Carnegie Mellon University

OPTIMIZE!

Database Query Optimization

IBM Starburst Query Rewriter + Optimizer

ERRATA

Charles Bachmann was the 8^{th} Turing Award winner in 1973, not the 3^{rd}.

The number of different join orderings for an ***n***-way binary join is **$(n\text{-}1)! \times C(n\text{-}1)$**, where **$C(n\text{-}1)$** is the **$(n\text{-}1)^{th}$** Catalan number

→ ***n*!** different orders of leaf nodes (original relations)

→ **$C(n\text{-}1)$** possible shapes of a full binary tree with ***n*** leaves

Send Corrections: db-mistakes@cs.cmu.edu

Source: Alexey Goncharuk

LAST CLASS

System R had the first cost-based query optimizer

→ Used dynamic programming to choose optimal join ordering.

System R selects each table's access method before the join ordering.

→ It is better to choose a table's access method in conjunction with the join method.

DATABASE TRENDS IN LATE 1980s

Object-Oriented Databases

→ Emerging applications with data that did not easily fit into the relational model.
→ See object-relational impedance mismatch.

Active Databases

→ Event-driven architecture where the DBMS automatically responds to internal and external conditions.
→ See triggers.

HISTORY OF QUERY OPTIMIZERS

Choice #1: Heuristics
→ INGRES (1970s), Oracle (until mid 1990s)

Choice #2: Heuristics + Cost-based Join Search
→ System R (1970s), early IBM DB2

Choice #3: Stratified Search
→ IBM Starburst (late 1980s), now IBM DB2 + Oracle

Choice #4: Unified Search
→ Volcano/Cascades (early 1990s), now MSSQL + Orca

Choice #5: Randomized Search
→ Academics in the 1980s, current Postgres

HISTORY OF QUERY OPTIMIZERS

Choice #1: Heuristics
→ INGRES (1970s), Oracle (until mid 1990s)

Choice #2: Heuristics + Cost-based Join Search
→ System R (1970s), early IBM DB2

Optimizer Generators

Choice #3: Stratified Search
→ IBM Starburst (late 1980s), now IBM DB2 + Oracle

Choice #4: Unified Search
→ Volcano/Cascades (early 1990s), now MSSQL + Orca

Choice #5: Randomized Search
→ Academics in the 1980s, current Postgres

OPTIMIZER GENERATORS

Framework to allow a DBMS implementer to write the 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.

The implementation of the optimizer's pattern matching method and transformation rules can be independent of its search strategy.

OPTIMIZER GENERATORS

Choice #1: Stratified Search

→ Planning is done in multiple stages (heuristics then cost-based search).

→ Examples: Starburst, CockroachDB

Choice #2: Unified Search

→ Perform query planning all at once.

→ Examples: Volcano/Cascades, OPT++, SQL Server

ON THE CORRECT AND COMPLETE ENUMERATION OF THE CORE SEARCH SPACE
SIGMOD 2013

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 many transformations, so it makes heavy use of memoization to reduce redundant work.

TODAY'S AGENDA

IBM Starburst
Relational Calculus
Query Rewriting
Plan Enumeration

IBM DATABASE HISTORY

System R (1975-1979)

R* (1979)

SQL/DS (1981)

DB2 (1983)

Starburst (1985)

IBM STARBURST

DBMS designed to allow developers to extend the system to support new workloads and data sets without rewriting.

Supported extensions
→ Storage/Access Methods
→ Data types, user-defined functions
→ Query operators

Adding new runtime functionality requires changes in the query optimizer.

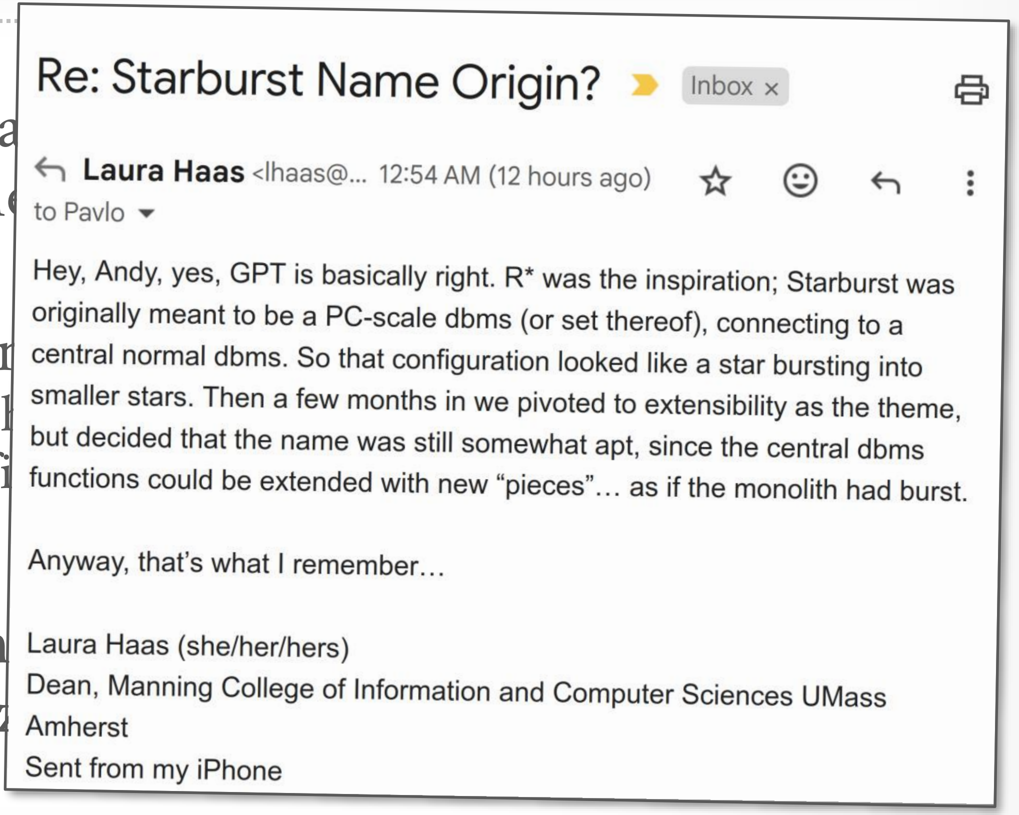
EXTENSIBLE QUERY PROCESSING IN STARBURST
SIGMOD RECORD 1989

IBM STARBURST

DBMS designed to a
system to support ne
without rewriting.

