Query Optimizer Works

- A Database Engine contains two major components:

- Storage engine
  - Reads the data between the disk and memory in a manner that optimizes concurrency while maintaining data integrity
- Query processor
  - accepts all queries submitted to SQL Server, devises a plan for their optimal execution, and then executes the plan and delivers the required results
  - the first job of the query processor is to devise a plan (best possible)
  - second job is to execute the query according to that plan

How the Query Optimizer Works

1. **Parsing and binding**

the query is parsed and bound

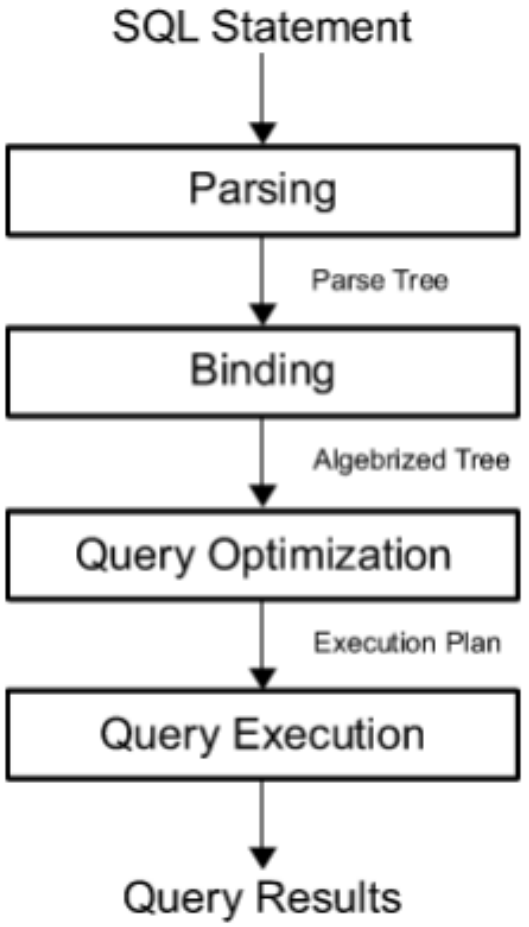
2. Query optimization

generation of possible execution plans

cost-assessment for each plan

3. Query execution, plan caching

query is executed by query engine



Generating Candidate Execution Plans

Search Space Definition

- the Query Optimizer defines all possible execution plans for a query

Transformation Rules and Heuristics

- candidate plans are generated using transformation rules and
- heuristics to reduce the number of possibilities.

Memo Storage

- plans are temporarily stored in the Memo structure during optimization

Assessing the Cost of Each Plan

Cardinality Estimation

- the optimizer estimates the number of records processed by each operator

Resource Estimation

- each operator’s cost in terms of CPU, I/O, and memory is calculated

Total Plan Cost

- the cost of all operators is aggregated to estimate the total plan cost

Query Execution and Plan Caching

Execution Engine

- the selected execution plan is passed to the execution engine for retrieving data

Plan Cache

- plans may be cached for reuse, avoiding the need for re-optimization

Plan Invalidations

- changes in database structure or significant data modifications can invalidate cached plans

Hinting

Directing Index Usage

- hints can force the Query Optimizer to use a specific index

Forcing Join Algorithms: Specific join algorithms (e.g., nested loop or hash join) can be enforced using hints

Providing a Plan: Users can provide a complete execution plan in XML format to be used directly.

