



Chaudhuri: An Overview of Query Optimization in Relational Systems

Nick Taylor

University of Pennsylvania

October 7, 2005

Two 'Great Works'

- Codd, "A Relational Model of Data for Large Shared Data Banks," 1970.
- Selinger *et al.*, "Access path selection in a relational database management system," 1979.
- This paper presents a more modern take on a field created by these two ideas.

Motivation for Relational Model

Primitive data models: in-memory records, serialized files, early databases

- Model relationships using pointers between records
- Data layout becomes problematic
 - ◆ Programmers need to be familiar with it
 - ◆ Programs become tied to it

The Relational Model

Codd's idea: simple, logical, value-based data model

- Data stored in tuples with named attributes
- Sets of tuples make up relations
- Makes programs independent of data representation
- Writing queries becomes simpler
- But what about efficiency?

Relational Calculus

- Use subset of first-order logic as query language
- Suppose relations $Teaches(prof, course)$ and $Takes(student, course)$
- $\{\langle s, p \rangle \mid \exists c Teaches(p, c) \wedge Takes(s, c)\}$
- Query answering takes time polynomial in the size of the database
- Basis for all query languages (SQL, datalog)
- Cannot compute transitive closure

Fixed Point Recursion

- Suppose relation $P(\text{parent}, \text{child})$
- $\phi(R) = \{ \langle a, d \rangle \mid R(a, d) \vee \exists p R(a, p) \wedge R(p, d) \}$
- $\phi(P)$ gives parent relation again
- $\phi(\phi(P))$ gives parents and grandparents
- Least fixed point of ϕ is the ancestor relation
- Query answering still polynomial in size of the database (number of tuples that can be derived from constants in database is polynomial)

Relational Algebra

Basic operations for executing queries

- σ selection
- π projection
- ρ rename
- \times cross product
- \bowtie join - σ and \times
- \cup union
- \cap intersection
- $-$ difference

Relational Algebra and Calculus

- Relational algebra operators can express any 'safe' relational calculus (i.e. SQL) query
 - ◆ $\{ \langle s, t \rangle \mid \exists c \text{ Teaches}(t, c) \wedge \text{Takes}(s, c) \}$
 - ◆ $\pi_{student, teacher}(\text{Teaches} \bowtie \text{Takes})$
- Can partition operators by hashing or ranging
- Correspond to physical operations in a database
- Therefore relational queries are parallelizable and scalable(!)

Query Optimization: Problems

- Many equivalences between relational algebra operations
 - ◆ $(A \bowtie B) \bowtie C \equiv A \bowtie (B \bowtie C)$
 - ◆ $\sigma_{\theta} (A \bowtie B) \equiv \sigma_{\theta_1} ((\sigma_{\theta_2} A) \bowtie (\sigma_{\theta_3} B))$
- Different physical operators for relational algebra operators
- Goal: find most efficient algebraic expression and physical operator implementation for a query

Query Optimization: Example (1)

