Carnegie Mellon University

Lecture #16

ADVANCED DATABASE SYSTEMS

Optimizer Implementation (Part II)

@Andy_Pavlo // 15-721 // Spring 2018

DATABASE TALK

Striim Streaming Platform

→ Today @ 4:30pm

→ GHC 8102

http://db.cs.cmu.edu/events/db-seminar-spring-2018-alok-pareek-striim/

TODAY'S AGENDA

Cascades / Columbia
Orca Optimizer
MemSQL Optimizer
Extra Credit Assignment

QUERY OPTIMIZATION STRATEGIES

Choice #1: Heuristics
→ INGRES, Oracle (until mid 1990s)

Choice #2: Heuristics + Cost-based Join Search
→ System R, early IBM DB2, most open-source DBMSs

Choice #3: Randomized Search
→ Academics in the 1980s, current Postgres

Choice #4: Stratified Search
→ IBM's STARBURST (late 1980s), now IBM DB2 + Oracle

Choice #5: Unified Search
→ Volcano/Cascades in 1990s, now MSSQL + Greenplum

POSTGRES OPTIMIZER

Imposes a rigid workflow for query optimization:
→ First stage performs initial rewriting with heuristics
→ It then executes a cost-based search to find optimal join ordering.
→ Everything else is treated as an "add-on".
→ Then recursively descends into sub-queries.

Difficult to modify or extend because the ordering has to be preserved.

OPTIMIZER GENERATORS

Framework to allow a DBMS implementer to write the declarative rules for optimizing queries.

→ Separate the search strategy from the data model.

→ Separate the transformation rules and logical operators from physical rules and physical operators.

Implementation can be independent of the optimizer's search strategy.

Examples: Starburst, Exodus, Volcano, Cascades, OPT++

STRATIFIED SEARCH

First rewrite the logical query plan using transformation rules.

→ The engine checks whether the transformation is allowed before it can be applied.

→ Cost is never considered in this step.

Then perform a cost-based search to map the logical plan to a physical plan.

UNIFIED SEARCH

Unify the notion of both logical→logical and logical→physical transformations.

→ No need for separate stages because everything is transformations.

This approach generates a lot more transformations so it makes heavy use of memoization to reduce redundant work.

TOP-DOWN VS. BOTTOM-UP

Top-down Optimization

→ Start with the final outcome that you want, and then work down the tree to find the optimal plan that gets you to that goal.

→ Example: Volcano, Cascades

Bottom-up Optimization

→ Start with nothing and then build up the plan to get to the final outcome that you want.

→ Examples: System R, Starburst

CASCADES OPTIMIZER

Object-oriented implementation of the Volcano query optimizer.

Simplistic expression re-writing can be through a direct mapping function rather than an exhaustive search.

Graefe

THE CASCADES FRAMEWORK FOR QUERY OPTIMIZATION
IEEE Data Engineering Bulletin 1995

CASCADES OPTIMIZER

Optimization tasks as data structures.

Rules to place property enforcers.

Ordering of moves by promise.

Predicates as logical/physical operators.

EFFICIENCY IN THE COLUMBIA DATABASE QUERY OPTIMIZER
Portland State University MS Thesis 1998

CASCADES – EXPRESSIONS

A **expression** is an operator with zero or more input expressions.

Logical Expression: $(A \bowtie B) \bowtie C$

Physical Expression: $(A_F \bowtie_{HJ} B_F) \bowtie_{NLJ} C_F$

CASCADES – GROUPS

A **group** is a set of logically equivalent logical and physical expressions that produce the same output.

→ All logical forms of an expression

→ All physical expressions that can be derived from selecting the allowable physical operators for the corresponding logical forms.

Output:	Logical Exps	Physical Exps
[ABC]	1. $(A \bowtie B) \bowtie C$	1. $(A_F \bowtie_L B_F) \bowtie_L C_F$
	2. $(B \bowtie C) \bowtie A$	2. $(B_F \bowtie_L C_F) \bowtie_L A_F$
	3. $(A \bowtie C) \bowtie B$	3. $(A_F \bowtie_L C_F) \bowtie_L B_F$
	4. $A \bowtie (B \bowtie C)$	4. $A_F \bowtie_L (C_F \bowtie_L B_F)$
	⋮	⋮

CASCADES – GROUPS

A **group** is a set of logically equivalent logical and physical expressions that produce the same output.
→ All logical forms of an expression
→ All physical expressions that can be derived from selecting the allowable physical operators for the corresponding logical forms.

Group

Output: [ABC]	Logical Exps	Physical Exps
	1. $(A \bowtie B) \bowtie C$	1. $(A_F \bowtie_L B_F) \bowtie_L C_F$
	2. $(B \bowtie C) \bowtie A$	2. $(B_F \bowtie_L C_F) \bowtie_L A_F$
	3. $(A \bowtie C) \bowtie B$	3. $(A_F \bowtie_L C_F) \bowtie_L B_F$
	4. $A \bowtie (B \bowtie C)$	4. $A_F \bowtie_L (C_F \bowtie_L B_F)$
	⋮	⋮

CASCADES – GROUPS

A **group** is a set of logically equivalent logical and physical expressions that produce the same output.

→ All logical forms of an expression

→ All physical expressions that can be derived from selecting the allowable physical operators for the corresponding logical forms.

Group

Output:	Logical Exps	Physical Exps
[ABC]	1. $(A \bowtie B) \bowtie C$	1. $(A_F \bowtie_L B_F) \bowtie_L C_F$
	2. $(B \bowtie C) \bowtie A$	2. $(B_F \bowtie_L C_F) \bowtie_L A_F$
	3. $(A \bowtie C) \bowtie B$	3. $(A_F \bowtie_L C_F) \bowtie_L B_F$
	4. $A \bowtie (B \bowtie C)$	4. $A_F \bowtie_L (C_F \bowtie_L B_F)$
	⋮	⋮

Equivalent Expressions

CASCADES – MULTI-EXPRESSION

Instead of explicitly instantiating all possible expressions in a group, the optimizer implicitly represents redundant expressions in a group as a **multi-expression**.

→ This reduces the number of transformations, storage overhead, and repeated cost estimations.

Output: [ABC]	Logical Multi-Exps	Physical Multi-Exps
	1. $[AB] \bowtie [C]$	1. $[AB] \bowtie_L [C]$
	2. $[BC] \bowtie [A]$	2. $[BC] \bowtie_L [A]$
	3. $[AC] \bowtie [B]$	3. $[AC] \bowtie_L [B]$
	4. $[A] \bowtie [BC]$	4. $[A] \bowtie_L [CB]$
	⋮	⋮

CASCADES – RULES

A **rule** is a transformation of an expression to a logically equivalent expression.