4. LEON and Balsa

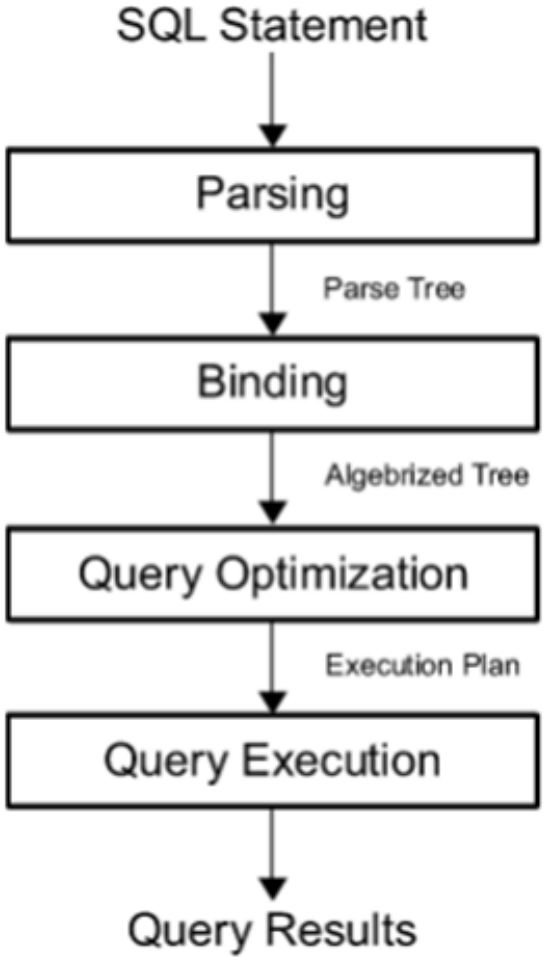
LEON

- “LEarned Optimizer for query plaNs” (LEON)
- A hybrid approach that combines machine learning-based techniques with “rule-based” expert knowledge
- Uses a contextual pairwise ranking objective instead of regression, focusing on ranking execution history
- Comes as PostgreSQL Extension, not pre-compiled
- Academic project based on Balsa (see next)
- Repo: https://github.com/Thisislegit/LeonProject

BALSA

- “Bayesian Active Learning for Self-Adaptation” (BALSA)
- Approach that uses deep reinforcement learning to optimize database queries without relying on expert-crafted optimizers
- Custom C Program using the pg_hint_plan extension; so you need the source code of PostgreSQL, then compile PostgreSQL with BALSA and pg_hint_plan with make.
- Academic project which uses PostgreSQL
- Balsa is based on Bao
- Repo: https://github.com/balsa-project/balsa

Where Balsa comes in?



Plugins: pg_hint_plan

	QUERY PLAN text
1	Bitmap Heap Scan on flight (cost=51.87..6611.44 rows=3458 width=12)
2	Recheck Cond: ((scheduled_departure >= '2023-08-12 00:00:00-05'::timestamp with time zone) AND (scheduled_departure <= '2023-08-13 00:(
3	-> Bitmap Index Scan on flight_scheduled_departure (cost=0.00..51.00 rows=3458 width=0)
4	Index Cond: ((scheduled_departure >= '2023-08-12 00:00:00-05'::timestamp with time zone) AND (scheduled_departure <= '2023-08-13 00:

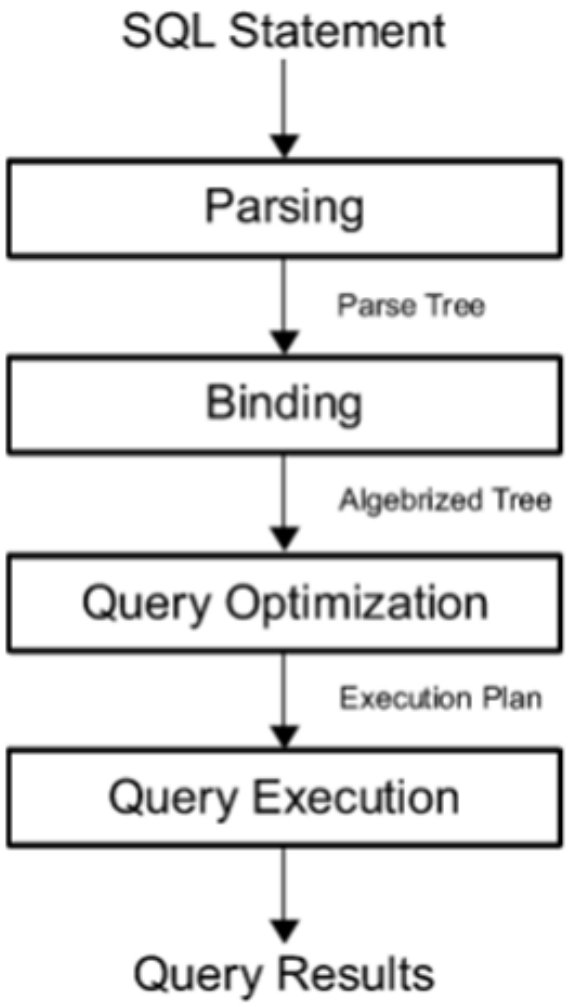
SQL

Plan

Latency

Model

Where LEON comes in?



1. Ranking of Plans
(keeping track of best plan)
2. Cost Function
(calibrates & corrects)

Plan → LEON

Extended Plan ← LEON

Logical Plan features exchange

Balsa Core Concept

- Leveraging reinforcement learning (RL)
  - Balsa leverages RL to optimize queries by interacting with the database and continuously learning from the outcomes of its actions
  - dynamic learning loop to evaluate for best strategy
- Learning without Experts
  - unlike traditional methods, Balsa learns to optimize queries without relying on existing expert optimizers
  - starts from no prior knowledge and gradually improves as it processes more queries
  - avoid inefficient query plans by receiving feedback in the form of execution latencies
  - this makes Balsa more adaptable than traditional optimizers and can lead to finding better execution paths without relying on fixed rules or human input

Balsa Core Concept (cont.)

- Adaptability to the DB usage
  - reinforcement learning enables the optimizer to handle unexpected workloads or changes in database performance
  - particularly effective for dynamic environments
  - continuous learning loop ensures that Balsa adapts to real-time conditions
  - allows it to generalize its learnings, improving its ability to optimize unseen queries as it develops a more complete understanding of the database

Balsa Core Concept (cont.)

- Avoiding Slow Plans
  - Balsa first learns in simulation, avoiding bad query plans without real execution
  - it refines its approach by running queries in the actual database, capturing system specifics
  - safe exploration ensures slow plans are limited using timeouts and execution caps
  - the system narrows its focus to the best plans over time
  - this two-phase learning makes Balsa efficient and safe while optimizing queries

Balsa Core Concept (cont.)

- Performance Against Expert Optimizers
  - Balsa matches or outperforms expert-built optimizers like PostgreSQL in just a few hours of training
  - on the JOB benchmark, it achieved up to 2.8x faster query execution
  - unlike human-built systems, Balsa learns autonomously, cutting down development time and cost
  - it adapts to new queries, improving its performance over time
  - this shows Balsa's potential to automate query optimization for diverse databases

Balsa Core Concept (cont. - fin)

- Future Impact of Learned Query Optimizers
  - Balsa demonstrates the potential for fully automated query optimizers
  - it could reduce the need for manual fine-tuning in new database systems
  - Balsa adapts to changing workloads, making it suitable for dynamic environments
  - future optimizers could adjust strategies based on real-time feedback
  - Balsa’s success may lead to more efficient, AI-driven optimizers in the future

LEON Core Concept

- ML-Aided Approaches
  - LEON leverages machine learning to aid traditional optimizers instead of replacing them
  - it combines ML techniques with the existing knowledge of expert optimizers to achieve efficient, self-adjusting query planning
  - LEON avoids the cold-start problem seen in ML-replaced approaches by starting with knowledge from traditional optimizers
  - this hybrid approach makes LEON both adaptive and practical for real-world database environments

LEON Core Concept (cont.)