Consider schema

- $Teaches(prof, course)$
- $Takes(student, course)$
- $Email(student, email)$

and query

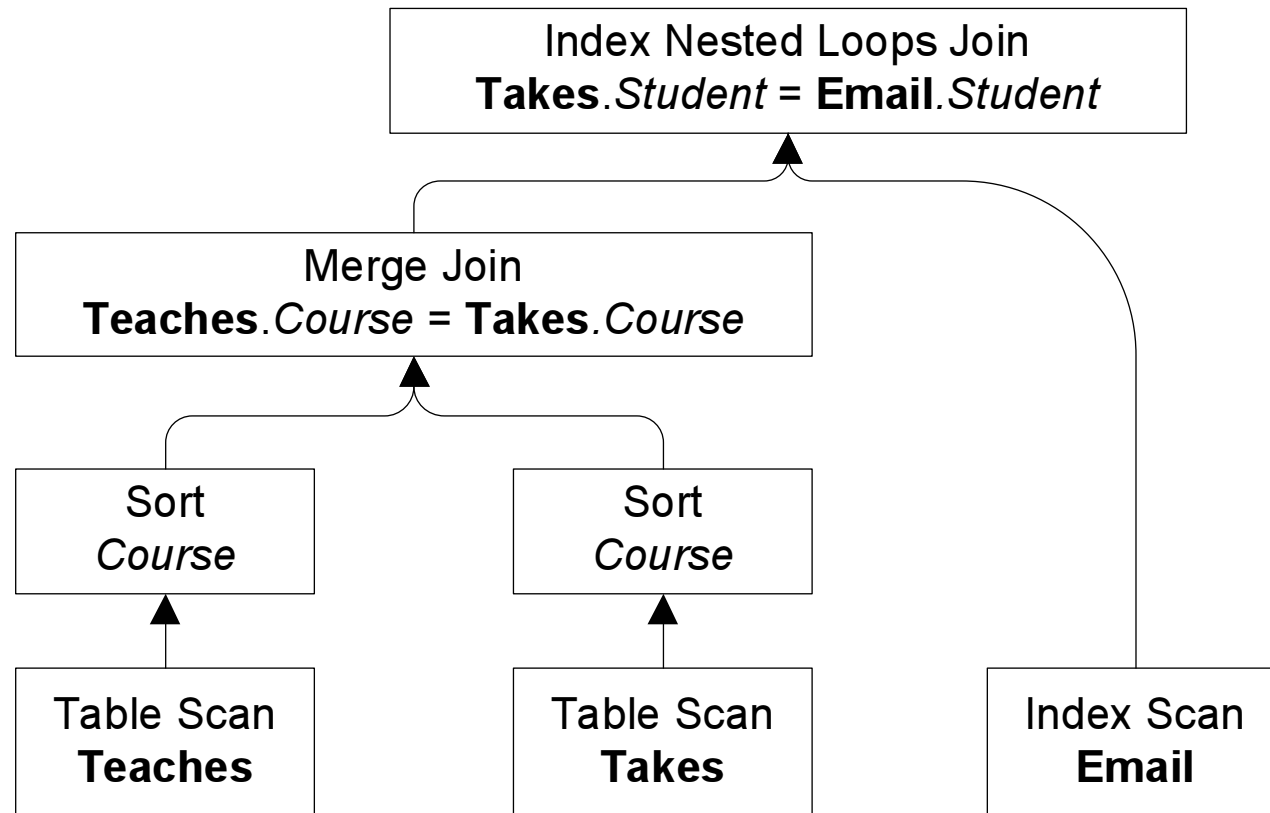
$$\{ \langle p, e \rangle \mid \exists s, c \ Teaches(p, c) \wedge Takes(s, c) \wedge Email(s, e) \}$$