→ **Transformation Rule:** Logical to Logical

→ **Implementation Rule:** Logical to Physical

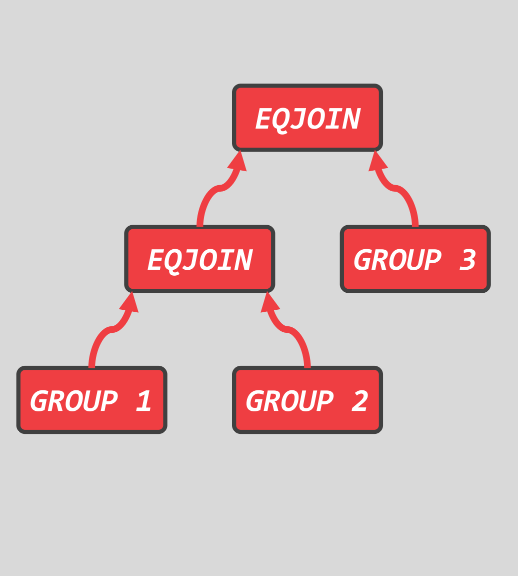
Each rule is represented as a pair of attributes:

→ **Pattern**: Defines the structure of the logical expression that can be applied to the rule.

→ **Substitute**: Defines the structure of the result after applying the rule.