Supported extensior
→ Storage/Access Metl
→ Data types, user-defi
→ Query operators

Adding new runtim
in the query optimiz


Re: Starburst Name Origin? Inbox ×

Laura Haas <lhaas@... 12:54 AM (12 hours ago)
to Pavlo

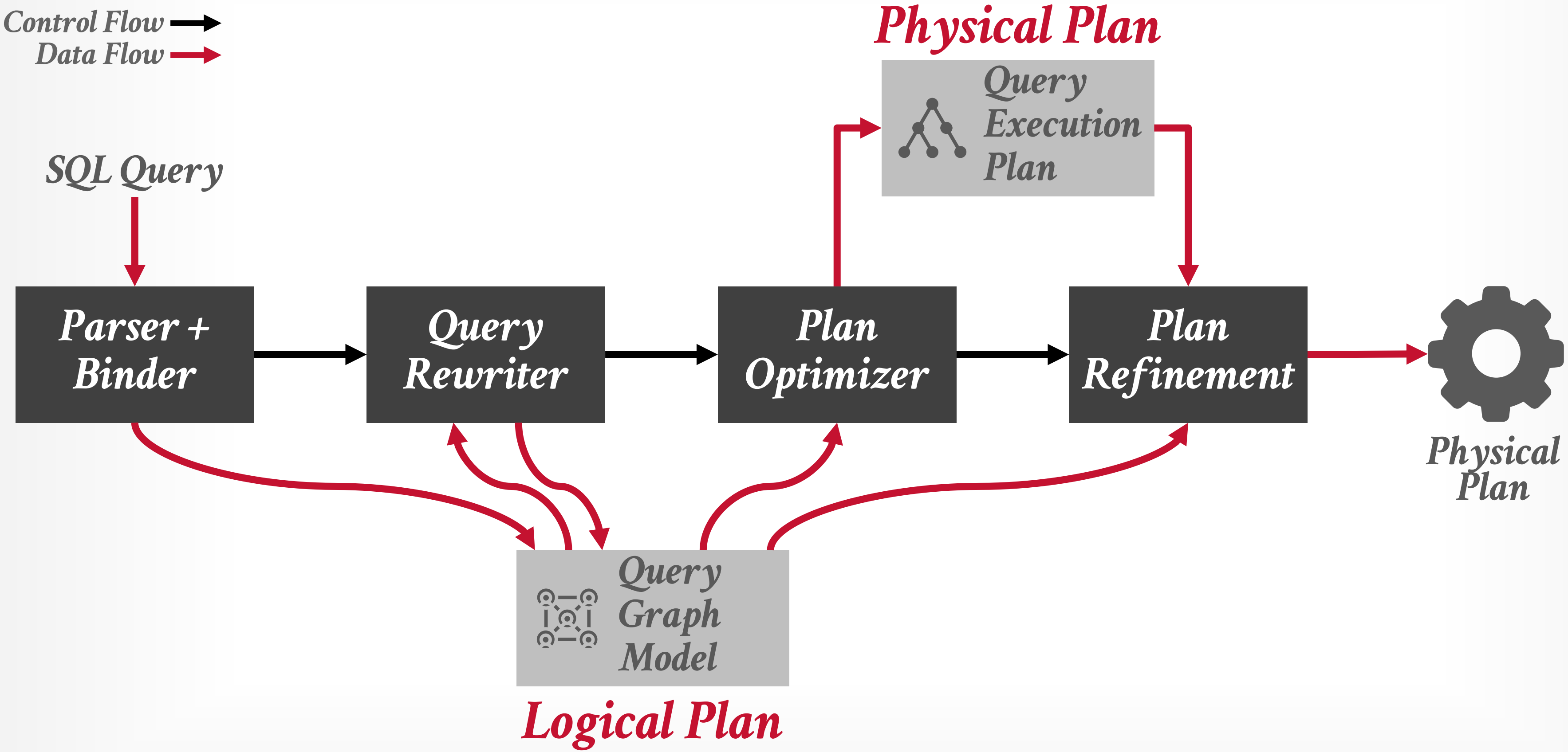
Hey, Andy, yes, GPT is basically right. R* was the inspiration; Starburst was originally meant to be a PC-scale dbms (or set thereof), connecting to a central normal dbms. So that configuration looked like a star bursting into smaller stars. Then a few months in we pivoted to extensibility as the theme, but decided that the name was still somewhat apt, since the central dbms functions could be extended with new "pieces"… as if the monolith had burst.

Anyway, that's what I remember…

Laura Haas (she/her/hers)
Dean, Manning College of Information and Computer Sciences UMass Amherst
Sent from my iPhone


EXTENSIBLE QUERY PROCESSING IN STARBURST
SIGMOD RECORD 1989

STARBURST: QUERY OPTIMIZER PIPELINE



OBSERVATION

We made a big deal about using a **declarative language** instead of a **procedural language** to query a database.

But relational algebra is procedural!
→ It defines an ordering of steps to execute a query.

Starburst's internal representation (query graph model) is based on **relational calculus**…

TUPLE RELATIONAL CALCULUS

A nonprocedural query language, where each query is of the form: $\{ t \mid P(t) \}$

→ It is the set of all tuples t such that predicate P is true for t

Definitions:

→ t is a tuple variable

→ $t[A]$ denotes the value of tuple t on attribute A

→ $t \in r$ denotes that tuple t is in relation r

→ P is a formula similar to that of the predicate calculus

Source: Database Systems Concepts

TUPLE RELATIONAL CALCULUS

Retrieve the id and salary for all employees whose salary is greater than $50,000.

Relational Calculus:

→ $\{ \boldsymbol{t} \mid \exists\, \boldsymbol{s} \in \text{employees} ($
$\boldsymbol{t}[\text{id}] = \boldsymbol{s}[\text{id}] \land$
$\boldsymbol{t}[\text{salary}] = \boldsymbol{s}[\text{salary}] \land$
$\boldsymbol{s}[\text{salary}] > 50000) \}$

Relational Algebra:

→ $\Pi_{\text{id,salary}}\ (\ \sigma_{\text{salary}>50000}\ (\text{employees})\ )$

Source: Database Systems Concepts

QUERY GRAPH MODEL

Internal representation of queries designed to reduce the complexity of query optimization.