with relational algebra translation

$$\pi_{prof, email} (Teaches \bowtie Takes \bowtie Email)$$

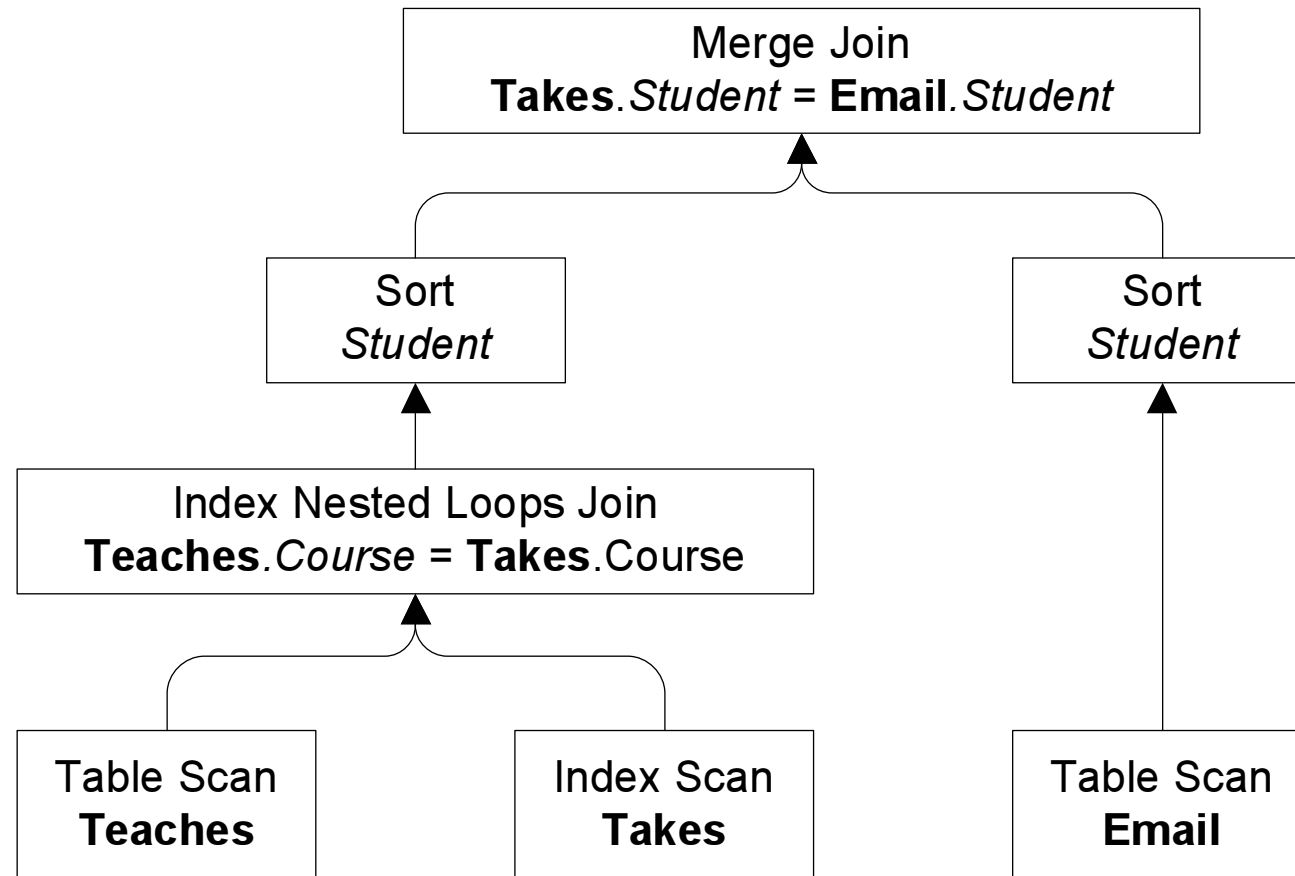
Query Optimization: Example (2)

This is one possible query plan:



Query Optimization: Example (3)

Here is another query plan:

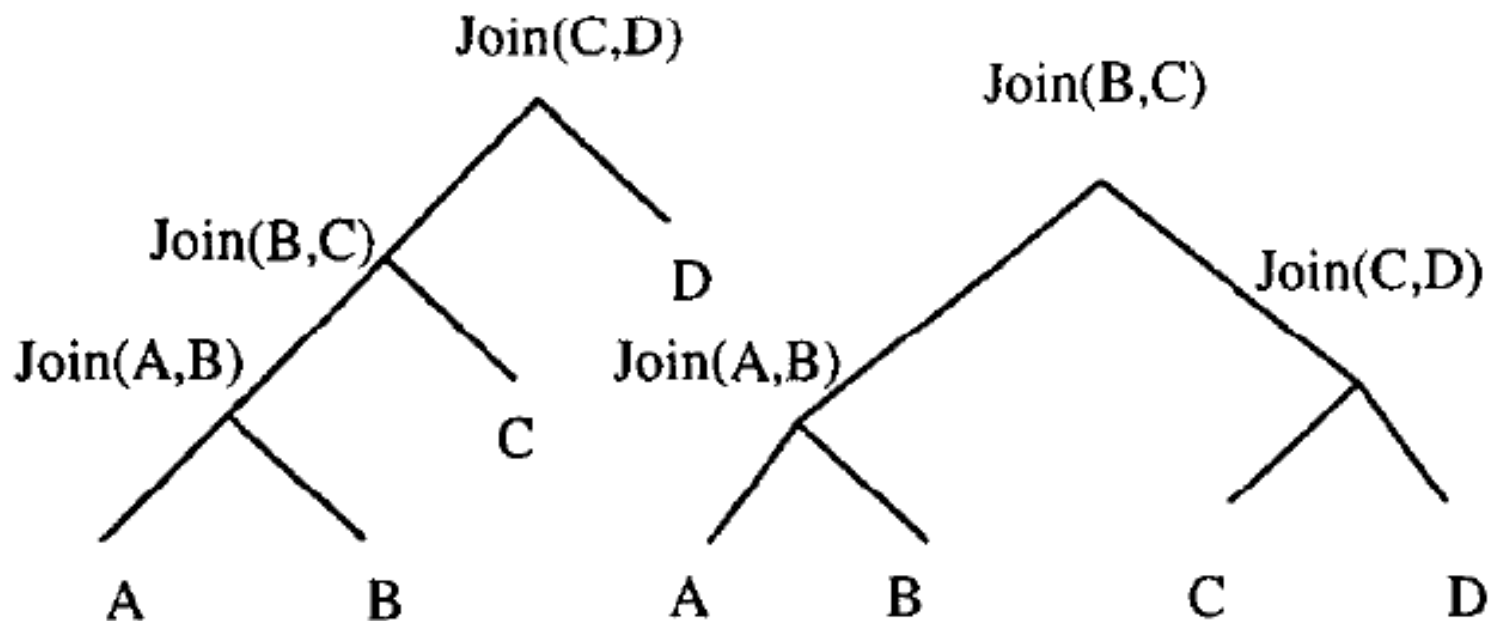


Query Optimization: Example (4)

- The first plan works better if each professor teaches many courses
- The second plan works better if each course contains many students
- Ways to vary plans
 - ◆ Algebraic transformations
 - ◆ Algorithm choice

Optimization Requirements

- Search space and plan enumerator
 - ◆ Linear
 - ◆ Bushy



Optimization Requirements

- Search space and plan enumerator
 - ◆ Linear
 - ◆ Bushy
- Cost estimator
 - ◆ Statistics (on relations, indices, columns)
 - ◆ Formulas to estimate statistics on derived relations
 - ◆ Formulas to estimate CPU and I/O costs

System-R Optimizer (1)

Consider a SPJ query consisting of n joins. The System-R optimizer made two important innovations:

- Dynamic programming
 - ◆ Need optimal solution to $k - 1$ joins to solve k joins
 - ◆ Start with 2 joins, work up to n

System-R Optimizer (1)

Consider a SPJ query consisting of n joins. The System-R optimizer made two important innovations:

- Interesting orders
 - ◆ Algorithms produce/need output in specific order, i.e. sorted on some value
 - ◆ Locally sub-optimal result may produce global optimum

System-R Optimizer (2)

- Considers $O(n2^{n-1})$ plans instead of naïve $O(n!)$ plans.
- Very influential, showed that relational databases are feasible
- Unfortunately, does not generalize beyond join ordering

More complex optimizations

Algebraic and calculus-level optimizations must be used carefully, since they may not actually help.

- Combine nested subqueries

```
SELECT DISTINCT prof FROM Teaches t
WHERE t.course IN (SELECT course FROM Takes)
```

More complex optimizations

Algebraic and calculus-level optimizations must be used carefully, since they may not actually help.