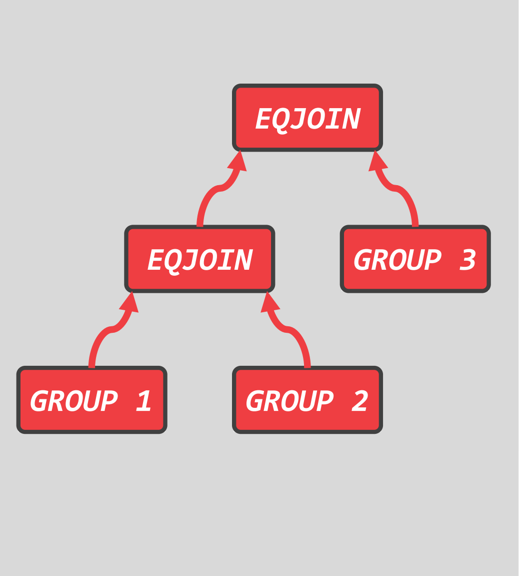
CASCADES – RULES

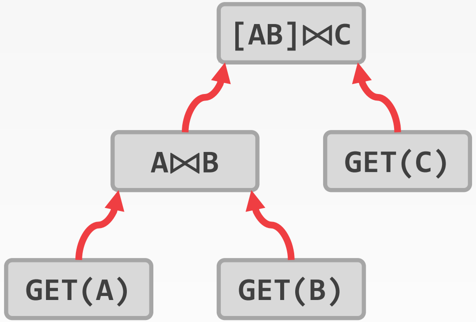
Pattern



CASCADES – RULES

Pattern

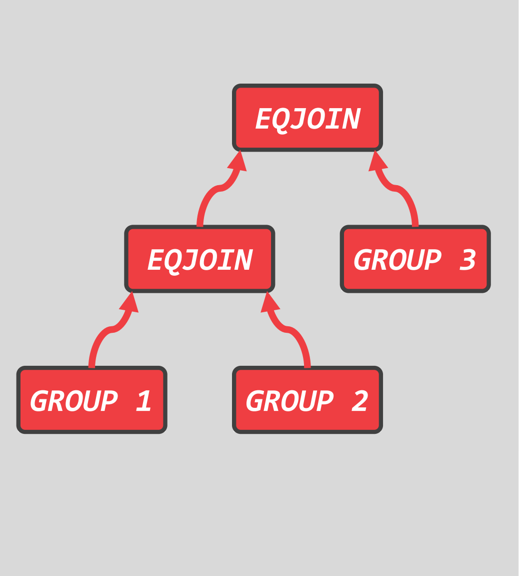


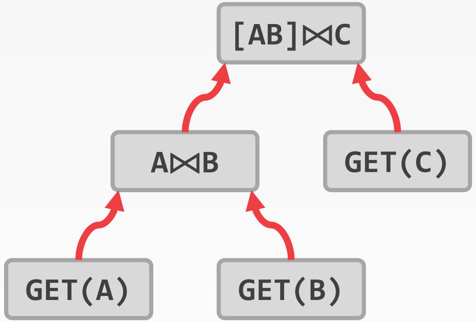


Matching Plan

CASCADES – RULES

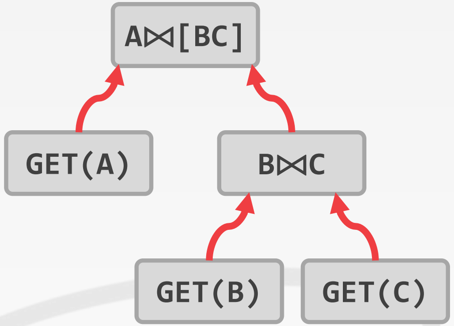
Pattern





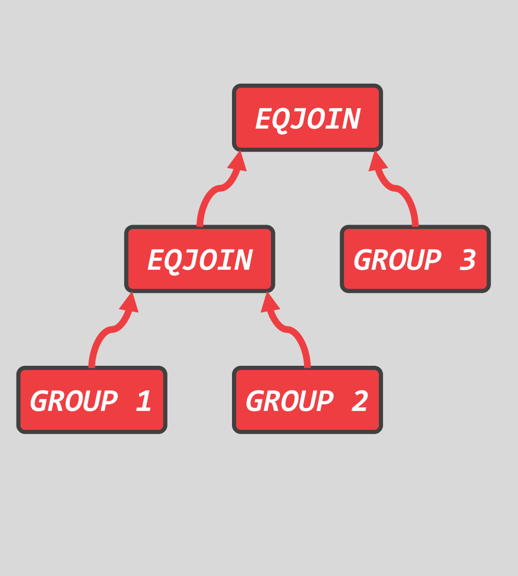
Matching Plan

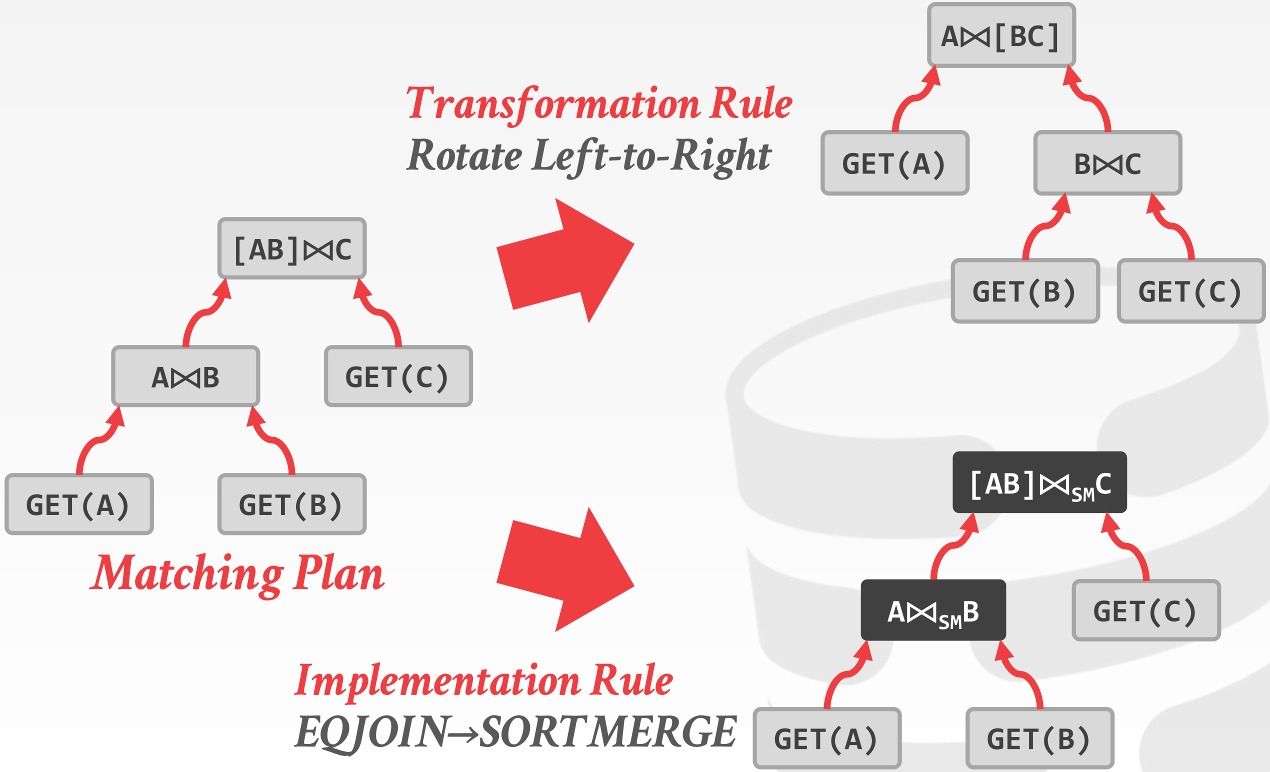
Transformation Rule
Rotate Left-to-Right



CASCADES – RULES

Pattern





CASCADES – MEMO TABLE

Stores all previously explored alternatives in a compact graph structure.

Equivalent operator trees and their corresponding plans are stored together in groups.

Provides memoization, duplicate detection, and property + cost management.

PRINCIPLE OF OPTIMALITY

Every sub-plan of an optimal plan is itself optimal.

This allows the optimizer to restrict the search space to a smaller set of expressions.

→ The optimizer never has to consider a plan containing sub-plan **P1** that has a greater cost than equivalent plan **P2** with the same physical properties.

EXPLOITING UPPER AND LOWER BOUNDS IN TOP-DOWN QUERY OPTIMIZATION
IDEAS 2001

CASCADES – MEMO TABLE

	Winner
[ABC]	
[AB]	
[A]	
[B]	
[C]	

	Logical M-Exps	Physical M-Exps
Output: [ABC]	1. [AB]⋈[C]	

	Logical M-Exps	Physical M-Exps
Output: [AB]	1. [A]⋈[B]	

	Logical M-Exps	Physical M-Exps
Output: [C]	1. GET(C)	

	Logical M-Exps	Physical M-Exps
Output: [A]	1. GET(A)	

	Logical M-Exps	Physical M-Exps
Output: [B]	1. GET(B)	

CASCADES – MEMO TABLE

	Winner
[ABC]	
[AB]	
[A]	
[B]	
[C]	

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B]	

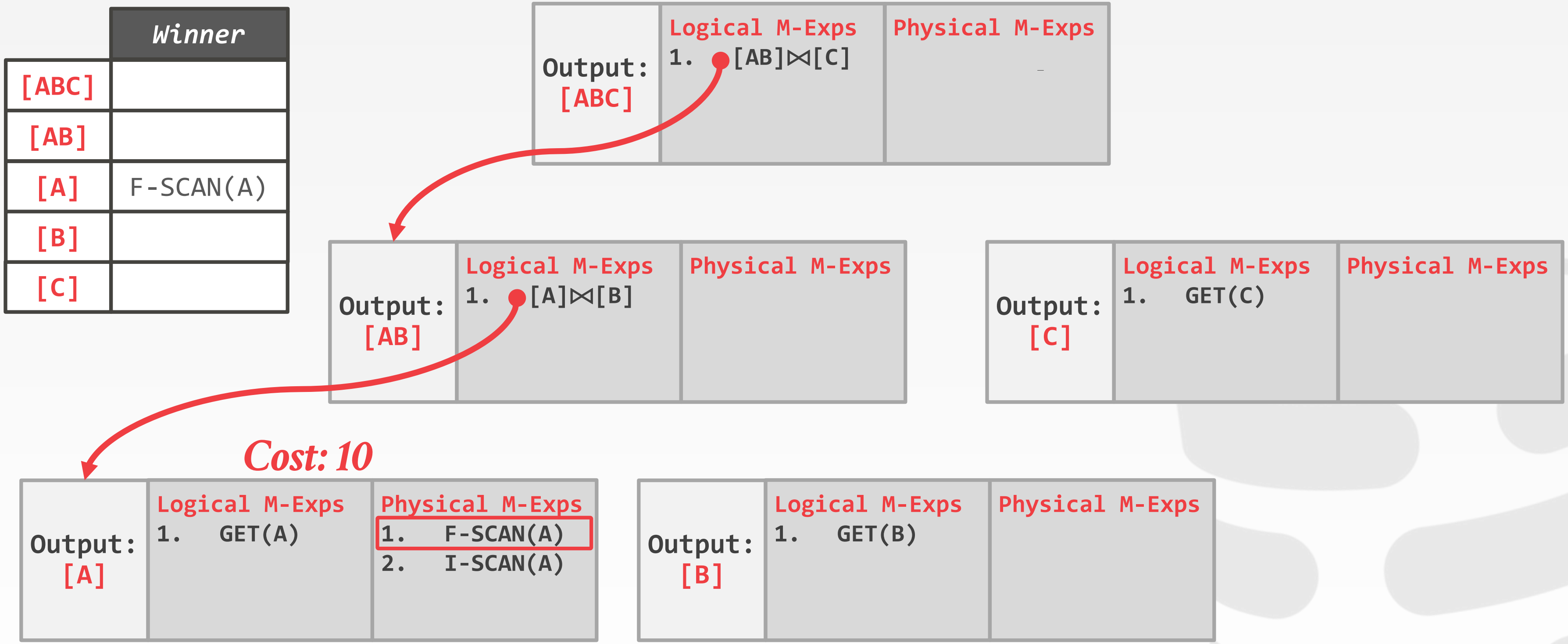
Output: [C]	Logical M-Exps	Physical M-Exps
	1. GET(C)	

