Optimizer Challenges in a Multi-Tenant World

- Pat Selinger

  pselinger@salesforce.come

Classic Query Optimizer Concepts & Assumptions

Relational Model

Cost = X * CPU + Y * I/O

Cardinality

Selectivity

Clustering and other storage methods

Table access methods

- Indexes (clustering and non)
- Full scan

Join order

Join methods

- Hash
- Merge
- Nested loop
- …

Statistics kept per table and per index

Classic Example for Single Tenant

Index and data pages with unclustered index st_ix on orders.st
SELECT sales_total st, ...
FROM orders
WHERE st > 100000

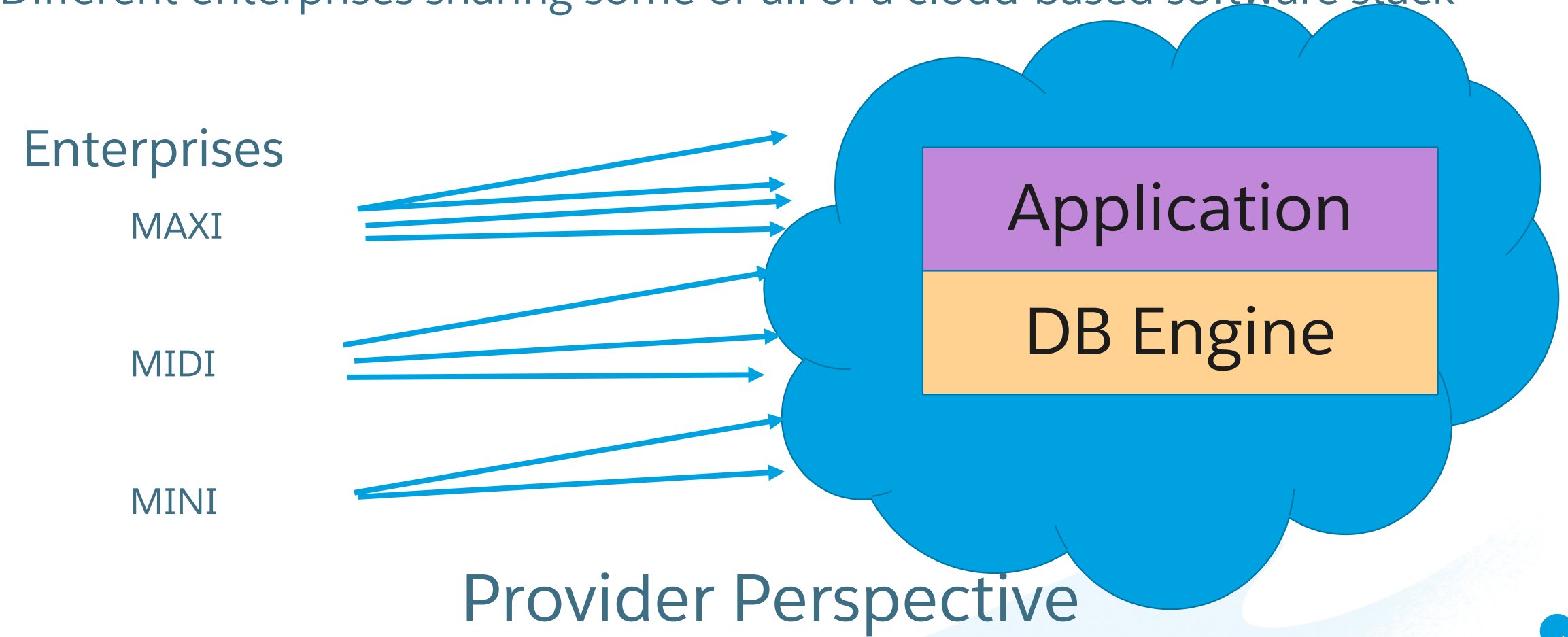
Choose Min(cost(table_scan(orders)),
((max(st)-100000) / (max(st)-min(st))) * cost(index_scan(st_ix))

Selinger et al., Access path selection in a relational database system, SIGMOD 1979

What is Multi-Tenancy According to Pat?

Shared HW, SW code, SW stack, SW stack + schema...

Different enterprises sharing some or all of a cloud-based software stack



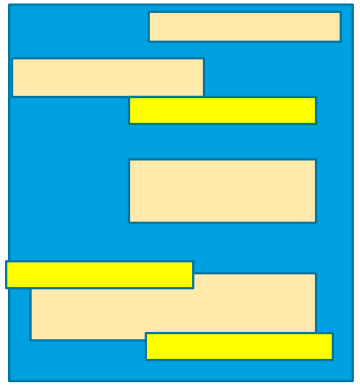
Provider Perspective

Multi-Tenancy Assumptions

Perspective of a cloud-based solution provider

Shared schema: Tables shared between (lots of) enterprises

Orders Table



MAXI

MIDI

MINI

Challenges of Multi-Tenancy

Table and index statistics are shared between enterprises, useless

Same DB request needs different plans, depending on enterprise

Are the same join methods useful?

Can materialized views (and queries and statistics on them) help?

Can “hints” or plan constraints help?

What does clustering mean in this context?