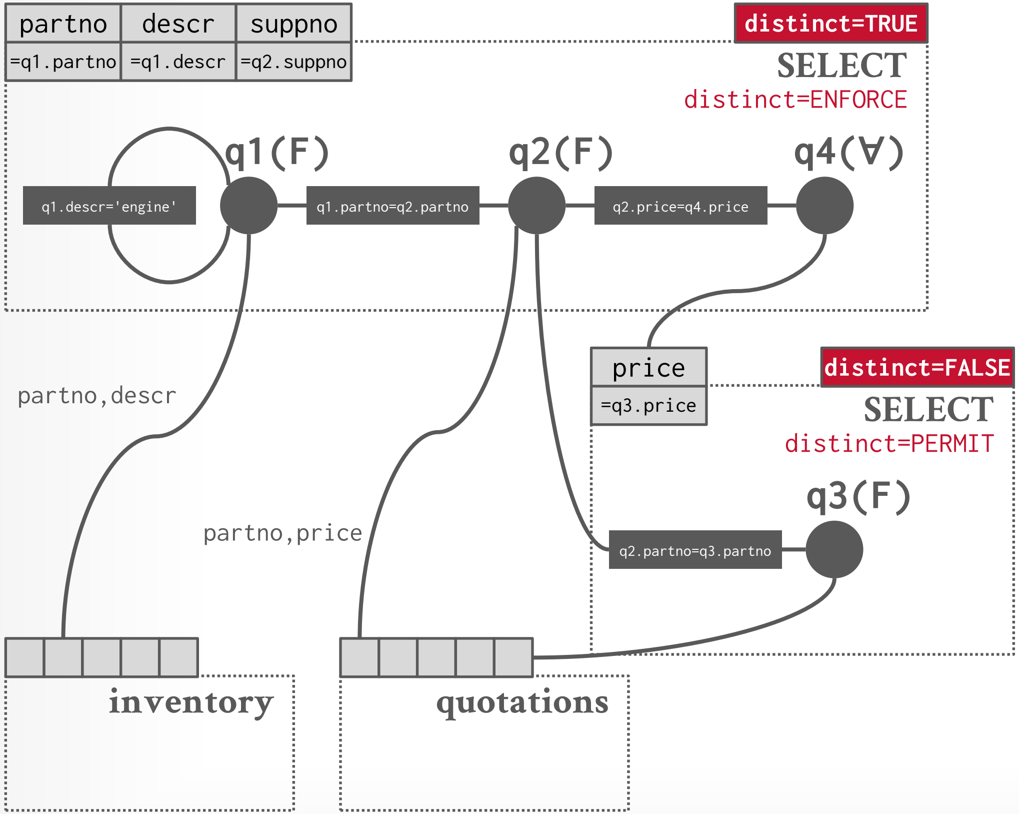
→ In-memory cache of catalog information on tables, columns, and predicates and their relationships.
→ Based on tuple relational calculus.

QGM describes input/output tables and their relationships in a query rather than operations.

→ **Body**: Quantifiers that perform an operation on inputs
→ **Head**: Meta-data about outputs and properties

EXTENSIBLE/RULE BASED QUERY REWRITE OPTIMIZATION IN STARBURST
SIGMOD RECORD 1992

QUERY GRAPH MODEL



Get the suppliers and parts information for which the supplier's price is less than that of all other suppliers.

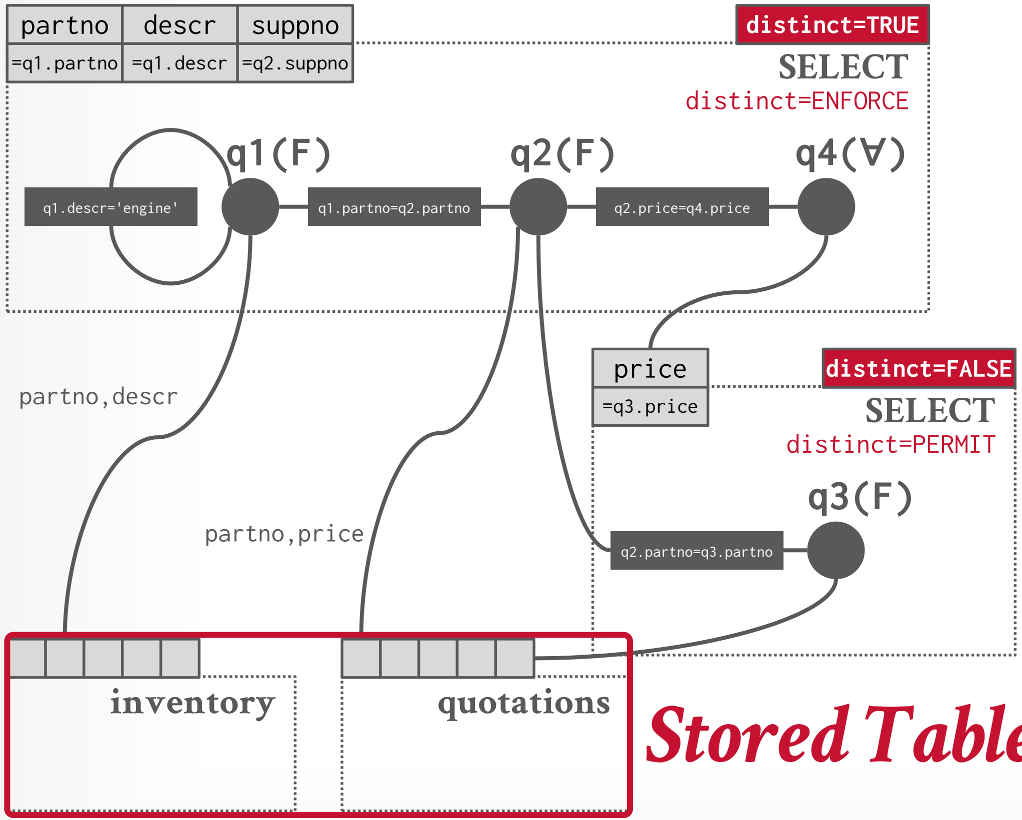
```
SELECT DISTINCT q1.partno, q1.descr, q2.suppno
  FROM inventory AS q1, quotations AS q2
 WHERE q1.partno = q2.partno
   AND q1.descr = 'engine'
   AND q1.price <= ALL(
     SELECT q3.price
       FROM quotations AS q3
      WHERE q2.partno = q3.partno );
```

Source: Hamid Pirahesh

QUERY GRAPH MODEL

Get the suppliers and parts information for which the supplier's price is less than that of all other suppliers.

```
SELECT DISTINCT q1.partno, q1.descr, q2.suppno
  FROM inventory AS q1, quotations AS q2
 WHERE q1.partno = q2.partno
   AND q1.descr = 'engine'
   AND q1.price <= ALL(
     SELECT q3.price
       FROM quotations AS q3
      WHERE q2.partno = q3.partno );
```



