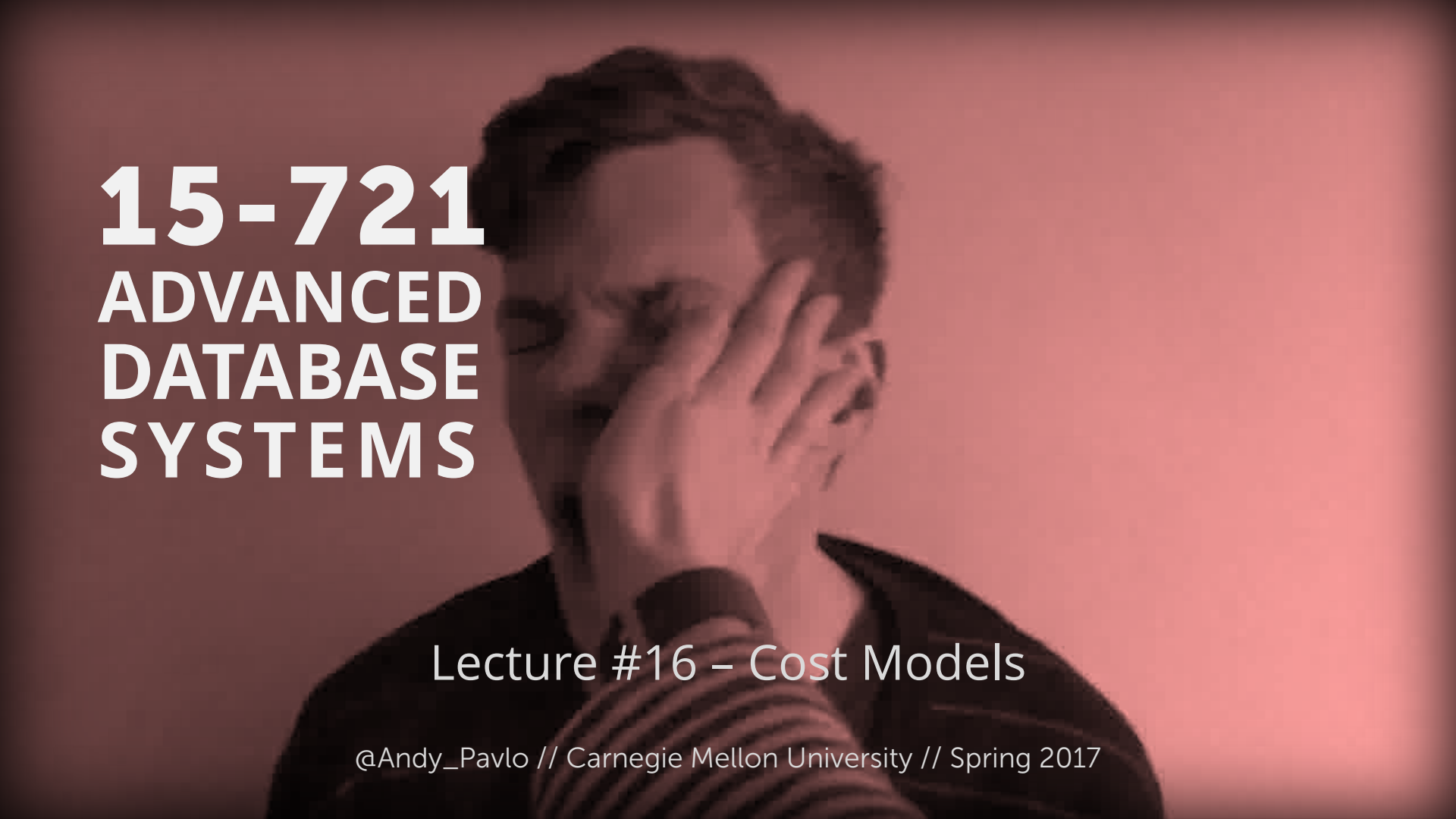


15-721 ADVANCED DATABASE SYSTEMS

Lecture #16 – Cost Models

@Andy_Pavlo // Carnegie Mellon University // Spring 2017

TODAY'S AGENDA

Cost Models

Cost Estimation

Extra Credit

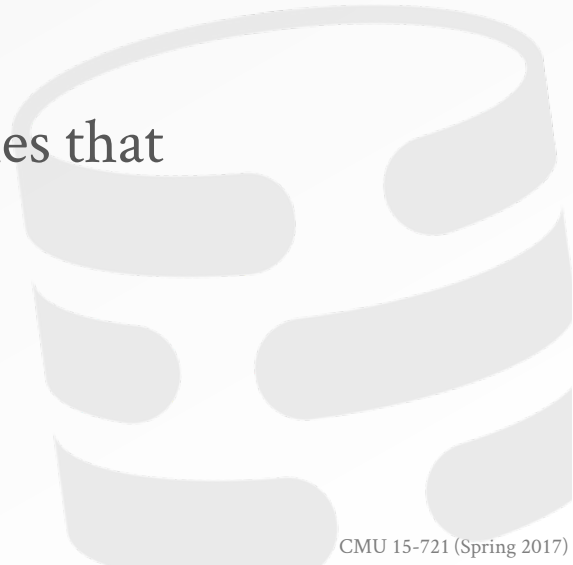
Working with a large code base

COST-BASED QUERY PLANNING

Generate an estimate of the cost of executing a particular query plan for the current state of the database.

→ Estimates are only meaningful internally.

This is independent of the search strategies that we talked about last class.



COST MODEL COMPONENTS

Choice #1: Physical Costs

→ Predict CPU cycles, I/O, cache misses, RAM consumption, pre-fetching, etc…
→ Depends heavily on hardware.

Choice #2: Logical Costs

→ Estimate result sizes per operator.
→ Independent of the operator algorithm.
→ Need estimations for operator result sizes.

Choice #3: Algorithmic Costs

→ Complexity of the operator algorithm implementation.

DISK-BASED DBMS COST MODEL

The number of disk accesses will always dominate the execution time of a query.

→ CPU costs are negligible.

→ Can easily measure the cost per I/O.

This is easier to model if the DBMS has full control over buffer management.

→ We will know the replacement strategy, pinning, and assume exclusive access to disk.

POSTGRES COST MODEL

Uses a combination of CPU and I/O costs that are weighted by "magic" constant factors.