Opportunities for tuning DB requests depend on what the solution provider can change

Application	DB Engine	Result
NO	NO	Out of luck
YES	NO	Case APP-only
YES	YES	Case APP-DB

APP-only Challenges…

DB statistics are table level, not enterprise within table

Example:
SELECT …
FROM orders
WHERE eid = ______
AND sales_total > 100000

Issues:

- N values for Enterprise ID does not imply eid selectivity is 1/N, n-tile, etc.
- Disparity in # orders for MAXI vs MINI
- Correlation between sales_total and eid

APP-only Challenges, cont'd

Static SQL (most efficient) doesn't vary between MAXI and MINI

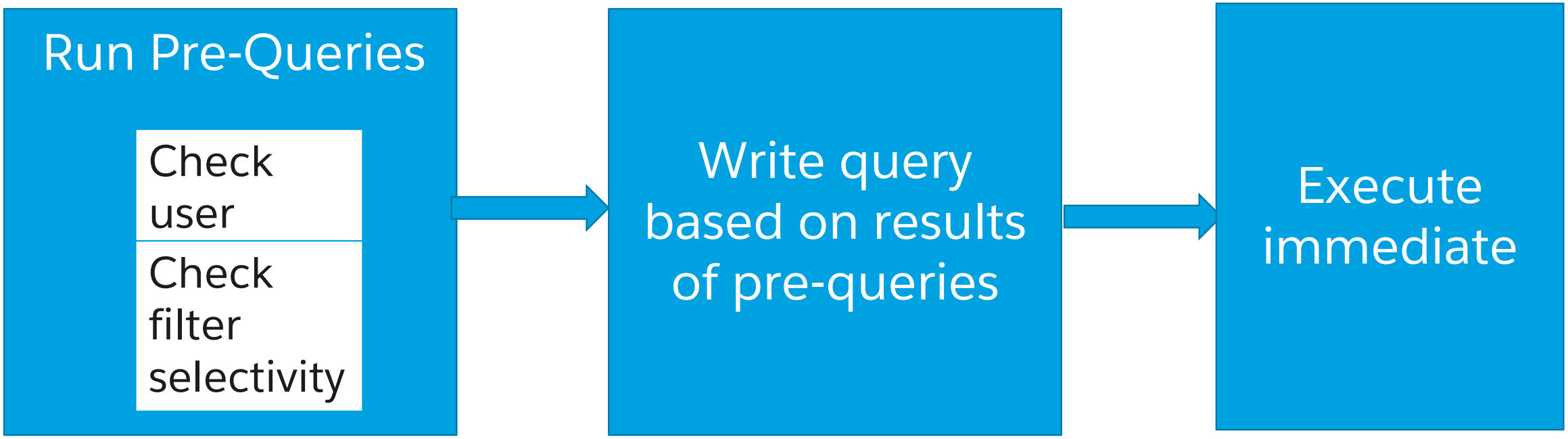
Dynamic SQL still doesn't have better DB statistics for a given eid value

"Learning optimizer" window inaccurate at predicting next eid value and best plan for that eid value, especially when millions of eids

Possible solutions:

- Materialized Views for each enterprise, but there may be millions – modify the schema each time you add an enterprise??
- Materialized Views for just the MAXIs, but app semantics may not permit it

APP-only possible solutions, cont'd



Application executes 3 stages for a request

From: Weissman, C. D. and Bobrowski, S. (2009). The design of the force.com multi-tenant internet application development platform. In Proceedings of the 35th SIGMOD 2009

Possible pre-queries and generated query

How many rows in each table for this eid? (already cached?)

What % of those rows can the current user view?

What's the selectivity of each predicate within this eid?

What join methods are best for this query?

Run limit queries to determine correlations, etc.

Select … from … where …
using HINTs or stored EXPLAIN plan to exploit

- index on specific predicate on T1,
- Merge join (T1 merge join T2) as the outer with (T3 merge join sort(T4))

Case APP-DB

“If I had an optimizer…
I’d optimize in the morning,
I’d optimize in the evening,
all over this land …”

Invention required…

We can use all the techniques from APP-only case plus more

How specific to a given application(s) should/could a DB engine be?

Challenges and Possible Solutions for APP-DB

Challenges for APP-DB case	Possible APP-DB Solutions
DB Statistics are Table Level	1. Build in DB knowledge of Enterprise_Key, similar to Primary Key; track in optimizer and use 2. below 2. Per Enterprise statistics per table

Challenges and Possible Solutions for APP-DB

Challenges for APP-DB case	Possible APP-DB Solutions
Full Table scans NEVER useful	1. Access method support (e.g. index) for access by specific Enterprise_Key, similar to Primary Key; leverage in optimizer, and 2. Use per Enterprise statistics per table

Challenges and Possible Solutions for APP-DB

Challenges for APP-DB case	Possible APP-DB Solutions
Static SQL (most efficient) doesn't vary between MAXI and MINI	1. Adopt pre-queries in APP and construct different queries/hints depending on enterprise statistics, reverting to dynamic SQL 2. Extend optimizer to create choice of different query plans, selected at runtime. E.g. MINI plan, MIDI plan, MAXI plan.

Challenges and Possible Solutions for APP-DB

Challenges for APP-DB case	Possible APP-DB Solutions
“Learning optimizer” window is enterprise-agnostic	1. Create enterprise-aware learning optimizer, determining best plans per enterprise per query, and possibly per user/role 2. Store and leverage these, depending on enterprise and user/role

Summary and further thoughts

Multi-tenant query optimization is a vision in progress

Are there new table access methods or join methods whose cost is independent of enterprise size?

Many possible solutions easier or only possible if APP is static and known to DB engine developer

Lots of classic ideas can be adapted to multi-tenancy

Many possible solutions work well for some queries but not others, e.g. works for single table request but not 11-way joins


Thank You