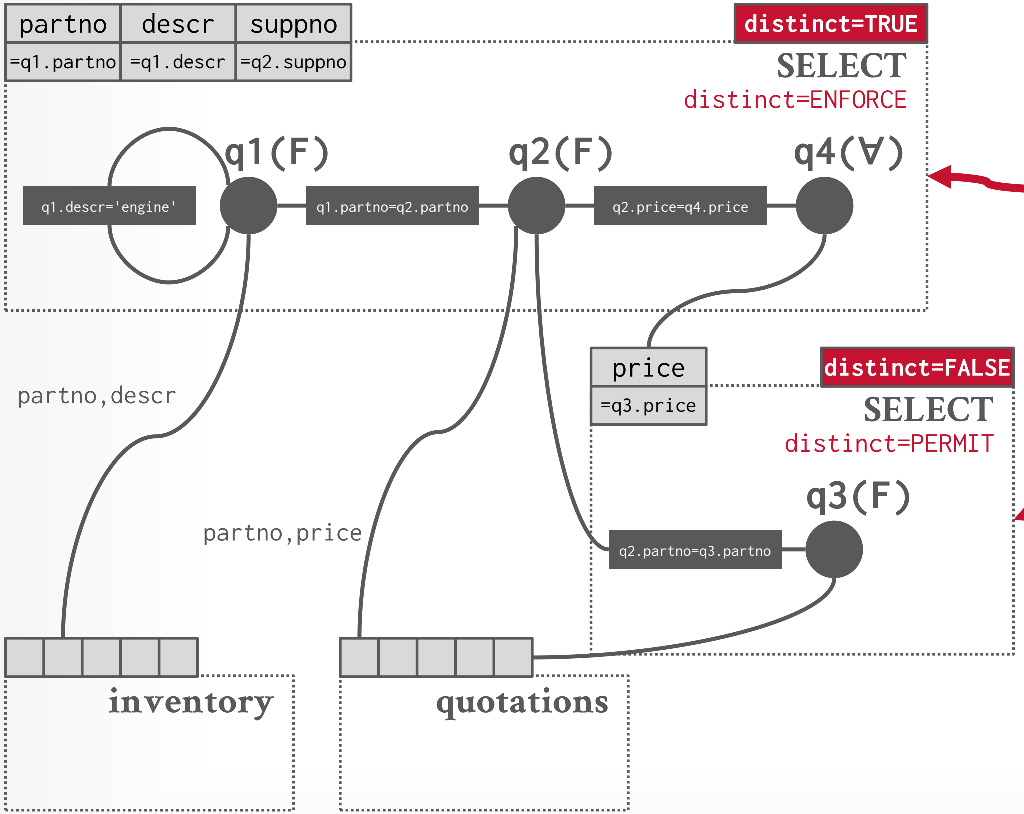
Stored Tables

Source: Hamid Pirahesh

QUERY GRAPH MODEL

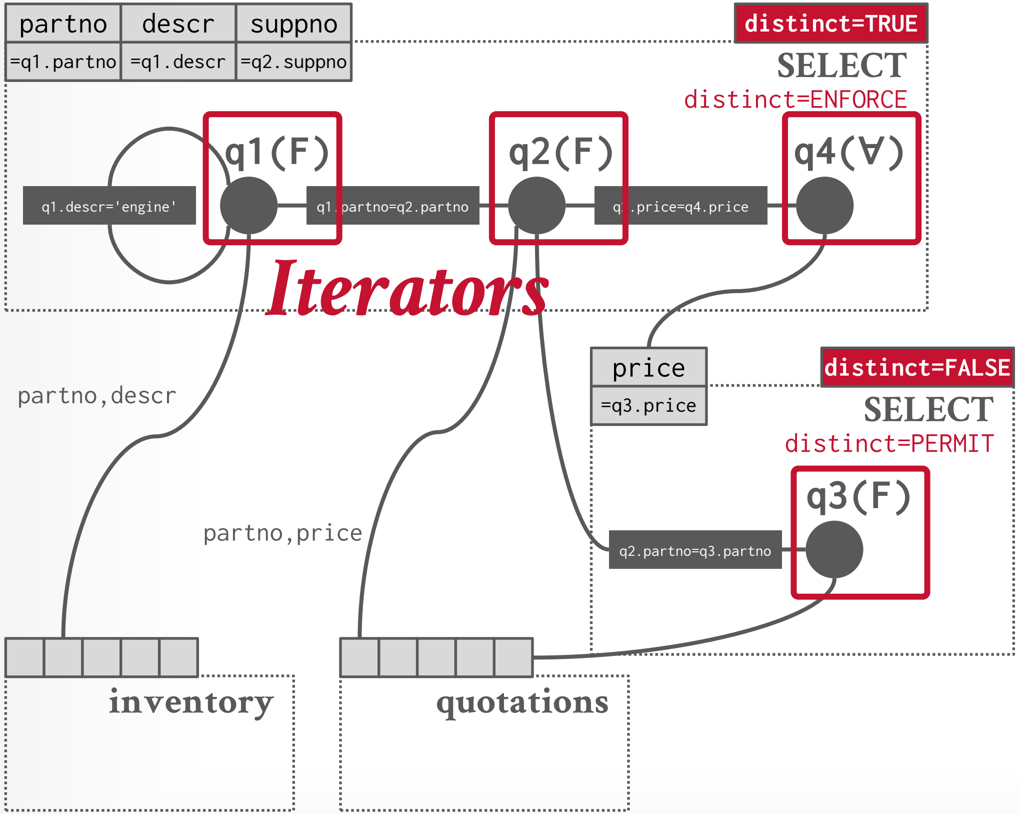
Get the suppliers and parts information for which the supplier's price is less than that of all other suppliers.

```
SELECT DISTINCT q1.partno, q1.descr, q2.suppno
  FROM inventory AS q1, quotations AS q2
 WHERE q1.partno = q2.partno
   AND q1.descr = 'engine'
   AND q1.price <= ALL(
     SELECT q3.price
       FROM quotations AS q3
      WHERE q2.partno = q3.partno );
```



Source: Hamid Pirahesh

QUERY GRAPH MODEL



Get the suppliers and parts information for which the supplier's price is less than that of all other suppliers.

```
SELECT DISTINCT q1.partno, q1.descr, q2.suppno
  FROM inventory AS q1, quotations AS q2
 WHERE q1.partno = q2.partno
   AND q1.descr = 'engine'
   AND q1.price <= ALL(
     SELECT q3.price
       FROM quotations AS q3
      WHERE q2.partno = q3.partno );
```