Default settings are obviously for a disk-resident database without a lot of memory:

→ Processing a tuple in memory is **400x** faster than reading a tuple from disk.
→ Sequential I/O is **4x** faster than random I/O.

19.7.2. Planner Cost Constants

The *cost* variables described in this section are measured on an arbitrary scale. Only their relative values matter, hence scaling them all up or down by the same factor will result in no change in the planner's choices. By default, these cost variables are based on the cost of sequential page fetches; that is, `seq_page_cost` is conventionally set to `1.0` and the other cost variables are set with reference to that. But you can use a different scale if you prefer, such as actual execution times in milliseconds on a particular machine.

> **Note:** Unfortunately, there is no well-defined method for determining ideal values for the cost variables. They are best treated as averages over the entire mix of queries that a particular installation will receive. This means that changing them on the basis of just a few experiments is very risky.

`seq_page_cost` (`floating point`)

Sets the planner's estimate of the cost of a disk page fetch that is part of a series of sequential fetches. The default is 1.0. This value can be overridden for tables and indexes in a particular tablespace by setting the tablespace parameter of the same name (see ALTER TABLESPACE).

`random_page_cost` (`floating point`)

IBM DB2 COST MODEL

Database characteristics in system catalogs

Hardware environment (microbenchmarks)

Storage device characteristics (microbenchmarks)

Communications bandwidth (distributed only)

Memory resources (buffer pools, sort heaps)

Concurrency Environment

→ Average number of users
→ Isolation level / blocking
→ Number of available locks

Source: Guy Lohman

IN-MEMORY DBMS COST MODEL

No I/O costs, but now we have to account for CPU and memory access costs.

Memory cost is more difficult because the DBMS has no control cache management.

→ Unknown replacement strategy, no pinning, shared caches, non-uniform memory access.

The number of tuples processed per operator is a reasonable estimate for the CPU cost.

SMALLBASE COST MODEL

Two-phase model that automatically generates hardware costs from a logical model.

Phase #1: Identify Execution Primitives

→ List of ops that the DBMS does when executing a query
→ Example: evaluating predicate, index probe, sorting.

Phase #2: Microbenchmark

→ On start-up, profile ops to compute CPU/memory costs
→ These measurements are used in formulas that compute operator cost based on table size.

MODELLING COSTS FOR A MM-DBMS
Real-Time Databases 1996

COST MODELS IN CASCADES

A top-down optimizer doesn't know the number of tuples passed as input to an operator without knowing the physical operators for its children.

Logical estimates are the "worst case" scenario for that operator.

Maintain two costs per operator group:
→ **Upper-bound:** Logical operator cost
→ **Lower-bound:** Physical operator cost

OBSERVATION

The number of tuples processed per operator depends on three factors:

→ The access methods available per table
→ The distribution of values in the database's attributes
→ The predicates used in the query

Simple queries are easy to estimate.
More complex queries are not.

SELECTIVITY

The **selectivity** of an operator is the percentage of data accessed for a predicate.

→ Modeled as probability of whether a predicate on any given tuple will be satisfied.

The DBMS estimates selectivities using:

→ Domain Constraints
→ Precomputed Statistics (Zone Maps)
→ Histograms / Approximations
→ Sampling

IBM DB2 – LEARNING OPTIMIZER

Update table statistics as the DBMS scans a table during normal query processing.

Check whether the optimizer's estimates match what it encounters in the real data and incrementally updates them.

LEO - DB2'S LEARNING OPTIMIZER
VLDB 2001

APPROXIMATIONS

Maintaining exact statistics about the database is expensive and slow.

Use approximate data structures called **sketches** to generate error-bounded estimates.

→ Count Distinct
→ Quantiles
→ Frequent Items
→ Tuple Sketch

See Yahoo! Sketching Library

SAMPLING

Execute a predicate on a random sample of the target data set.

The # of tuples to examine depends on the size of the table.

The DBMS can perform this sampling with **READ UNCOMMITTED** isolation.

RESULT CARDINALITY

The number of tuples that will be generated per operator is computed from its selectivity multiplied by the number of tuples in its input.

RESULT CARDINALITY

Assumption #1: Uniform Data

→ The distribution of values (except for the heavy hitters) is the same.

Assumption #2: Independent Predicates

→ The predicates on attributes are independent

Assumption #3: Inclusion Principle

→ The domain of join keys overlap such that each key in the inner relation will also exist in the outer table.

CORRELATED ATTRIBUTES

Consider a database of automobiles:

→ # of Makes = 10, # of Models = 100

And the following query:

→ `(make="Honda" AND model="Accord")`

With the independence and uniformity assumptions, the selectivity is:

→ 1/10 × 1/100 = 0.001

But since only Honda makes Accords the real selectivity is 1/100 = 0.01

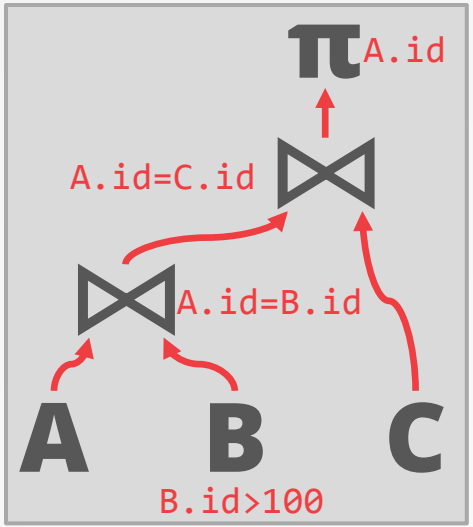
COLUMN GROUP STATISTICS

The DBMS can track statistics for groups of attributes together rather than just treating them all as independent variables.

→ Only supported in commercial systems.
→ Requires the DBA to declare manually.