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. F-SCAN(A) 2. I-SCAN(A)

Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	

CASCADES – MEMO TABLE

	Winner
[ABC]	
[AB]	
[A]	F-SCAN(A)
[B]	
[C]	



CASCADES – MEMO TABLE

	Winner
[ABC]	
[AB]	
[A]	F-SCAN(A)
[B]	
[C]	

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B]	

Output: [C]	Logical M-Exps	Physical M-Exps
	1. GET(C)	

Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. F-SCAN(A)
		2. I-SCAN(A)

Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	1. F-SCAN(B)
		2. I-SCAN(B)

CASCADES – MEMO TABLE

	Winner
[ABC]	
[AB]	
[A]	F-SCAN(A)
[B]	F-SCAN(B)
[C]	

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B]	

Output: [C]	Logical M-Exps	Physical M-Exps
	1. GET(C)	

Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	$\boxed{\text{1. F-SCAN(A)}}$
		2. I-SCAN(A)

Cost: 20

Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	$\boxed{\text{1. F-SCAN(B)}}$
		2. I-SCAN(B)

CASCADES – MEMO TABLE

	Winner
[ABC]	
[AB]	
[A]	F-SCAN(A)
[B]	F-SCAN(B)
[C]	

Output:	Logical M-Exps	Physical M-Exps
[ABC]	1. [AB]⋈[C]	

Output:	Logical M-Exps	Physical M-Exps
[AB]	1. [A]⋈[B] 2. [B]⋈[A]	

Output:	Logical M-Exps	Physical M-Exps
[C]	1. GET(C)	

Cost: 10

Output:	Logical M-Exps	Physical M-Exps
[A]	1. GET(A)	1. F-SCAN(A) 2. I-SCAN(A)

Cost: 20

Output:	Logical M-Exps	Physical M-Exps
[B]	1. GET(B)	1. F-SCAN(B) 2. I-SCAN(B)

CASCADES – MEMO TABLE

	Winner
[ABC]	
[AB]	
[A]	F-SCAN(A)
[B]	F-SCAN(B)
[C]	

Output: [ABC]
Logical M-Exps
1. [AB]$\bowtie$[C]

Physical M-Exps

Output: [AB]
Logical M-Exps
1. [A]$\bowtie$[B]
2. [B]$\bowtie$[A]

Physical M-Exps

Output: [C]
Logical M-Exps
1. GET(C)

Physical M-Exps

Cost: 10

Output: [A]
Logical M-Exps
1. GET(A)

Physical M-Exps
1. $\boxed{\text{F-SCAN(A)}}$
2. I-SCAN(A)

Cost: 20

Output: [B]
Logical M-Exps
1. GET(B)

Physical M-Exps
1. $\boxed{\text{F-SCAN(B)}}$
2. I-SCAN(B)

CASCADES – MEMO TABLE

	Winner
[ABC]	
[AB]	
[A]	F-SCAN(A)
[B]	F-SCAN(B)
[C]	

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B] 2. [B]⋈[A]	

Output: [C]	Logical M-Exps	Physical M-Exps
	1. GET(C)	

Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	$\boxed{\text{1. F-SCAN(A)}}$ 2. I-SCAN(A)

Cost: 20

Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	$\boxed{\text{1. F-SCAN(B)}}$ 2. I-SCAN(B)

CASCADES – MEMO TABLE

	Winner
[ABC]	
[AB]	
[A]	F-SCAN(A)
[B]	F-SCAN(B)
[C]	

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B]	1. $[A]\bowtie_{L}[B]$
	2. [B]⋈[A]	2. $[A]\bowtie_{SM}[B]$
		3. $[B]\bowtie_{L}[A]$

Output: [C]	Logical M-Exps	Physical M-Exps
	1. GET(C)	

Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	$\boxed{\text{1. F-SCAN(A)}}$
		2. I-SCAN(A)

Cost: 20

Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	$\boxed{\text{1. F-SCAN(B)}}$
		2. I-SCAN(B)

CASCADES – MEMO TABLE

	Winner
[ABC]	
[AB]	$[A]\bowtie_{SM}[B]$
[A]	F-SCAN(A)
[B]	F-SCAN(B)
[C]	

Output:	Logical M-Exps	Physical M-Exps
[ABC]	1. $[AB]\bowtie[C]$	

Cost: 50+(10+20)

Output:	Logical M-Exps	Physical M-Exps
[AB]	1. $[A]\bowtie[B]$ 2. $[B]\bowtie[A]$	1. $[A]\bowtie_{L}[B]$ 2. $\boxed{[A]\bowtie_{SM}[B]}$ 3. $[B]\bowtie_{L}[A]$

Output:	Logical M-Exps	Physical M-Exps
[C]	1. GET(C)	

Cost: 10

Output:	Logical M-Exps	Physical M-Exps
[A]	1. GET(A)	1. $\boxed{\text{F-SCAN(A)}}$ 2. I-SCAN(A)

Cost: 20

Output:	Logical M-Exps	Physical M-Exps
[B]	1. GET(B)	1. $\boxed{\text{F-SCAN(B)}}$ 2. I-SCAN(B)

CASCADES – MEMO TABLE

	Winner
[ABC]	
[AB]	[A]$\bowtie_{SM}$[B]
[A]	F-SCAN(A)
[B]	F-SCAN(B)
[C]	I-SCAN(C)

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]$\bowtie$[C]	

Cost: 50+(10+20)

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]$\bowtie$[B]	1. [A]$\bowtie_{L}$[B]
	2. [B]$\bowtie$[A]	2. [A]$\bowtie_{SM}$[B]
		3. [B]$\bowtie_{L}$[A]

Cost: 5

Output: [C]	Logical M-Exps	Physical M-Exps
	1. GET(C)	1. F-SCAN(C)
		2. I-SCAN(C)

Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. F-SCAN(A)
		2. I-SCAN(A)

Cost: 20

Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	1. F-SCAN(B)
		2. I-SCAN(B)

CASCADES – MEMO TABLE

	Winner
[ABC]	
[AB]	$[A]\bowtie_{SM}[B]$
[A]	F-SCAN(A)
[B]	F-SCAN(B)
[C]	I-SCAN(C)

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	1. $[AB]\bowtie_L C$
	2. [BC]⋈[A]	2. $[BC]\bowtie_L A$
	3. [AC]⋈[B]	3. $[AC]\bowtie_L B$
	4. [B]⋈[AC]	⋮

Cost: 50+(10+20)

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B]	1. $[A]\bowtie_L[B]$

Output: [BC]	Logical M-Exps	Physical M-Exps
	1. [B]⋈[C]	

Cost: 5

Output: [C]	Logical M-Exps	Physical M-Exps
	1. GET(C)	1. F-SCAN(C)
		2. I-SCAN(C)

Output: [AC]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[C]	
	2. [C]⋈[A]	

Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. F-SCA
		2. I-SCAN(A)

Output: [B]		...l M-Exps
		...SCAN(B)
		...SCAN(B)

SEARCH TERMINATION

Approach #1: Wall-clock Time

→ Stop after the optimizer runs for some length of time.

Approach #2: Cost Threshold

→ Stop when the optimizer finds a plan that has a lower cost than some threshold.

Approach #3: Transformation Exhaustion

→ Stop when there are no more ways to transform the target plan. Usually done per group.

CASCADES IMPLEMENTATIONS

Standalone:

→ Wisconsin OPT++ (1990s)
→ Portland State Columbia (1990s)
→ Pivotal Orca (2010s)
→ Apache Calcite (2010s)

Integrated:

→ Microsoft SQL Server (1990s)
→ Tandem NonStop SQL (1990s)
→ Clustrix (2000s)
→ CMU Peloton (2010s)

PREDICATE EXPRESSIONS

Predicates are defined as part of each operator.
→ These are typically represented as an AST.
→ Postgres implements them as flatten lists.

The same logical operator can be represented in multiple physical operators using variations of the same expression.

PREDICATE PUSHDOWN

Approach #1: Logical Transformation

→ Like any other transformation rule in Cascades.
→ Can use cost-model to determine benefit.

Approach #2: Rewrite Phase

→ Perform pushdown <u>before</u> starting search using an initial rewrite phase. Tricky to support complex predicates.

Approach #3: Late Binding

→ Perform pushdown <u>after</u> generating optimal plan in Cascades. Will likely produce a bad plan.

PREDICATE MIGRATION

Observation: Not all predicates cost the same to evaluate on tuples.

```
SELECT * FROM foo
 WHERE foo.id = 1234
   AND SHA_512(foo.val) = '...'
```

The optimizer should consider selectivity and computation cost when determining the evaluation order of predicates.

PREDICATE MIGRATION: OPTIMIZING QUERIES WITH EXPENSIVE PREDICATES
SIGMOD 1993

PIVOTAL ORCA

Standalone Cascades implementation.

→ Originally written for Greenplum.

→ Extended to support HAWQ.

A DBMS can use Orca by implementing API to send catalog + stats + logical plans and then retrieve physical plans.

Supports multi-threaded search.

ORCA: A MODULAR QUERY OPTIMIZER ARCHITECTURE FOR BIG DATA
SIGMOD 2014

ORCA – ENGINEERING

Issue #1: Remote Debugging

→ Automatically dump the state of the optimizer (with inputs) whenever an error occurs.

→ The dump is enough to put the optimizer back in the exact same state later on for further debugging.

Issue #2: Optimizer Accuracy

→ Automatically check whether the ordering of the estimate cost of two plans matches their actual execution cost.

MEMSQL OPTIMIZER

Rewriter

→ Logical-to-logical transformations with access to the cost-model.

Enumerator

→ Logical-to-physical transformations.

→ Mostly join ordering.