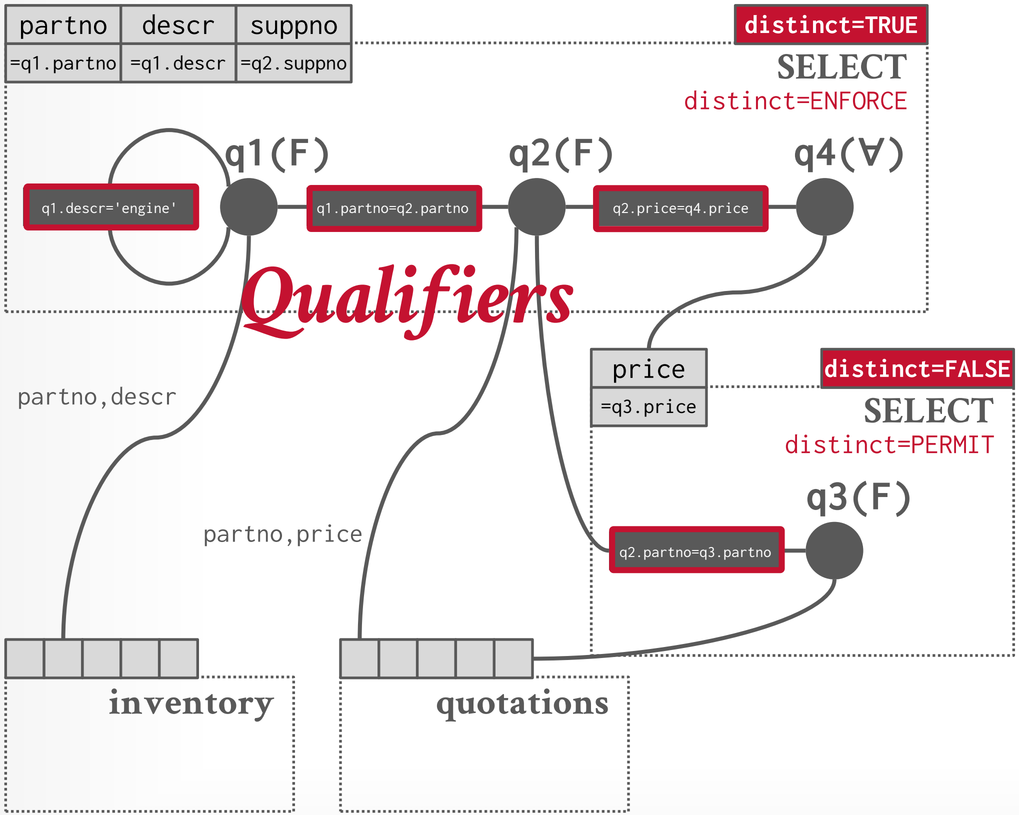
Iterators

→ SetFormers: **F**

→ Quantifiers: **∀**, **∃**

Source: Hamid Pirahesh

QUERY GRAPH MODEL



Get the suppliers and parts information for which the supplier's price is less than that of all other suppliers.

```
SELECT DISTINCT q1.partno, q1.descr, q2.suppno
  FROM inventory AS q1, quotations AS q2
 WHERE q1.partno = q2.partno
   AND q1.descr = 'engine'
   AND q1.price <= ALL(
     SELECT q3.price
       FROM quotations AS q3
      WHERE q2.partno = q3.partno );
```

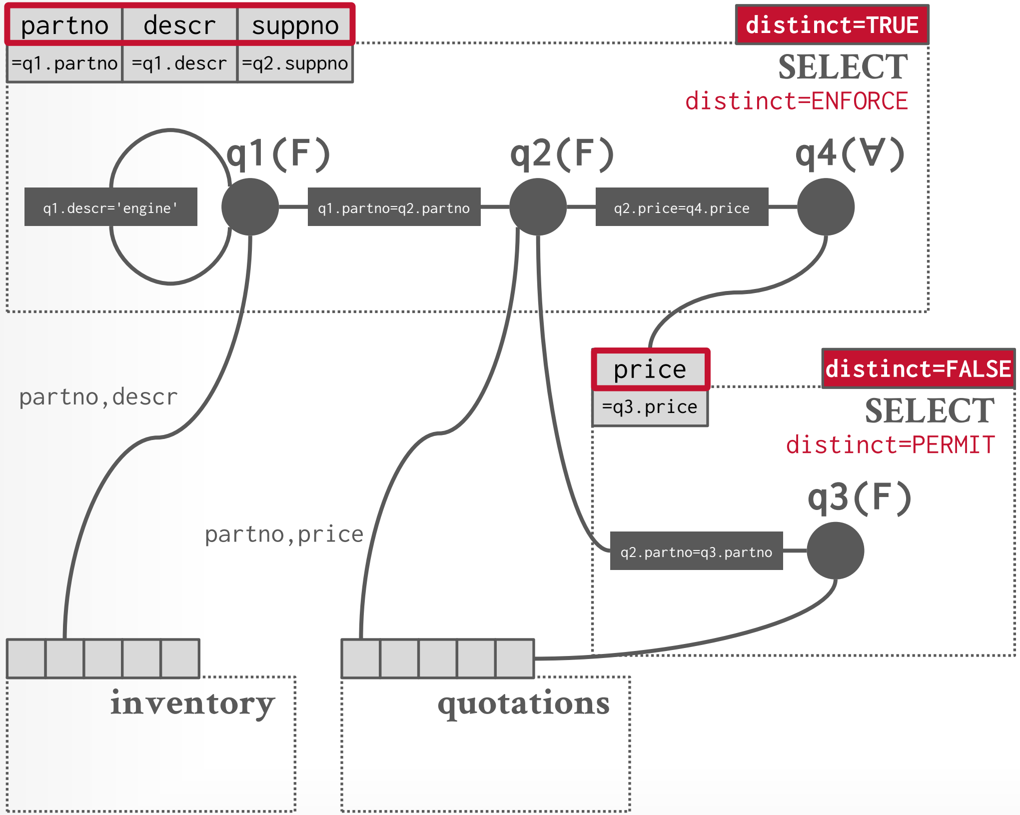
Iterators

→ SetFormers: **F**

→ Quantifiers: **∀**, **∃**

Source: Hamid Pirahesh

QUERY GRAPH MODEL



Get the suppliers and parts information for which the supplier's price is less than that of all other suppliers.

```
SELECT DISTINCT q1.partno, q1.descr, q2.suppno
  FROM inventory AS q1, quotations AS q2
 WHERE q1.partno = q2.partno
   AND q1.descr = 'engine'
   AND q1.price <= ALL(
     SELECT q3.price
       FROM quotations AS q3
      WHERE q2.partno = q3.partno );
```