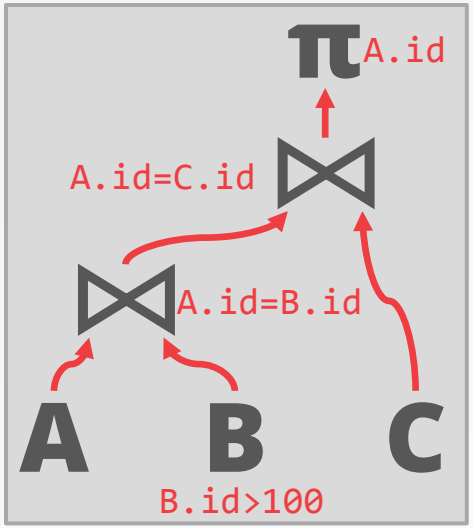
ESTIMATION PROBLEM

```
SELECT A.id
  FROM A, B, C
 WHERE A.id = B.id
   AND A.id = C.id
   AND B.id > 100
```



ESTIMATION PROBLEM

```
SELECT A.id
  FROM A, B, C
 WHERE A.id = B.id
   AND A.id = C.id
   AND B.id > 100
```



Compute the cardinality of base tables

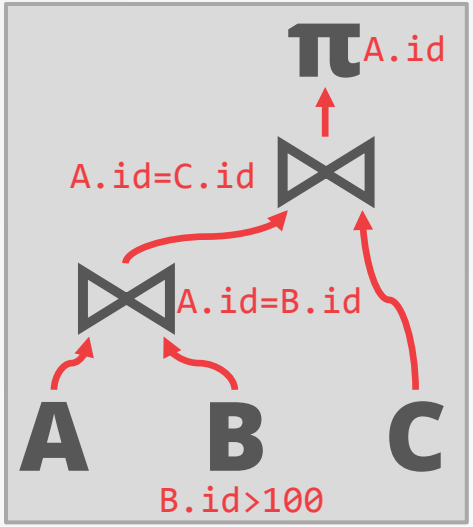
$$\mathbf{A} \rightarrow |\mathbf{A}|$$

$$\mathbf{B}.\text{id}>100 \rightarrow |\mathbf{B}| \quad \textit{sel}(\mathbf{B}.\text{id}>100)$$

$$\mathbf{C} \rightarrow |\mathbf{C}|$$

ESTIMATION PROBLEM

```
SELECT A.id
  FROM A, B, C
 WHERE A.id = B.id
   AND A.id = C.id
   AND B.id > 100
```



Compute the cardinality of base tables

$$A \rightarrow |A|$$

$$B.id>100 \rightarrow |B| \ \ sel(B.id>100)$$

$$C \rightarrow |C|$$

Compute the cardinality of join results

$$A \bowtie B = (|A| \ \ |B|) / max(sel(A.id=B.id), sel(B.id>100))$$

$$(A \bowtie B) \bowtie C = (|A| \ \ |B| \ \ |C|) / max(sel(A.id=B.id), sel(B.id>100), sel(A.id=C.id))$$

ESTIMATOR QUALITY

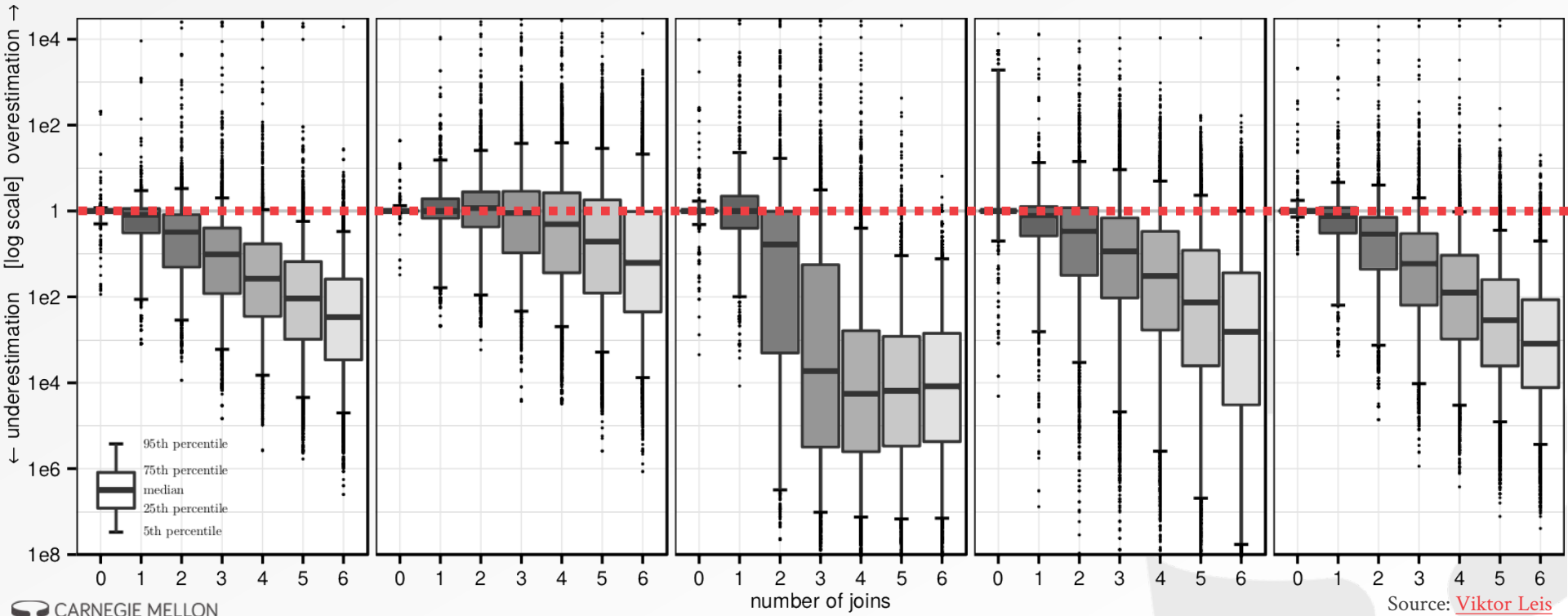
Evaluate the correctness of cardinality estimates generated by DBMS optimizers as the number of joins increases.