- Aid, Not Replace, Expert Optimizers
  - ML methods alone struggle with the complexities of databases and cannot replace years of expert-crafted optimization logic
  - LEON uses ML to supplement traditional optimizers by improving aspects like cost estimation and plan search
  - traditional optimizers hold crucial domain knowledge, such as query transformation rules, that ML models cannot easily learn
  - LEON enhances the cost model by integrating ML where it performs best: adjusting performance for specific workloads
  - by combining both systems, LEON avoids the inefficiencies that occur when ML models operate in isolation

LEON Core Concept (cont.)

- Pairwise Ranking for Query Plans
  - uses pairwise ranking to compare query execution plans
  - focuses on ranking the relative performance of query plans rather than predicting their exact costs, reducing errors in prediction
  - this ranking approach improves the selection of execution plans by prioritizing the top-performing ones based on the database context
  - learns through feedback by ranking plan pairs, ensuring that only the most promising plans are explored during optimization
  - ranking system provides LEON with faster convergence and better accuracy compared to absolute value predictions

LEON Core Concept (cont.)

- Plan Exploration and Safe Pruning
  - introduces a robust exploration strategy to avoid getting stuck
  - it balances exploration and exploitation, focusing more on high-ranked plans while also exploring uncertain plans to correct errors
  - incorporates uncertainty-based exploration, using feedback to guide the optimizer towards better plans while
  - it safely prunes suboptimal plans during plan search by leveraging ML to reduce redundant calculations, improving optimization efficiency
  - this pruning ensures that only high-quality plans are retained, leading to faster execution times and reduced computation overhead

LEON Core Concept (cont.)

- LEON’s Performance vs. Traditional Optimizers
  - consistently outperforms traditional optimizers like PostgreSQL in latency performance, achieving up to 1.57x speedup
  - extensive testing shows LEON surpasses both ML-replaced methods and expert systems, especially in scenarios involving complex queries
  - the hybrid approach reduces query regression
  - maintains better overall stability, avoiding the performance fluctuations common in other learning-based optimizers
  - its ability to learn and adapt quickly ensures long-term performance gains even as workloads evolve

LEON Core Concept (cont.)

- Training Efficiency and Scalability
  - achieves significant improvements in training efficiency, often converging faster than other ML models or traditional optimizers
  - it leverages existing expert knowledge, allowing it to perform well even during the early stages of training with minimal data
  - compared to Balsa, LEON demonstrates faster convergence, lower variance, and superior performance after a few hours of training
  - adaptability is evident in dynamic workloads, where it adjusts seamlessly to new queries and workloads
  - this makes LEON ideal for environments where database queries and workloads are constantly changing

LEON Core Concept (cont.)

- Future Impact of ML-Aided Optimizers
  - hybrid approach could become a new standard for database systems
  - its success suggests that future query optimizers will likely continue to blend traditional expertise with machine learning for better results
  - the ML-aided approach ensures that optimizers can adjust to specific datasets and workloads without constant manual fine-tuning
  - this could lead to broader adoption in systems like PostgreSQL, where traditional optimizers can benefit from self-adjusting capabilities
  - the future of query optimization will likely involve deeper integration of ML to enhance, rather than replace, existing database tools

5. Comparison and Outlook

Comparison: Balsa vs. LEON

Feature/Aspect	Balsa	LEON
Approach	Fully RL-based optimizer	ML-aided optimizer with expert knowledge
Optimization Focus	End-to-end replacement	Hybrid approach aiding traditional system
Learning Objective	Regression-based cost estimation	Pairwise ranking of plans
Plan Exploration	Explores all possible plans	Focuses on top-ranked and uncertain plans
Pruning	No specific pruning strategy	ML-guided pruning for efficiency
Cold-Start Problem	Learns from scratch	Starts with expert knowledge
Performance	High variance, slow convergence	Stable, fast convergence, fewer regressions
Fallback Mechanism	No fallback	Falls back to expert optimizer when ML fails

Are AI/ML-based Optimizers a Solution?