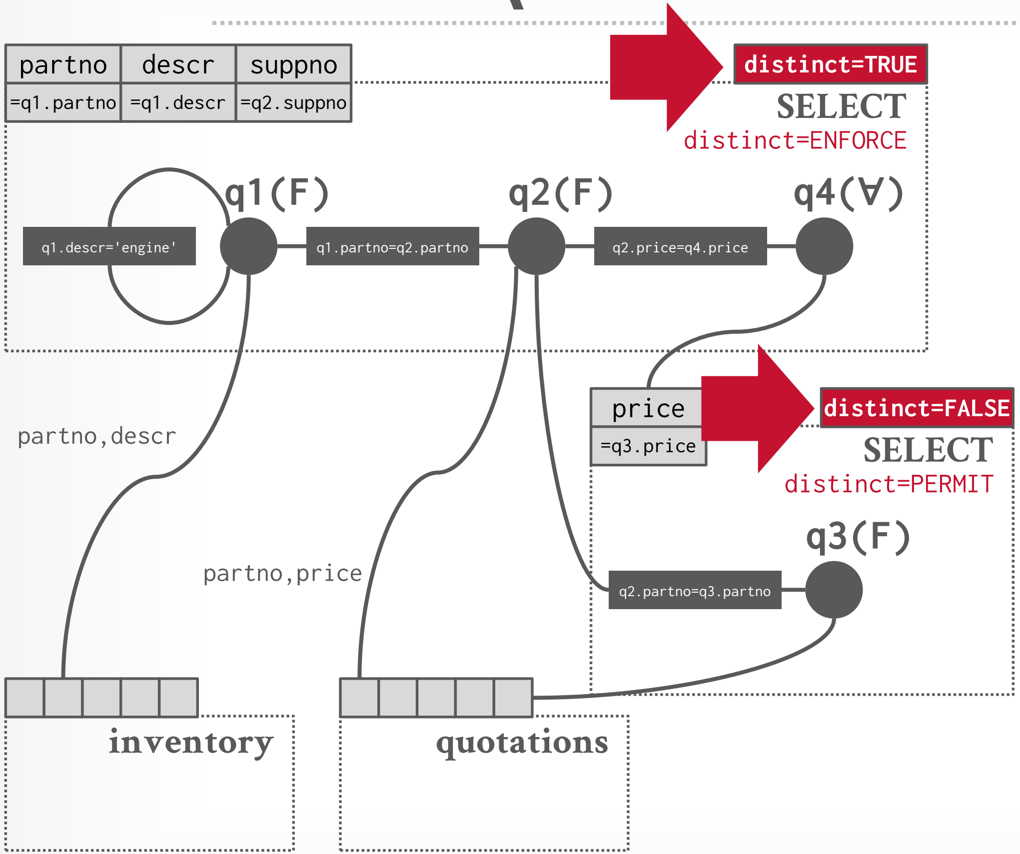
Iterators

→ SetFormers: **F**

→ Quantifiers: **∀**, **∃**

Source: Hamid Pirahesh

QUERY GRAPH MODEL



Get the suppliers and parts information for which the supplier's price is less than that of all other suppliers.

```
SELECT DISTINCT q1.partno, q1.descr, q2.suppno
  FROM inventory AS q1, quotations AS q2
 WHERE q1.partno = q2.partno
   AND q1.descr = 'engine'
   AND q1.price <= ALL(
     SELECT q3.price
       FROM quotations AS q3
      WHERE q2.partno = q3.partno );
```

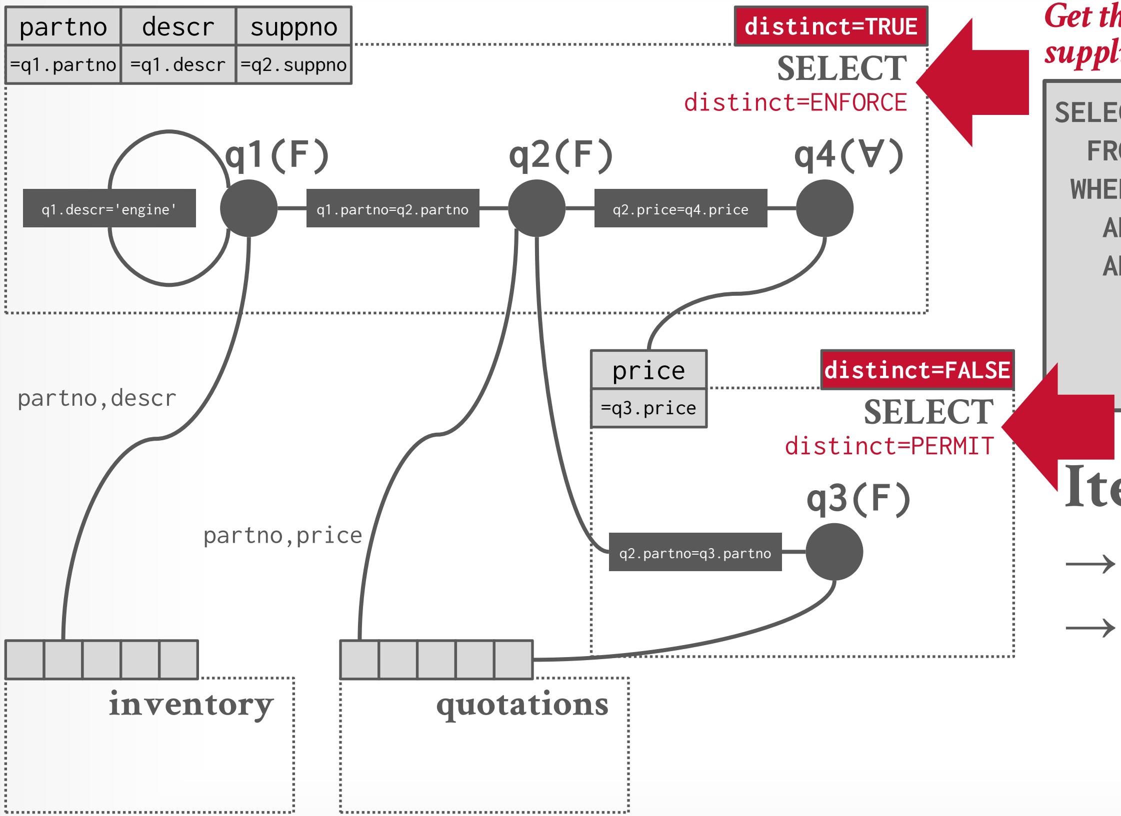
Iterators

→ SetFormers: **F**

→ Quantifiers: **∀**, **∃**

Source: Hamid Pirahesh

QUERY GRAPH MODEL



Get the suppliers and parts information for which the supplier's price is less than that of all other suppliers.

```
SELECT DISTINCT q1.partno, q1.descr, q2.suppno
  FROM inventory AS q1, quotations AS q2
 WHERE q1.partno = q2.partno
   AND q1.descr = 'engine'
   AND q1.price <= ALL(
     SELECT q3.price
       FROM quotations AS q3
      WHERE q2.partno = q3.partno );
```

Iterators

→ SetFormers: **F**

→ Quantifiers: **∀**, **∃**

Source: Hamid Pirahesh

OBSERVATION

The initial QGM produced by the parser/binder is guaranteed to be valid but will split nested subqueries into separate **SELECT** operators (boxes).

But removing subqueries will require the optimizer to reason across multiple boxes.

Goal: Whenever possible, convert a multi-SELECT QGM to a new QGM with a single SELECT operator.

IBM STARBURST: REWRITER

Rule-based rewriter to change one QGM representation into another QGM.

→ Transform "procedural" queries into an equivalent query that is more understandable by the optimizer.

→ Apply transformations that are known to always be a good idea.

Does not need to consider plan costs at this stage.

EXTENSIBLE/RULE BASED QUERY REWRITE OPTIMIZATION IN STARBURST
SIGMOD RECORD 1992

REWRITE RULES

High-level specifications of legal QGM alternatives.

Each rule is defined in terms of a matching condition function and an action function.

→ Primitives for manipulating query graphs
→ Nested rule execution
→ Controllable rule evaluation ordering
→ Termination Guarantees

Keep track of rules applied to enable tracing the origin of a query plan.