→ Let each DBMS perform its stats collection.
→ Extract measurements from query plan

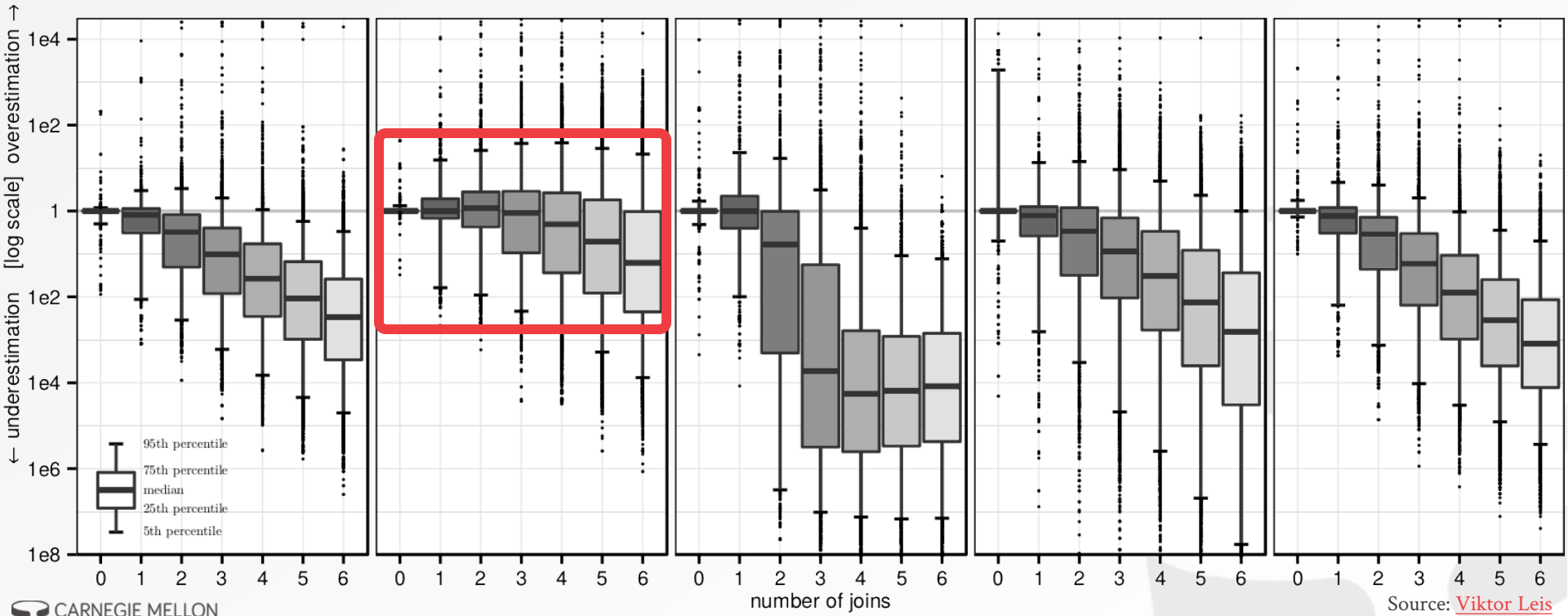
Compared five DBMSs using 100k queries.