- Combine nested subqueries

```
SELECT DISTINCT prof FROM Teaches te, Takes ta
WHERE ta.course = te.course
```

More complex optimizations

Algebraic and calculus-level optimizations must be used carefully, since they may not actually help.

- Combine nested subqueries
- Push group-by through joins (subject to conditions)

```
SELECT prof, course, COUNT(student) AS classSz
FROM Teaches te, Takes ta
WHERE ta.course = te.course
GROUP BY prof, course
```

More complex optimizations

Algebraic and calculus-level optimizations must be used carefully, since they may not actually help.

- Combine nested subqueries
- Push group-by through joins (subject to conditions)

```
SELECT prof, classSz FROM Teaches te,  
(SELECT course, COUNT(student) AS classSz  
  FROM Takes GROUP BY course) taAgg  
WHERE te.course = taAgg.course
```

More complex optimizations

Algebraic and calculus-level optimizations must be used carefully, since they may not actually help.

- Combine nested subqueries
- Push group-by through joins (subject to conditions)

```
SELECT prof, COUNT(student) AS numStudents
FROM Teaches te, Takes ta
WHERE ta.course = te.course
GROUP BY prof
```

More complex optimizations

Algebraic and calculus-level optimizations must be used carefully, since they may not actually help.

- Combine nested subqueries
- Push group-by through joins (subject to conditions)

```
SELECT prof, SUM(classSz) AS numStudents
FROM Teaches te,
  (SELECT course, COUNT(student) AS classSz
   FROM Takes GROUP BY course) taAgg
WHERE te.course = taAgg.course
GROUP BY prof
```

More complex optimizations

Algebraic and calculus-level optimizations must be used carefully, since they may not actually help.

- Combine nested subqueries
- Push group-by through joins (subject to conditions)
- Multi-block optimization (similar to interprocedural analysis)

Statistical Summaries

- Histograms
 - ◆ Widely used
 - ◆ Fail to capture correlations between columns
- 2-dimension histograms
 - ◆ Can be very large
 - ◆ Many combinations of columns
- Used to estimate join and predicate selectivities
 - ◆ Hidden correlations lead to rapid loss of accuracy

Modern Optimizers: Starburst (1)