- Remember: The problems of complexity and uncertainty remain:
  - There's no absolute "best plan" possible because exhaustive search is impractical for real-world databases-aside from the lack of statistics.
- Enter ML-based optimizers - but
  - Overhead / performance problem: Integrating ML into query optimizers introduces overhead in terms of plan evaluation and model training.
  - Lack of maturity (training data): We requires more training data.
  - Lack of reliability: Current ML-based optimizers suffer from "graceful degradation" - the inability to indicate when they are failing.
  - Lack of integration to be used “out-of-the box”

Evolution of Query Optimization

- Past(?)
  - Manual query optimization
- Current situation
  - Query monitoring including visualization
  - Heuristic, hard-coded optimization generators
  - Hybrid optimizers, combining ML with expert systems, towards self-adjusting optimization
  - Optimizers preferring relative ranking plans over predicting exact costs
- Near future
  - Adapting to dynamic workloads: Real-time learning to handle changing query patterns
  - Smart pruning: Efficient search space exploration to minimize overhead
  - Stability and convergence: More stable optimizers with faster learning
- Further Future
  - Seamless integration: Enhancing ML-based optimizers within systems
  - Self-improving, autonomous query optimization

Discussion

Final thought:

- Lack of cooperation of DB-as-a-Service providers and experts?
- “Shame on anyone who thinks evil of it”: Do they profit from lack of optimization?

Next events:

- 26+27 Juni 2025: Swiss PGDay 2025 (en+de), Rapperswil. www.pgday.ch
- Others: see SwissPUG, www.swisspug.ch

APPENDIX: PostgreSQL params. for tuning

max_connections
Too many connections can overload resources, tuning this optimizes resource allocation.
shared_buffers
Larger buffers reduce disk I/O by caching frequently accessed data, speeding up queries.
effective_cache_size
Informs the planner of available cache, improving query execution plans by reducing disk reads.
maintenance_work_mem
More memory here speeds up vacuum and index rebuilding, improving database upkeep.
checkpoint_completion_target
Spreads checkpoint activity to reduce I/O spikes, smoother performance during heavy loads.
wal_buffers
Larger buffers improve write performance by reducing disk writes for WAL logs.
default_statistics_target
Better statistics improve query planning, leading to more efficient execution strategies.
random_page_cost
Lowering this value can speed up queries that rely on index scans by reducing cost of random I/O.

APPENDIX: PostgreSQL params. for tuning (cont.)

effective_io_concurrency
Higher values allow more concurrent I/O operations, improving performance on SSDs.
work_mem
Allocating more memory per query operation reduces disk sorting, improving query execution speed.
huge_pages
Using larger memory pages reduces overhead for large memory allocations, improving memory performance.
min_wal_size
A higher value reduces the frequency of WAL file recycling, improving write performance.
max_wal_size
Increases WAL retention, reducing checkpoint frequency, and improving write-intensive performance.
max_worker_processes
More worker processes enable background jobs to run concurrently, boosting throughput.
max_parallel_workers_per_gather
Increases the number of workers for parallel queries, speeding up large data retrievals.
max_parallel_workers
Controls the total number of parallel workers across all queries, improving overall parallel query performance.
max_parallel_maintenance_workers
Allows more workers for maintenance tasks like vacuuming, improving maintenance efficiency.

APPENDIX: LEON postgres.conf

Component	LEON
Hardware	Multiple PostgreSQL instances, high memory requirements for local machine instances
Operating System	Not specified
PostgreSQL Version	Modified PostgreSQL 14.5
PostgreSQL Configuration	Custom configuration via GUC: `enable_leon`, `not_cali`, `leon_port`, `leon_host`

For PostgreSQL configuration see: leon-postgres.conf

APPENDIX: Balsa postgres.conf

Component	Balsa
Hardware	Microsoft Azure VMs with 8 cores, 64GB RAM, and SSDs
Operating System	Not specified
PostgreSQL Version	PostgreSQL 12.5
PostgreSQL Configuration	32GB shared buffers, 4GB work memory, GEQO disabled, cache size of 32GB

For PostgreSQL Config see: balsa-postgres.conf