HOW GOOD ARE QUERY OPTIMIZERS, REALLY?
VLDB 2015

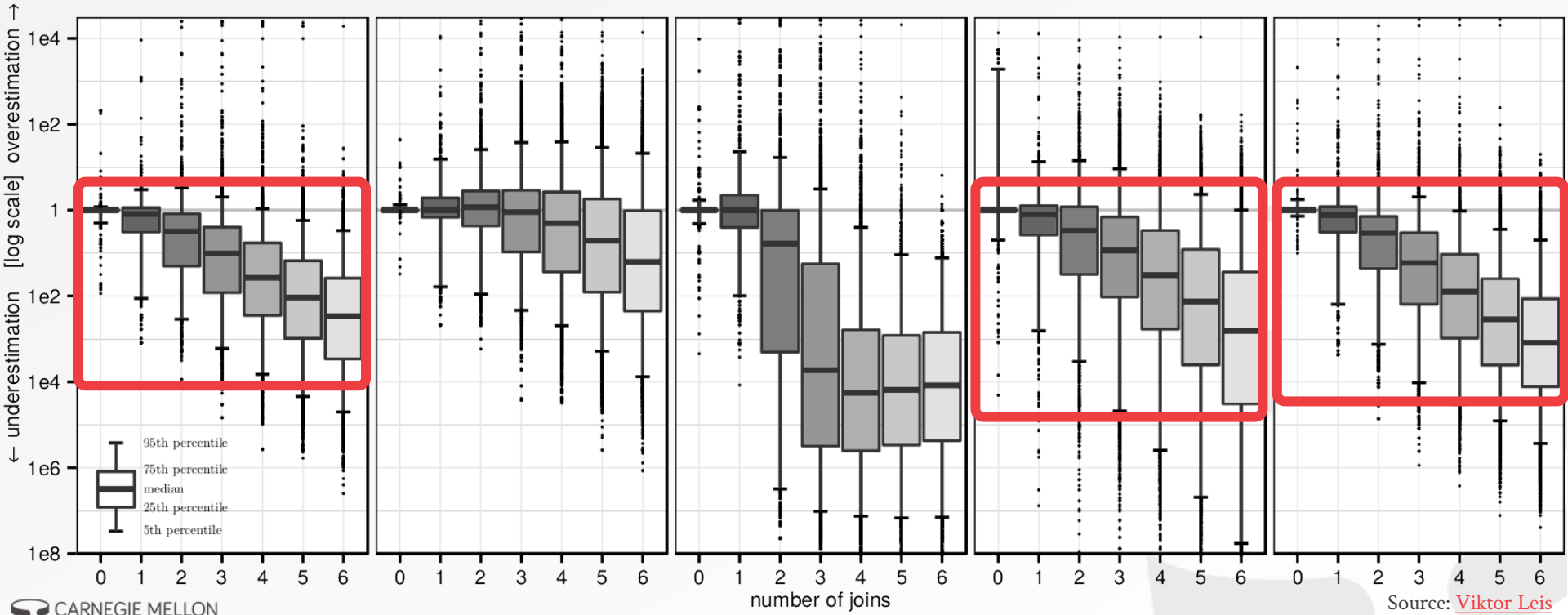
ESTIMATOR QUALITY



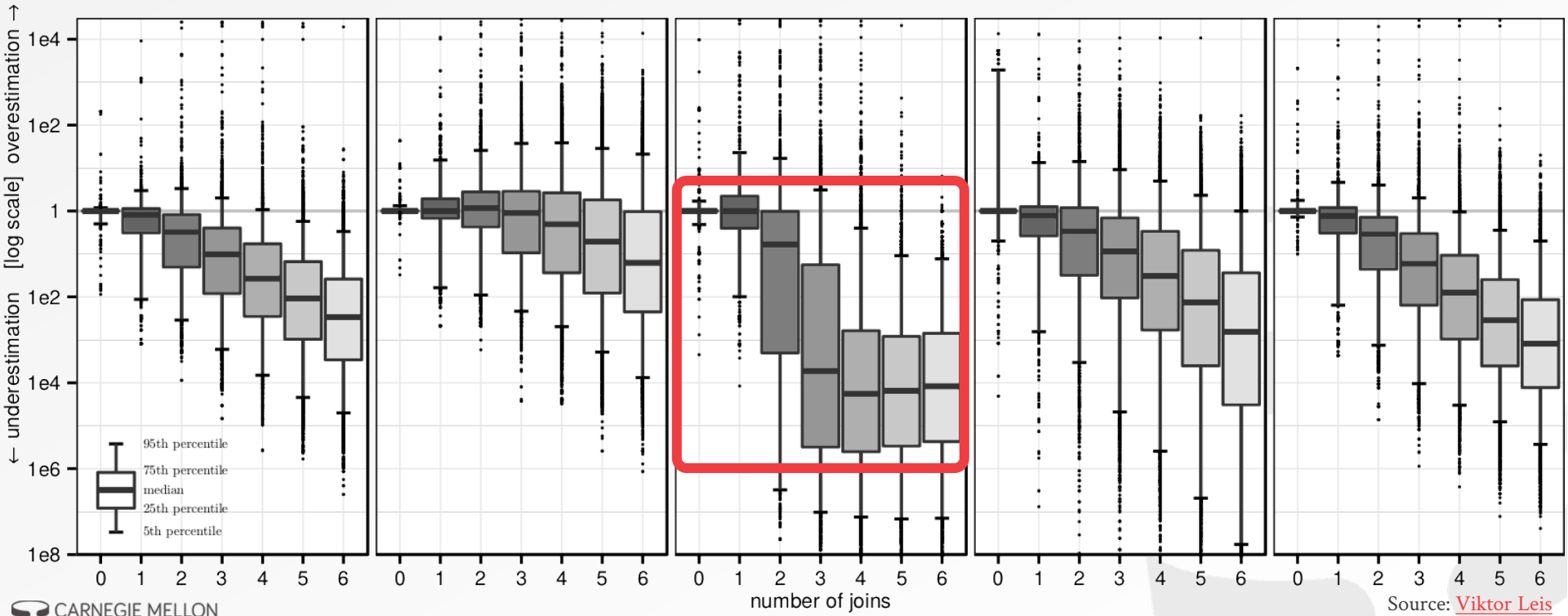
ESTIMATOR QUALITY



ESTIMATOR QUALITY

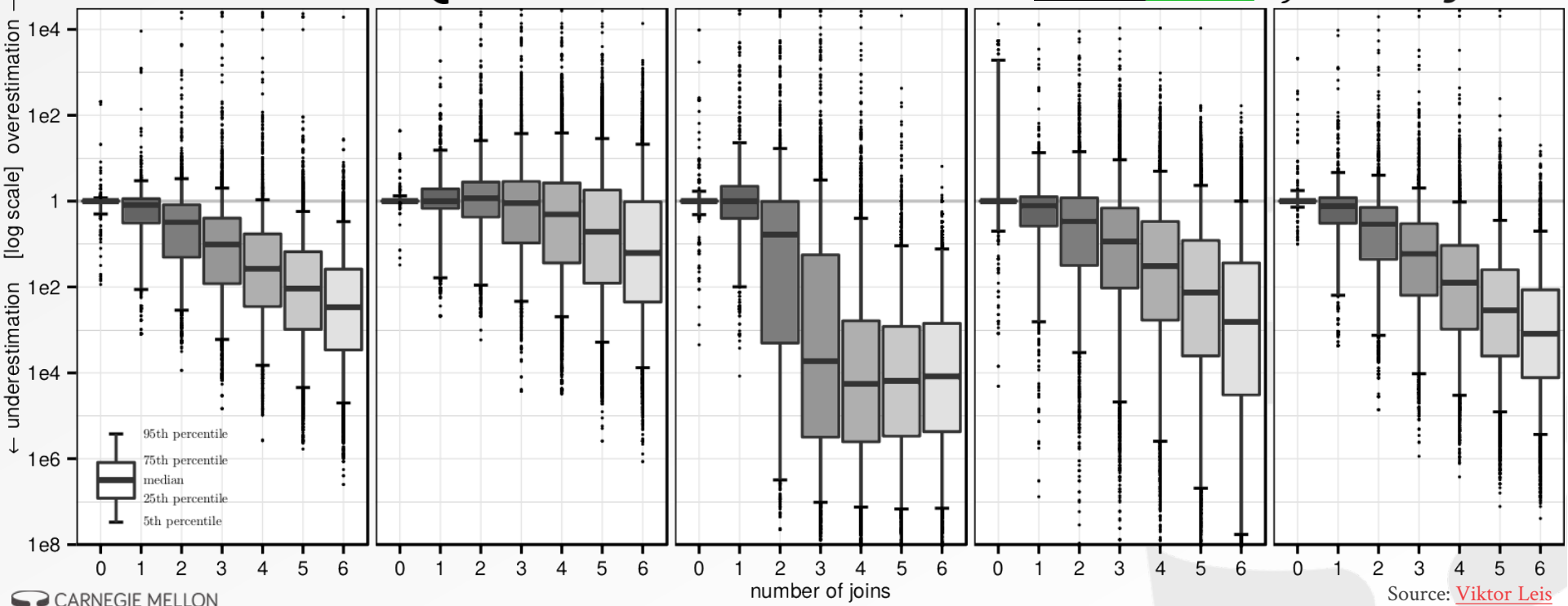


ESTIMATOR QUALITY



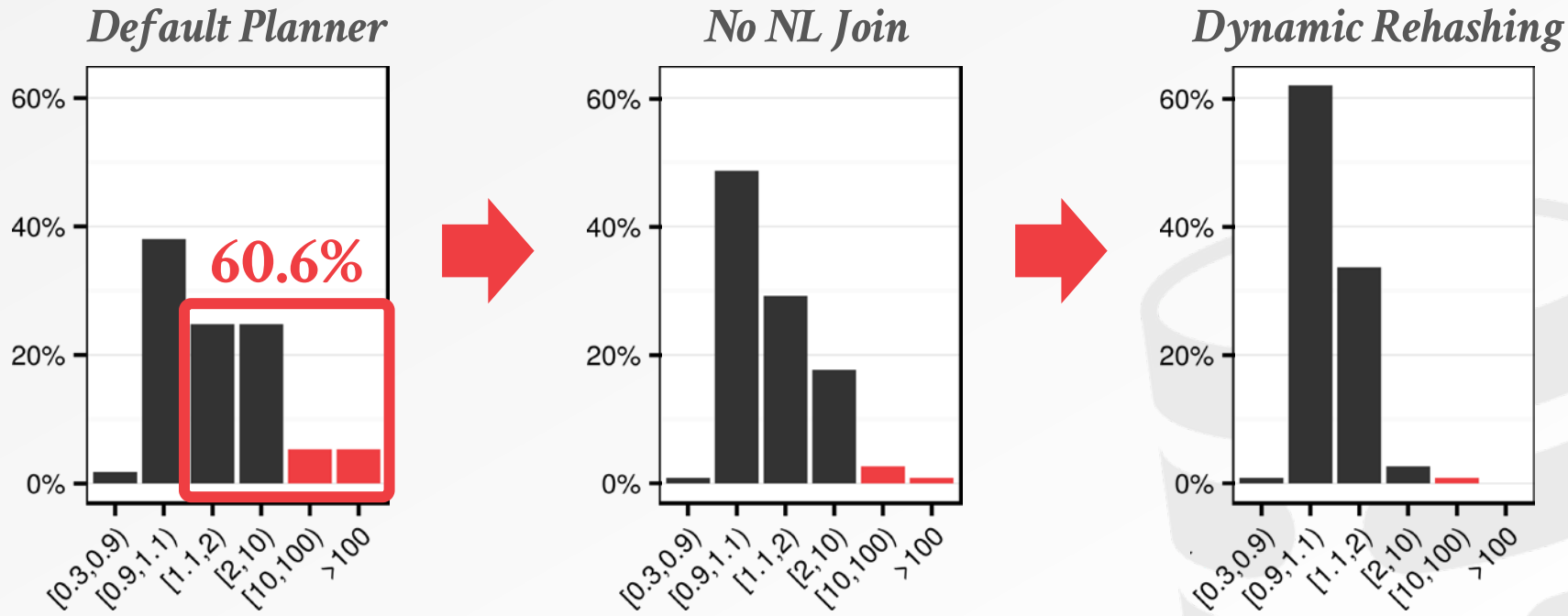
ESTIMATOR QUALITY

PostgreSQL Microsoft® SQL Server® ORACLE® IBM DB2 HyPer



EXECUTION SLOWDOWN

Postgres 9.4 – JOB Workload



Slowdown compared to using true cardinalities

Source: Viktor Leis

LESSONS FROM THE GERMANS

Query opt is more important than a fast engine
→ Cost-based join ordering is necessary

Cardinality estimates are routinely wrong
→ Try to use operators that do not rely on estimates