RULE ENGINE

Control Strategies
- → Sequential (process rules sequentially)
- → Priority (higher priorities are evaluated first)
- → Statistical (next rule chosen randomly from a user-defined distribution)

Given a budget for search. When budget exhausted, rule processing stops at a consistent QGM.

EXAMPLE: SELECT MERGE

```
if (in a SELECT box (upper)
    a quantifier has type F
   AND ranges over a SELECT box (lower)
   AND no other quantifier ranges over lower
   AND (
      upper.head.distinct = TRUE
     OR
      upper.body.distinct = PERMIT
     OR
      lower.body.distinct != ENFORCE

) then {
   MERGE lower into upper
   if (lower.body.distinct = ENFORCE
      AND upper.body.distinct = != PERMIT) {
       upper.body.distinct = ENFORCE;
} }
```

```sql
CREATE VIEW iptv AS (
 SELECT DISTINCT itp.itemn, pur.vendn
   FROM itp JOIN pur
     ON itp.ponum = pur.ponum
  WHERE pur.odate > '2025'
);
```

```sql
SELECT itm.itmn, itpv.vendn
  FROM itm JOIN itpv
    ON itm.itemn = itpv.itemn
   AND item.itemn >= '01'
   AND item.itemn <= '20';
```

EXAMPLE: SELECT MERGE

```
if (in a SELECT box (upper)
    a quantifier has type F
    AND ranges over a SELECT box (lower)
    AND no other quantifier ranges over lower
    AND (
        upper.head.distinct = TRUE
      OR
        upper.body.distinct = PERMIT
      OR
        lower.body.distinct != ENFORCE
) then {
    MERGE lower into upper
    if (lower.body.distinct = ENFORCE
        AND upper.body.distinct = != PERMIT) {
         upper.body.distinct = ENFORCE;
} }
```

```sql
SELECT itm.itmn, itpv.vendn
  FROM itm JOIN (SELECT DISTINCT itp.itemn, pur.vendn
                   FROM itp JOIN pur
                     ON itp.ponum = pur.ponum
                  WHERE pur.odate > '2025') AS itpv
    ON itm.itemn = itpv.itemn
   AND item.itemn >= '01'
   AND item.itemn <= '20';
```

PLAN OPTIMIZATION

Convert a QGM into execution plan comprised of physical operators using rules.

Rules transform higher-level QGM "non-terminal" operations into "terminal" constructs.
→ Different than the rewriter rules.

Rules may produce multiple alternative constructs for the optimizer to evaluate to determine its cost.

EXTENSIBLE QUERY PROCESSING IN STARBURST
SIGMOD RECORD 1989

PLANNER RULE GRAMMAR

Rules construct new operators from base operators that operate on tables.

Specifying the conditions under which a rule is applicable is (usually) harder than specifying a rule's transformation.

Parameterized rules that allow for flexibility in what matches a rule.

STARBURST: LOLEPOP

LOw-LEvel Plan OPerator (LOLEPOP)

Database operator interpretable at runtime.

Extension of relational algebra operators that includes additional functionality

→ Examples: **ACCESS**, **STORE**, **SORT**, **SHIP**

Each LOLEPOP takes in one or more tables as inputs and produces a single table as its output.

→ Input tables can be stored tables or streams derived from the output of other LOLEPOPs.

Parameters can also specify "flavor" of a LOLEPOP.

GRAMMAR-LIKE FUNCTIONAL RULES FOR REPRESENTING QUERY OPTIMIZATION ALTERNATIVES
SIGMOD RECORD 1988

STARBURST: STAR

STrategy Alternative Rules (STAR)

High-level declarative specification of the legal strategies for executing a query.

Each STAR is a named object that defines one or more alternative definitions based one or more LOLEPOPs or other STARs.

→ Describe how to build higher-level constructs from primitive operators rather than transform primitive operators.

GRAMMAR-LIKE FUNCTIONAL RULES FOR REPRESENTING QUERY OPTIMIZATION ALTERNATIVES
SIGMOD RECORD 1988

PLAN PROPERTIES

Query plan meta-data the describes the characteristics of data and the worked performed by that plan's operators.

→ **Relational**: Tables and columns accessed
→ **Physical**: Tuple ordering, data location
→ **Estimated**: cardinalities, execution cost

The DBMS initially derives properties from base tables or access methods referenced in plan.

They are then altered by LOLEPOPs when they are added to a plan.

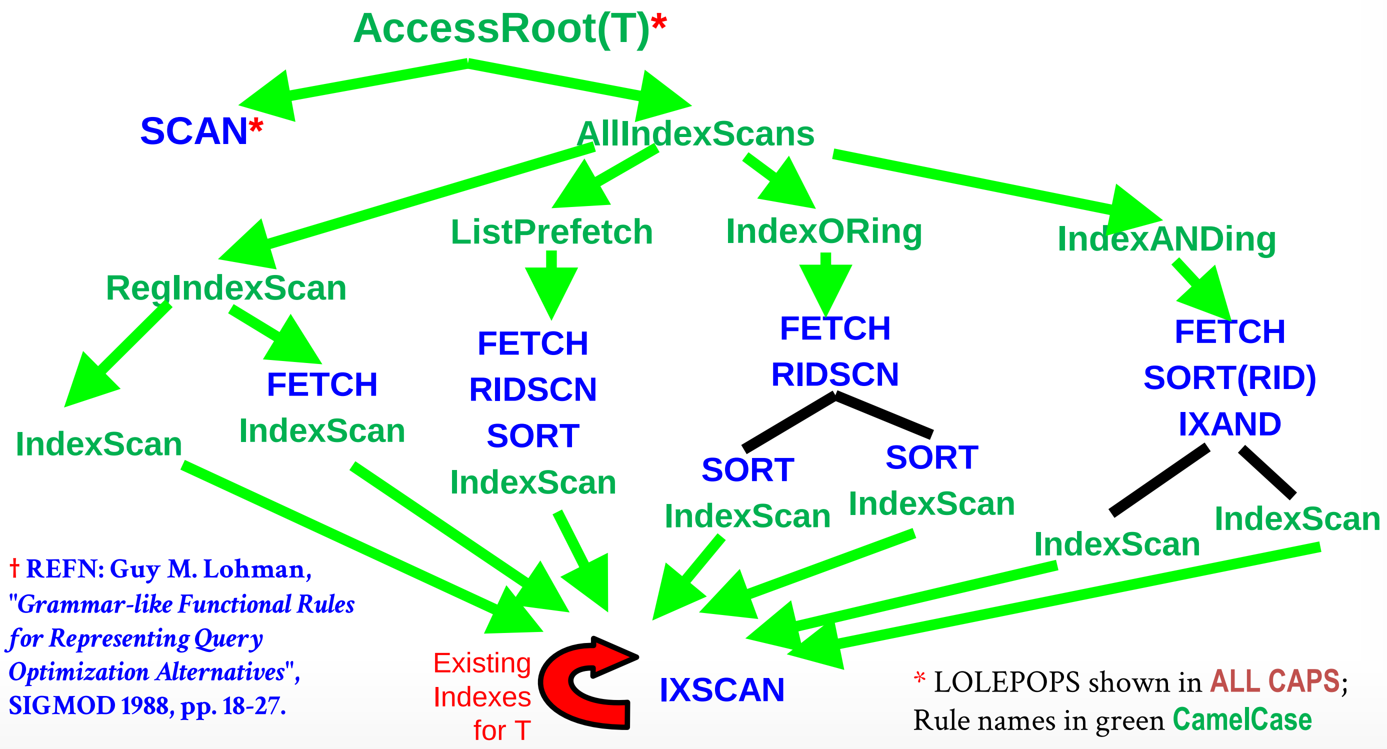
STARBURST: GLUE

Special STARs that find the cheapest plan satisfying the required properties for a query.