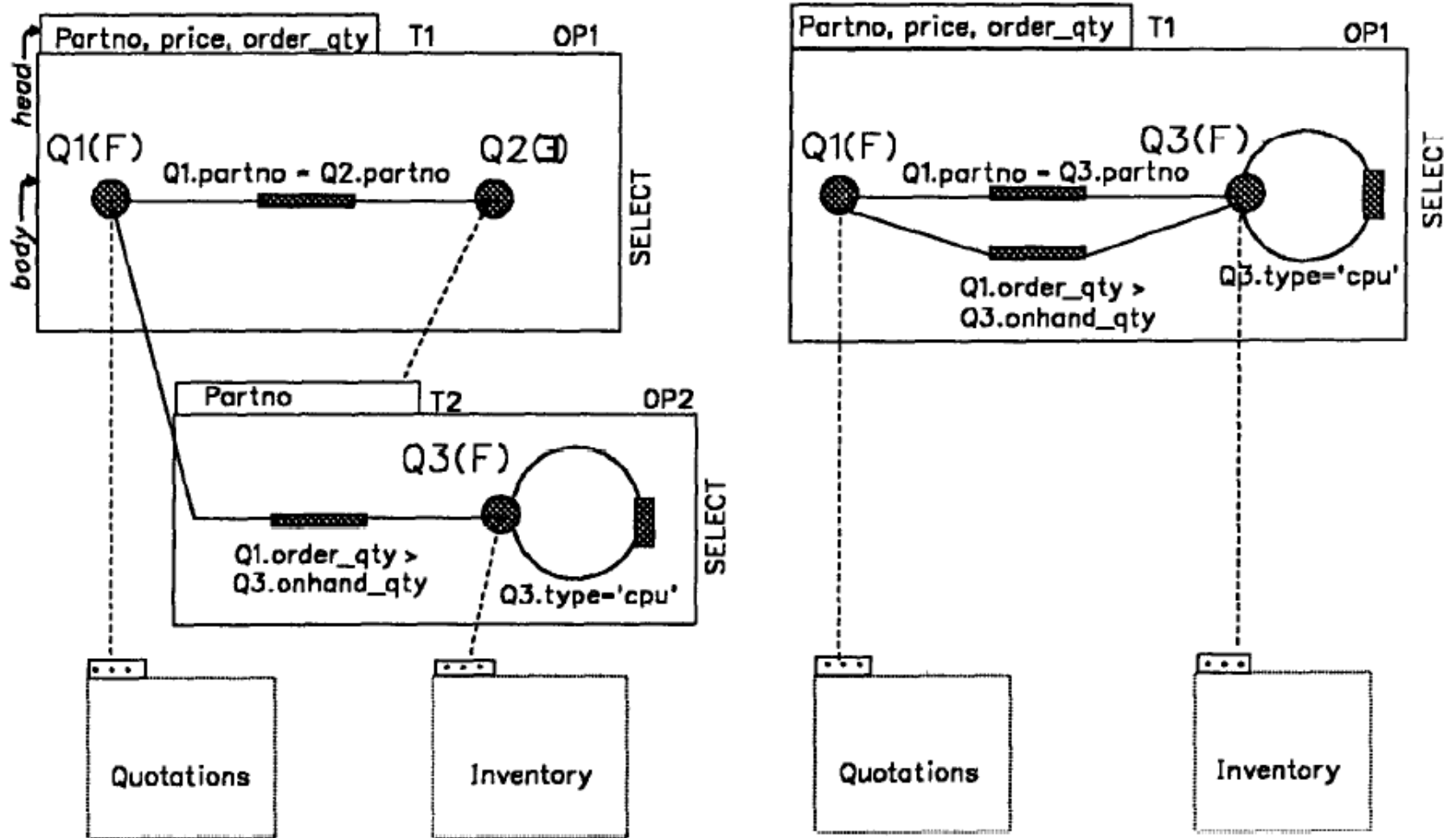
First phase (SQL/calculus-level transformations):

- SQL query is converted to QGM that captures relationship between blocks
- Rewrite rules are applied heuristically
 - ◆ Predicate migration
 - ◆ Projection push-down
 - ◆ Operation merging
 - ◆ More complex optimizations
- Alternative plans can be preserved until cost-based optimization

Modern Optimizers: Starburst (2)

```
SELECT partno, price, order-qty
FROM quotations Q1
WHERE Q1.partno IN
    (SELECT partno
     FROM inventory Q3
     WHERE Q3.onhand-qty < Q1.order-qty
     AND Q3.type = 'CPU' )
```

Modern Optimizers: Starburst (3)



Modern Optimizers: Starburst (4)

Second phase (algebraic transformations and physical operator selection):

- Each operator in the rewritten QGM is converted **LOW LEVEL Plan OPERATORS** (LOLEPOPs).
 - ◆ Bottom-up optimization like System-R
 - ◆ Expensive plans are pruned
 - ◆ Considers interesting orders

Starburst eventually became DB2.

Modern Optimizers: Volcano

- Top-down (recursion with memoization)
- All optimization is cost-based (even higher level transformations)
- Store best plan for each sub-expression and combination of physical properties
- Only one optimization phase
- Volcano eventually became the SQL Server optimizer.

Conclusions

- Need two things to change the (software) world
 - ◆ Carefully chosen model
 - ◆ Clever implementation
- Relational model finds 'sweet spot' between expressiveness and tractability
- Relational databases 'optimize for the common case' by efficiently implementing bulk operations