Hash joins + seq scans are a robust exec model
→ The more indexes that are available, the more brittle the plans become (but also faster on average)

Working on accurate models is a waste of time
→ Better to improve cardinality estimation instead

PARTING THOUGHTS

Using number of tuples processed is a reasonable cost model for in-memory DBMSs.
→ But computing this is non-trivial.

If you are building a new DBMS, then using Volcano/Cascade planning + # of tuples cost model is the way to go.

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.

DBDB.IO

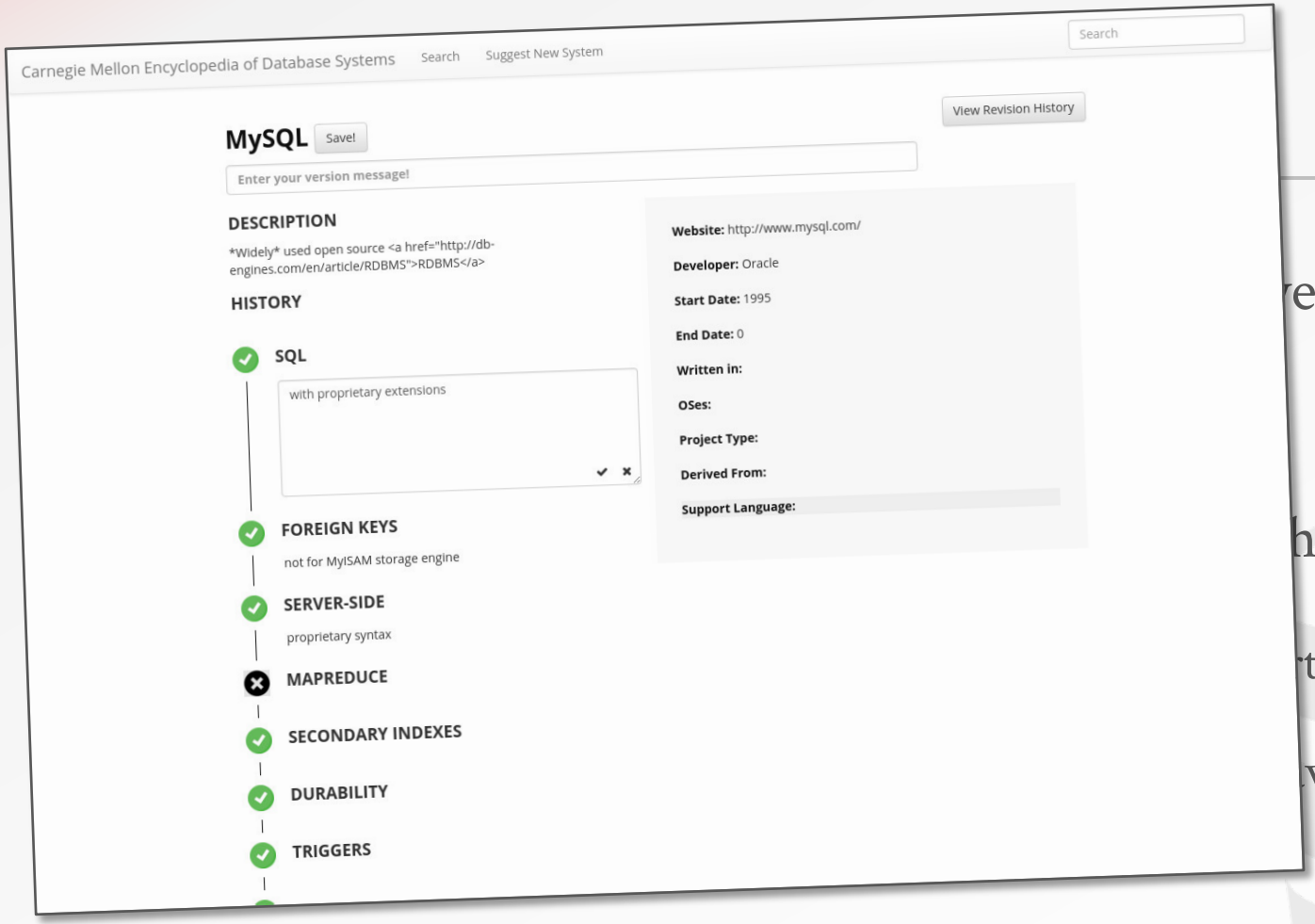
All the articles will be hosted on our new website (currently under development).