If necessary, Glue STARs may add LOLEPOPs to a plan to ensure they meet requirements.

Generation of Table Access Alternatives

- Rules specify one or more alternatives, like a grammar †
- Each alternative specifies a nesting of other rules or LOLEPOPs*
- Can have iterators (e.g. all indexes for a table – see red arrow)

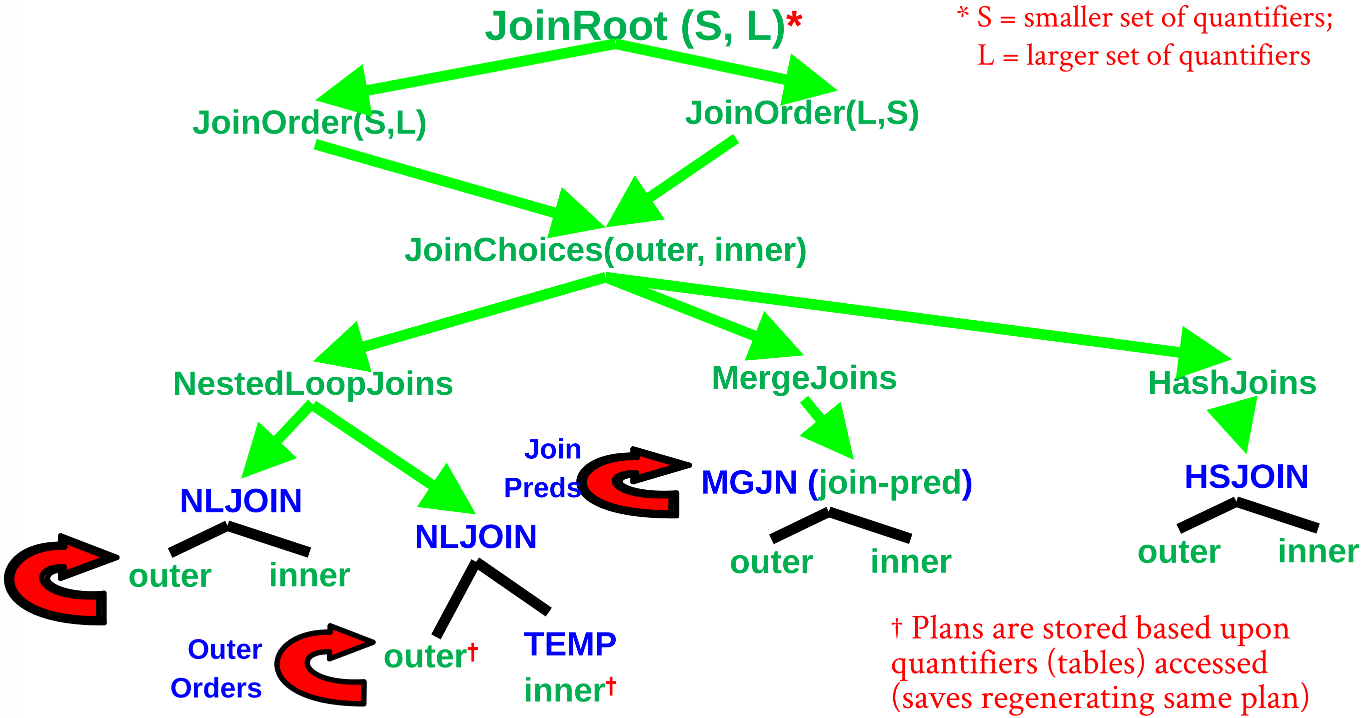


† REFN: Guy M. Lohman, *"Grammar-like Functional Rules for Representing Query Optimization Alternatives"*, SIGMOD 1988, pp. 18-27.

* LOLEPOPS shown in **ALL CAPS**; Rule names in green **CamelCase**

Source: Guy Lohman

Generation of Join Alternatives



* S = smaller set of quantifiers;
L = larger set of quantifiers

† Plans are stored based upon quantifiers (tables) accessed (saves regenerating same plan)

NOTE: All rules were **interpreted** (read as **data**) in Starburst, but **compiled** in DB2 LUW!

REFN: **Mavis Lee, Johann Christoph Freytag, Guy Lohman,**
"Implementing an Interpreter for Functional Rules in a Query Optimizer", VLDB 1988: 218-229

Source: Guy Lohman

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: Exhaustion

→ Stop when there are no more enumerations of the target plan. Usually done per sub-plan/group.

Approach #4: Transformation Count

→ Stop after a certain number of rules/transformations have been considered.

PARTING THOUGHTS

IBM Starburst is one of the first query optimizers that represents query plans in a higher-level form to make it easier to construct rules.

Also one of the first to perform rewriting before optimizing query in cost-based search.

Many other interesting aspects in Starburst/DB2's optimizer that we will discuss later..

PARTING THOUGHTS

IBM Star
that repre
make it e

Also one
optimizi

Many oth
optimize

Four DB2 Code Bases?

Many years ago I worked on IBM DB2 and so I occasionally get the question, "how the heck could you folks possibly have four relational database management system code bases?" Some go on to argue that a single code base would have been much more efficient. That's certainly true. And, had we moved to a single code base, that engineering resource efficiency improvement would have led to a very different outcome in the database wars. I'm skeptical on this extension of the argument but the question is an interesting one and I wrote up a more detailed answer than usually possible off the cuff.

James Hamilton

Disclaimer: The opinions expressed here are my own and do not necessarily represent those of current or past employers.

Recent Comments

- Raffaele on David Patterson Retires After 40 Years
- James Hamilton on David Patterson Retires After 40 Years
- James Hamilton on Pat Selinger
- Mariana Carvalho on Pat Selinger
- Raffaele Santopaolo on David Patterson Retires After 40 Years
- James Hamilton on Seagate HAMR
- Tom Davies on Seagate HAMR
- Matt on Seagate HAMR

NEXT CLASS

Unified Query Optimizers
→ Exodus
→ Volcano