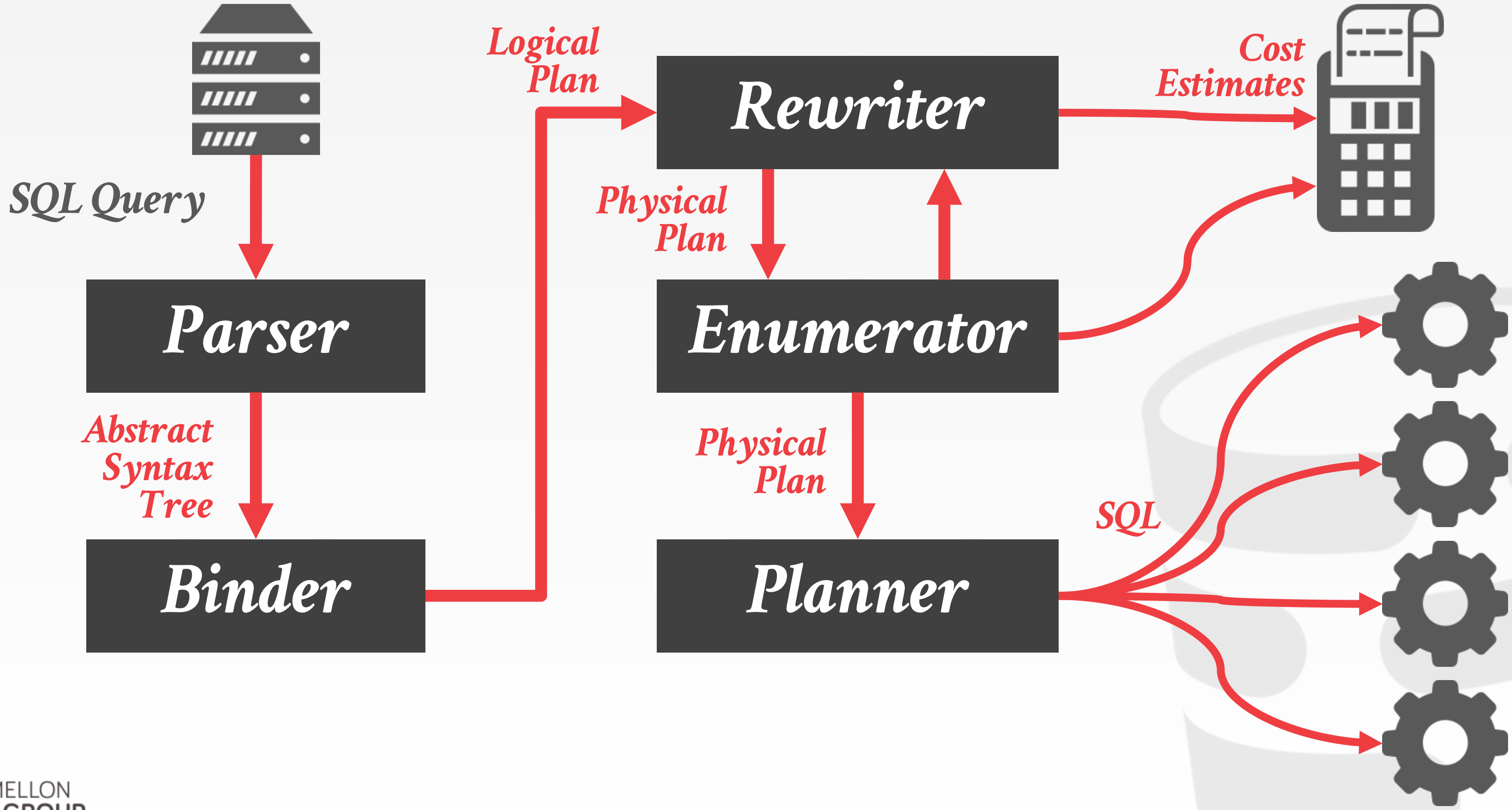
Planner

→ Convert physical plans back to SQL.

→ Contains MemSQL-specific commands for moving data.

THE MEMSQL QUERY OPTIMIZER
VLDB 2017

MEMSQL OPTIMIZER OVERVIEW



PARTING THOUGHTS

"Query optimization is not rocket science. When you flunk out of query optimization, we make you go build rockets." – *David DeWitt*

The research literature suggests that there is no difference in quality between bottom-up vs. top-down search strategies.

All of this hinges on a good cost model.
A good cost model needs good statistics.


Joe Hellerstein
@joe_hellerstein
Following
Surprising comment from Graefe on query optimization: use dynamic prog for joins, cascades for extensibility operators.
11:14 AM - 18 May 2017
4 Retweets 14 Likes
1 4 14
Siva Narayanan @K2_181 · 18 May 2017
Replying to @joe_hellerstein
can you elaborate?
1
Joe Hellerstein @joe_hellerstein · 18 May 2017
He said that there's a paper (maybe Thomas neumann?) showing dp more efficient than cascades for join enumeration.
1

EXTRA CREDIT

Each student can earn extra credit if they write a encyclopedia article about a DBMS.

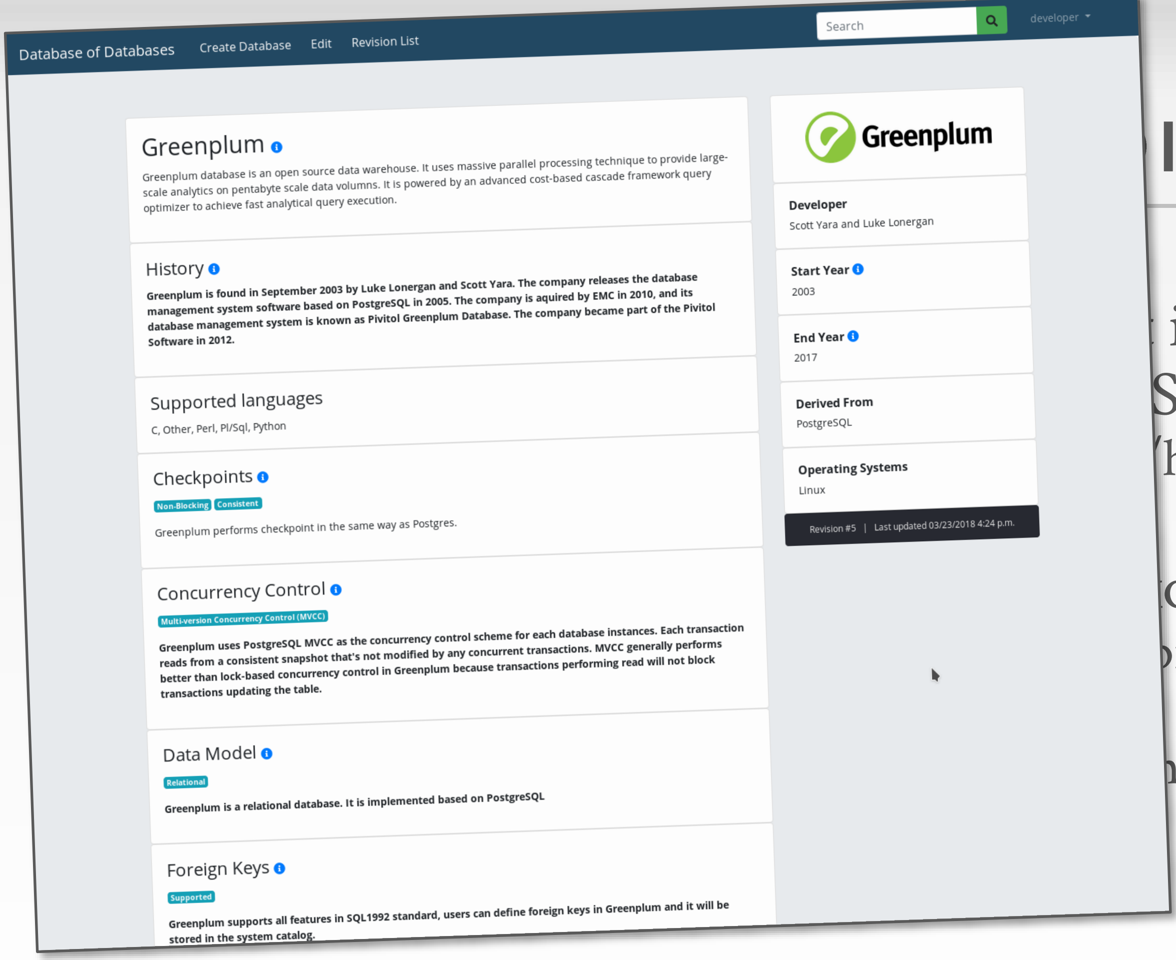
→ Can be academic/commercial, active/historical.

Each article will use a standard taxonomy.

→ For each feature category, you select pre-defined options for your DBMS.

→ You will then need to provide a summary paragraph with citations for that category.