→ 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.

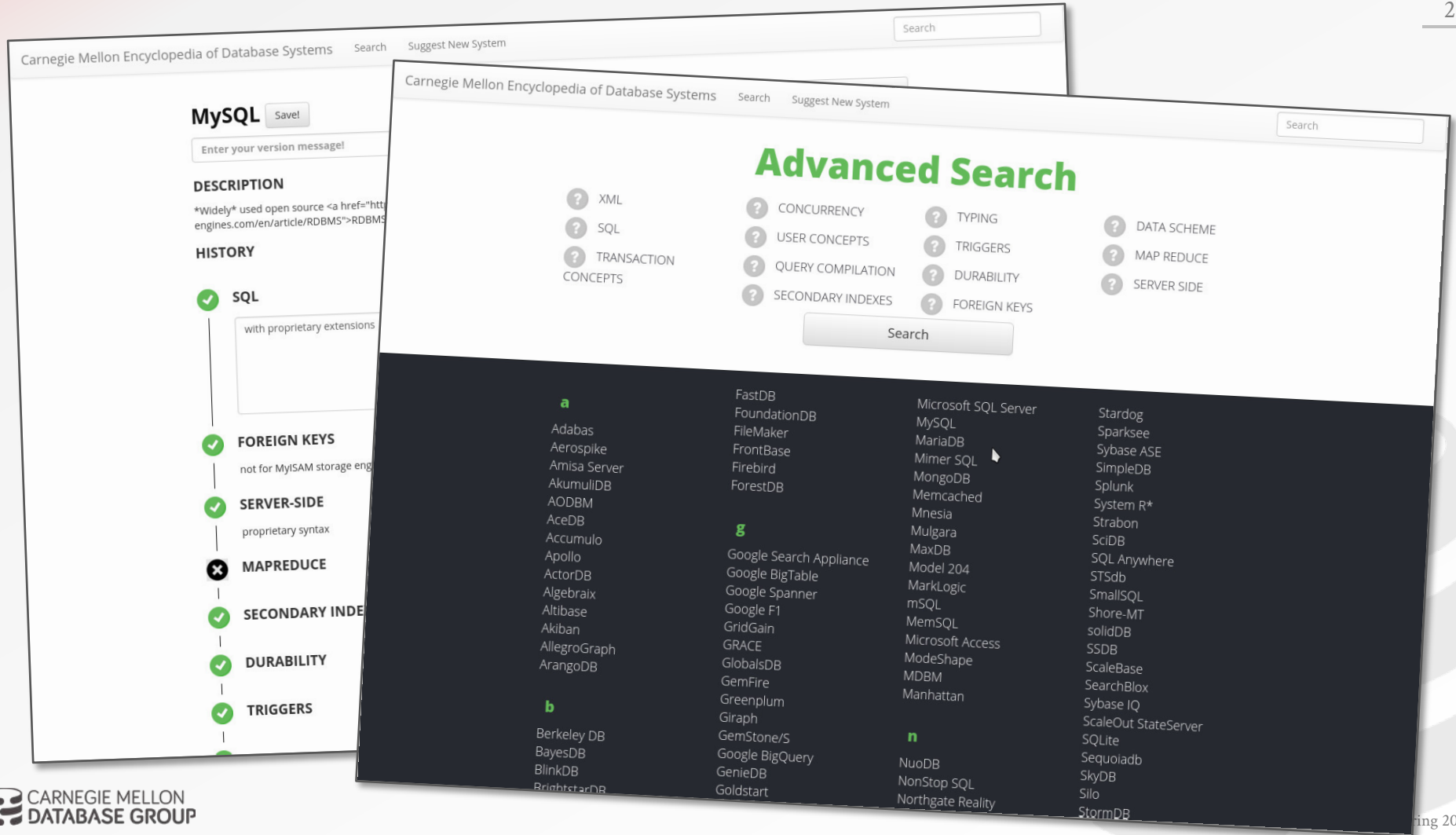


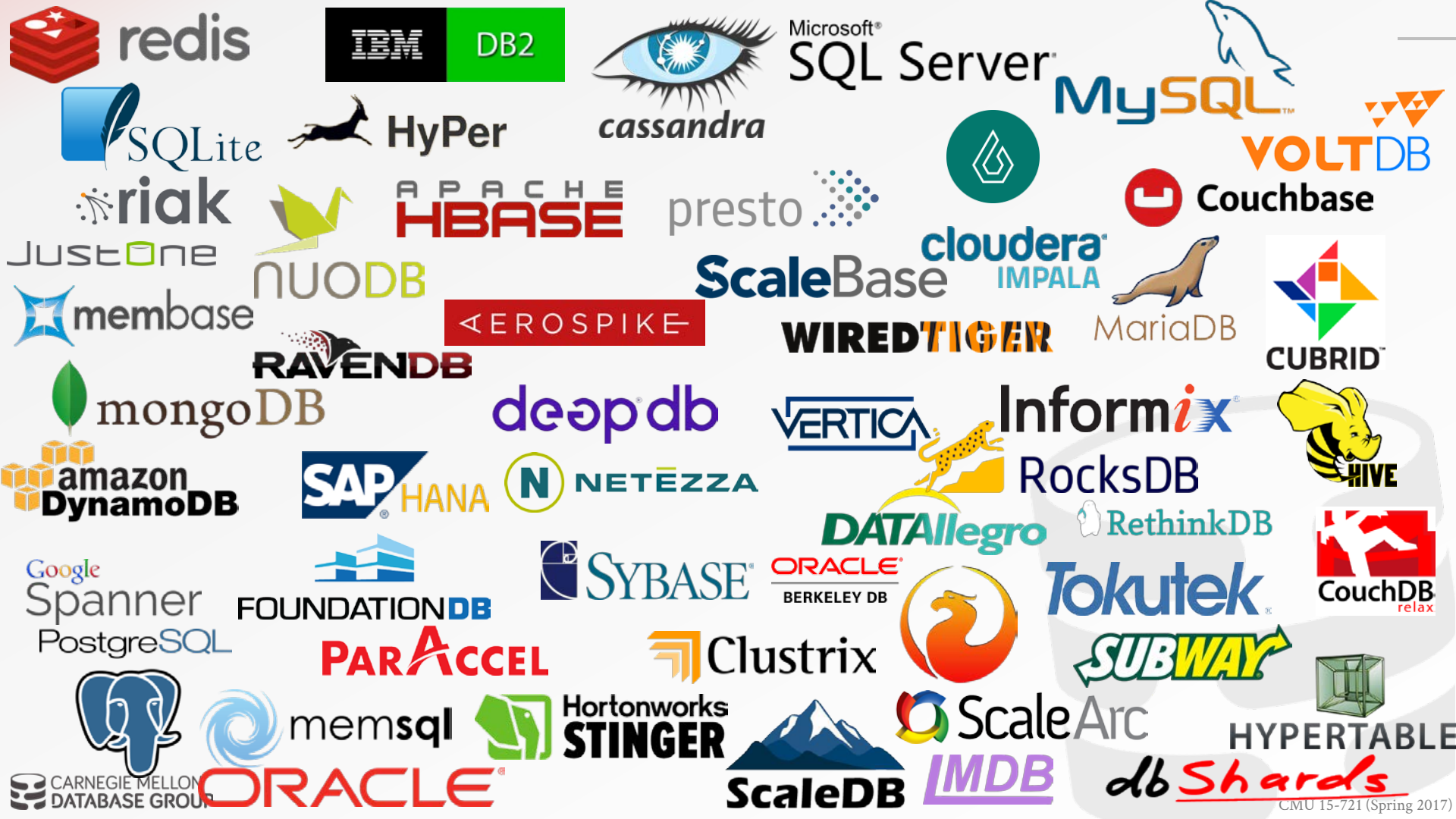
ebsite

hat

ticle will

ve do


Carnegie Mellon Encyclopedia of Database Systems
Search
Suggest New System
MySQL
Save!
Enter your version message!
DESCRIPTION
HISTORY
SQL
with proprietary extensions
FOREIGN KEYS
not for MyISAM storage eng
SERVER-SIDE
proprietary syntax
MAPREDUCE
SECONDARY INDE
DURABILITY
TRIGGERS
Carnegie Mellon Encyclopedia of Database Systems
Search
Suggest New System
Search
Advanced Search
XML
SQL
TRANSACTION CONCEPTS
CONCURRENCY
USER CONCEPTS
QUERY COMPILATION
SECONDARY INDEXES
TYPING
TRIGGERS
DURABILITY
FOREIGN KEYS
DATA SCHEME
MAP REDUCE
SERVER SIDE
Search
a
Adabas
Aerospike
Amisa Server
AkumuliDB
AODBM
AceDB
Accumulo
Apollo
ActorDB
Algebraix
Altibase
Akiban
AllegroGraph
ArangoDB
b
Berkeley DB
BayesDB
BlinkDB
FastDB
FoundationDB
FileMaker
FrontBase
Firebird
ForestDB
g
Google Search Appliance
Google BigTable
Google Spanner
Google F1
GridGain
GRACE
GlobalsDB
GemFire
Greenplum
Giraph
GemStone/S
Google BigQuery
GenieDB
Goldstart
Microsoft SQL Server
MySQL
MariaDB
Mimer SQL
MongoDB
Memcached
Mnesia
Mulgara
MaxDB
Model 204
MarkLogic
mSQL
MemSQL
Microsoft Access
ModeShape
MDBM
Manhattan
n
NuoDB
NonStop SQL
Northgate Reality
Stardog
Sparksee
Sybase ASE
SimpleDB
Splunk
System R*
Strabon
SciDB
SQL Anywhere
STSdb
SmallSQL
Shore-MT
solidDB
SSDB
ScaleBase
SearchBlox
Sybase IQ
ScaleOut StateServer
SQLite
Sequoiadb
SkyDB
Silo
StormDB


redis
IBM DB2
Microsoft SQL Server
MySQL
SQLite
HyPer
cassandra
VOLTDB
riak
APACHE HBASE
presto
Couchbase
JustOne
NUODB
cloudera IMPALA
ScaleBase
membase
AEROSPIKE
MariaDB
CUBRID
RAVENDB
WIREDTIGER
mongoDB
deepdb
VERTICA
Informix
HIVE
amazon DynamoDB
SAP HANA
NETEZZA
RocksDB
DATAllegro
RethinkDB
Google Spanner
FOUNDATIONDB
SYBASE
ORACLE BERKELEY DB
Tokutek
CouchDB relax
PostgreSQL
PARACCEL
Clustrix
SUBWAY
memsql
Hortonworks STINGER
ScaleArc
HYPERTABLE