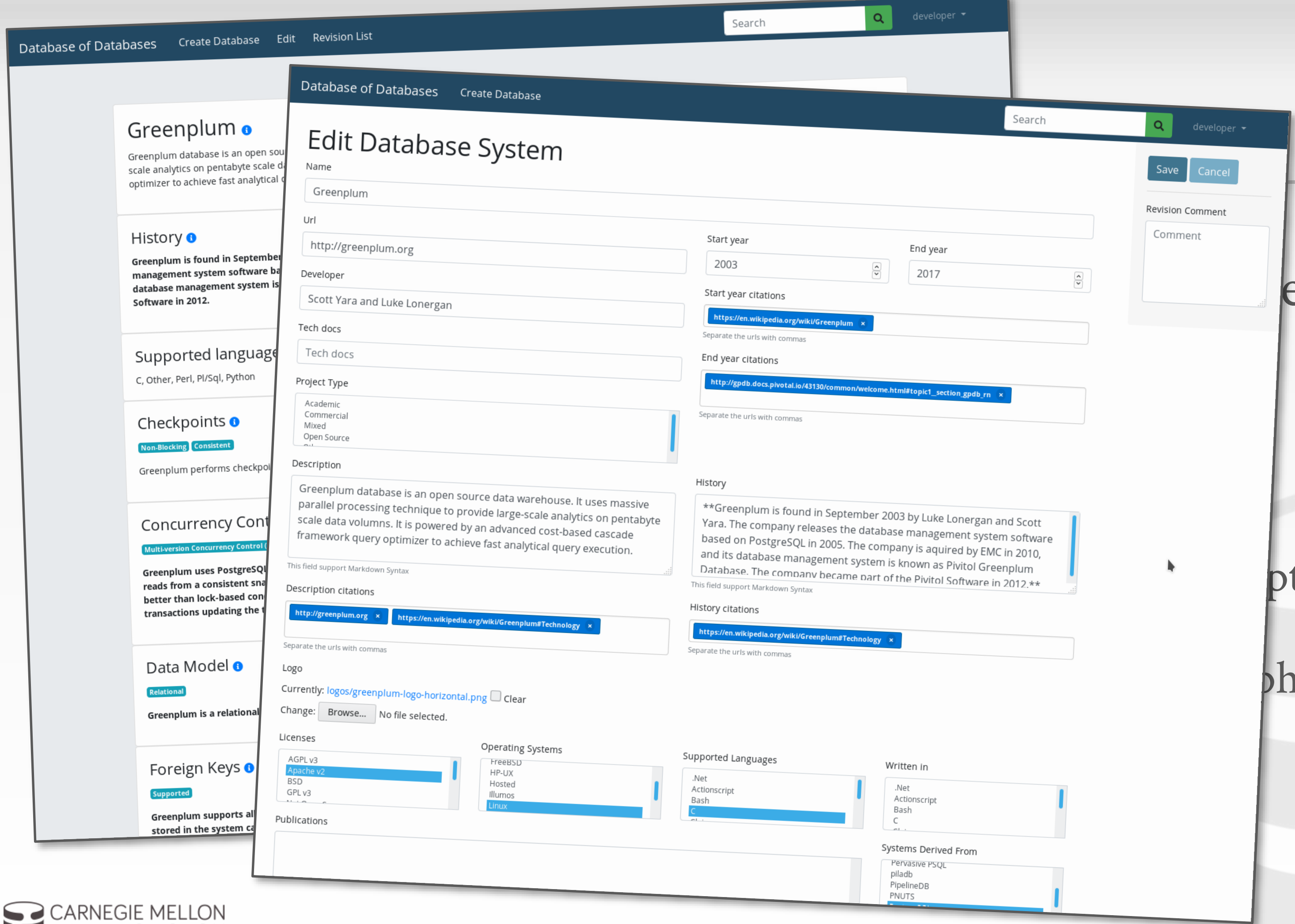
IT



if they write a
S.
historical.

onomy.
pre-defined options

ary paragraph with

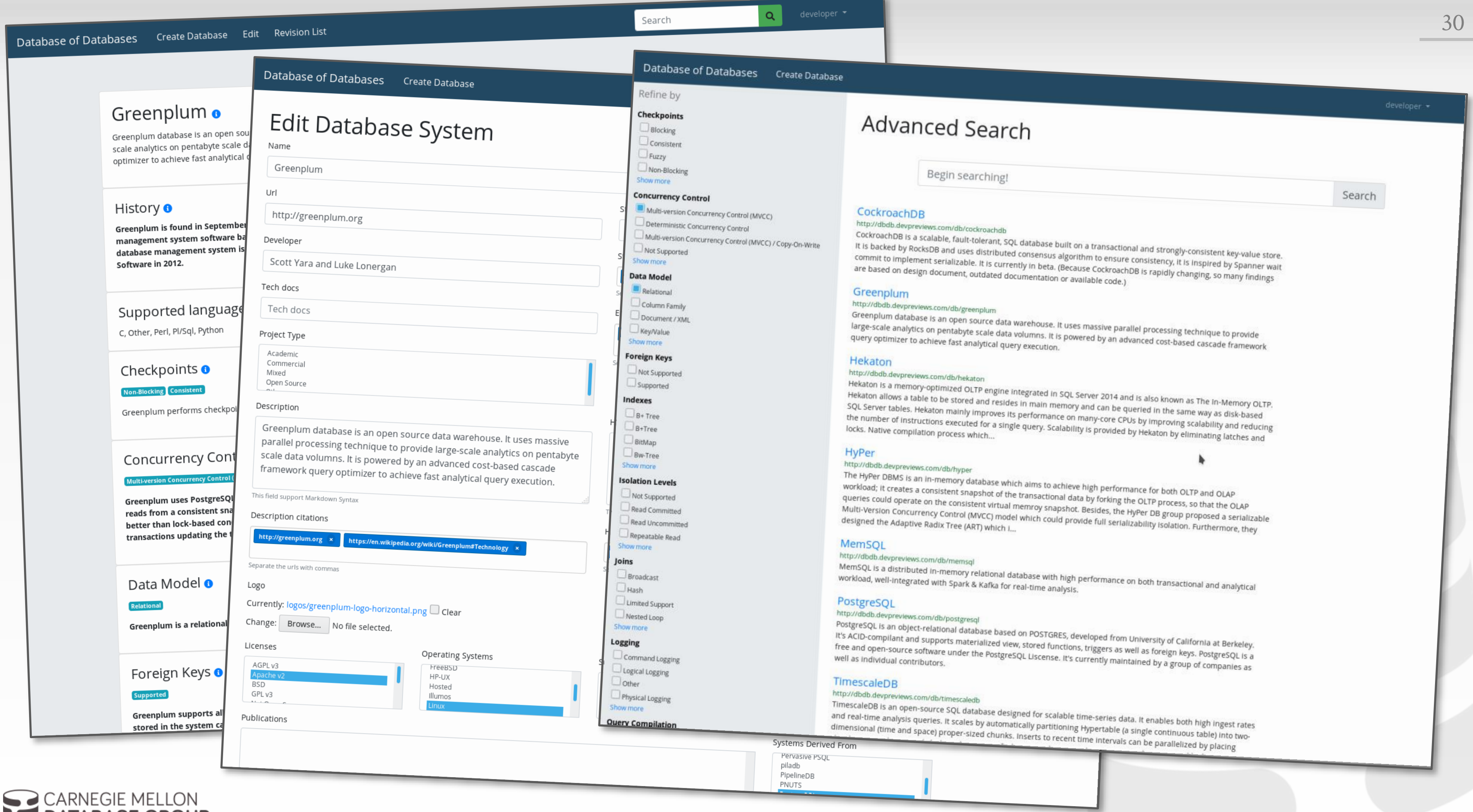


e a

ptions

ph with

CARNEGIE MELLON
DATABASE GROUP



Database of Databases
Create Database
Edit
Revision List
Search
developer
Greenplum
History
Supported languages
C, Other, Perl, Pl/Sql, Python
Checkpoints
Non-Blocking
Consistent
Concurrency Control
Data Model
Relational
Foreign Keys
Supported
Edit Database System
Name
Greenplum
Url
http://greenplum.org
Developer
Scott Yara and Luke Lonergan
Tech docs
Project Type
Academic
Commercial
Mixed
Open Source
Description
Greenplum database is an open source data warehouse. It uses massive parallel processing technique to provide large-scale analytics on pentabyte scale data volumns. It is powered by an advanced cost-based cascade framework query optimizer to achieve fast analytical query execution.
This field support Markdown Syntax
Description citations
http://greenplum.org
https://en.wikipedia.org/wiki/Greenplum#Technology
Separate the urls with commas
Logo
Currently: logos/greenplum-logo-horizontal.png Clear
Change: Browse... No file selected.
Licenses
AGPL v3
Apache v2
BSD
GPL v3
Operating Systems
HP-UX
Hosted
Illumos
Linux
Publications
Systems Derived From
piladb
PipelineDB
PNUTS
Refine by
Checkpoints
Blocking
Consistent
Fuzzy
Non-Blocking
Show more
Concurrency Control
Multi-version Concurrency Control (MVCC)
Deterministic Concurrency Control
Multi-version Concurrency Control (MVCC) / Copy-On-Write
Not Supported
Data Model
Relational
Column Family
Document / XML
Key/Value
Foreign Keys
Not Supported
Supported
Indexes
B+ Tree
B+Tree
BitMap
Bw-Tree
Isolation Levels
Not Supported
Read Committed
Read Uncommitted
Repeatable Read
Joins
Broadcast
Hash
Limited Support
Nested Loop
Logging
Command Logging
Logical Logging
Other
Physical Logging
Query Compilation
Advanced Search
Begin searching!
Search
CockroachDB
http://dbdb.devpreviews.com/db/cockroachdb
CockroachDB is a scalable, fault-tolerant, SQL database built on a transactional and strongly-consistent key-value store. It is backed by RocksDB and uses distributed consensus algorithm to ensure consistency, it is inspired by Spanner wait commit to implement serializable. It is currently in beta. (Because CockroachDB is rapidly changing, so many findings are based on design document, outdated documentation or available code.)
Greenplum
http://dbdb.devpreviews.com/db/greenplum
Greenplum database is an open source data warehouse. It uses massive parallel processing technique to provide large-scale analytics on pentabyte scale data volumns. It is powered by an advanced cost-based cascade framework query optimizer to achieve fast analytical query execution.
Hekaton
http://dbdb.devpreviews.com/db/hekaton
HyPer
http://dbdb.devpreviews.com/db/hyper
MemSQL
http://dbdb.devpreviews.com/db/memsql
MemSQL is a distributed in-memory relational database with high performance on both transactional and analytical workload, well-integrated with Spark & Kafka for real-time analysis.
PostgreSQL
http://dbdb.devpreviews.com/db/postgresql
TimescaleDB
http://dbdb.devpreviews.com/db/timescaledb

DBDB.IO

All the articles will be hosted on our new website (currently under development).

→ I will post the user/pass on Piazza.

I will post a sign-up sheet for you to pick what DBMS you want to write about.

→ If you choose a widely known DBMS, then the article will need to be comprehensive.

→ If you choose an obscure DBMS, then you will have do the best you can to find information.

redis
IBM DB2
SQREAM
Microsoft SQL Server
TiDB
MySQL
STARCOUNTER
BuntDB
SQLite
WIREDTIGER
HyPer
cassandra
ClickHouse
BIGCHAINDB
WIREDTIGER
YugaByte
levelDB
VOLTDB
riak
ThinkSQL RDBMS
APACHE HBASE
presto
GoshawkDB
Couchbase
[tile]DB
cloudera IMPALA
Stardog
CDB
JustOne
LEANXCALE
FlureeDB
ScaleBase
BAGRI
NUODB
accumulo
MariaDB
INFINITEGRAPH
membase
AEROSPIKE
tdb
kinetica
SequoiaDB
CUBRID
RAVENDB
OrientDB
mldb.ai
mongoDB
SYLVA
deepdb
Informix
vectorwise
HIVE
HarperDB
AntidoteDB
RexDB
VERTICA
actorDB
AXIBASE
amazon DynamoDB
SAP HANA
NETEZZA
Hortonworks STINGER
RocksDB
RethinkDB
Spark
CUBRID
TILE38
DATAllegro
ArangoDB
Google Spanner
gun
FOUNDATIONDB
SYBASE
ORACLE BERKELEY DB
quasardb
Tokutek
CouchDB relax
BLAZINGDB
citusdata
piladb
VITESSE DATA
SCYLLA
PostgreSQL
PARACCEL
Clustrix
SUBWAY
HYPERTABLE
Trafodion
QuestDB PERFORMANCE DISTILLED
FATHOMDB
influxdb
memsql
MAPD
Paradigm4
CRATE.IO
TIMESCALE
dbShards
CARNEGIE MELLON DATABASE GROUP
ORACLE
ScaleDB
ScaleArc
IMDB

☠ PLAGIARISM WARNING ☠

This article must be your own writing with your own images. You may **not** copy text/images directly from papers or other sources that you find on the web.

Plagiarism will **not** be tolerated.
See CMU's Policy on Academic Integrity for additional information.

NEXT CLASS

Cost Models

Working in a large code base