ORACLE
ScaleDB
IMDB
db Shards
CARNEGIE MELLON DATABASE GROUP

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.

ANDY'S

LIFE LESSONS FOR WORKING ON CODE

DISCLAIMER

I have worked on a few large projects in my lifetime (2 DBMSs, 1 distributed system).

I have also read a large amount of "enterprise" code for legal stuff over multiple years.

But I'm not claiming to be all knowledgeable in modern software engineering practices.

OBSERVATION

Most software development is never from scratch. You will be expected to be able to work with a large amount of code that you did not write.

Being able to independently work on a large code base is the #1 skill that companies tell me they are looking for in students they hire.

PASSIVE READING

Reading the code for the sake of reading code is (usually) a waste of time.

→ It's hard to internalize functionality if you don't have direction.

It's important to start working with the code right away to understand how it works.

TEST CASES

Adding or improving tests allows you to improve the reliability of the code base without the risk of breaking production code.

→ It forces you to understand code in a way that is not possible when just reading it.

Nobody will complain (hopefully) about adding new tests to the system.

REFACTORING

Find the general location of code that you want to work on and start cleaning it up.

→ Add/edit comments
→ Clean up messy code
→ Break out repeated logic into separate functions.

Tread lightly though because you are changing code that you are not familiar with yet.

TOOLCHAINS & PROCESSES

Beyond working on the code, there will also be an established protocol for software development.

More established projects will have either training or comprehensive documentation.

→ If the documentation isn't available, then you can take the initiative and try to write it.

NEXT CLASS

Project #3 Proposals

Please email me if you want to talk about your potential topic.