Database Tuning and Physical Design: Basics of Query Execution
Fall 2015

David Toman
School of Computer Science
University of Waterloo
Databases CS338

Basics of Query Execution

Goal

Develop a simple relational calculator that answers queries.

Considerations:

1. How is data physically represented?
2. How to compute answers to complex queries?
3. How are intermediate results managed?

Relational Algebra

Idea

Define a set of operations on the universe of finite relations... called a RELATIONAL ALGEBRA.

Constants:

- $R_i$: one for each relational scheme

Unary operators:

- $\sigma_{\varphi}$: selection (keeps only tuples satisfying $\varphi$)
- $\pi_V$: projection (keeps only attributes in $V$)

Binary operators:

- $\times$: Cartesian product
- $\cup$: union
- $-$: set difference

Relational Calculus/SQL to Algebra

How do we know that these operators are sufficient to evacuate all Relational Calculus queries?

Theorem (Codd)

For every domain independent Relational Calculus query there is an equivalent Relational Algebra expression.

$$\text{RCtoRA}(R_i(x_1, \ldots, x_k)) = R_i$$

$$\text{RCtoRA}(Q \land x_i = x_j) = \sigma_{#i = #j}(\text{RCtoRA}(Q))$$

$$\text{RCtoRA}(\exists x_i.Q) = \pi_{FV(Q) \setminus \{#i\}}(\text{RCtoRA}(Q))$$

$$\text{RCtoRA}(Q_1 \lor Q_2) = \text{RCtoRA}(Q_1) \cup \text{RCtoRA}(Q_2)$$

$$\text{RCtoRA}(Q_1 \land \neg Q_2) = \text{RCtoRA}(Q_1) - \text{RCtoRA}(Q_2)$$

...queries in $\land$ must have disjoint sets of free variables
...we must invent consistent way of referring to attributes
Iterator Model for RA

How do we avoid (mostly) storing *intermediate* results?

**Idea**

We use the *cursor OPEN/FETCH/CLOSE interface*.

Every *implementation* of a Relational Algebra operator:

1. implements the cursor interface to produce answers
2. uses the *same* interface to get answers from its children

...we make (at least) one *physical implementation* per operator.

---

**Physical Operators (example: selection)**

```c++
// select_{#i=#j}(Child)
OPERATOR child;
int i, j;

public:

OPERATOR selection(OPERATOR c, int i0, int j0)
  { child = c; i = i0; j = j0; }
void open() { child.open(); };
tuple fetch()
  { tuple t = child.fetch();
    if (t==NULL || t.attr(i)=t.attr(j))
      return t;
    return this.fetch();
  };
void close() { child.close(); }
```

---

**Physical Operators (cont.)**

The rest of the lot:

- **product:**
  - simple nested loops algorithm
- **projection:**
  - eliminate *unwanted attributes* from each tuple
- **union:**
  - simple concatenation
- **set difference:**
  - nested loops algorithm that checks for tuples on r.h.s.

**WARNING!**

This doesn’t quite work: projection and union may produce *duplicates*... need to be followed by a *duplicate elimination operator*.

---

**How to make it FAST(er)?**

**Observation**

Naive implementation for each operator will work

...very (very very very) slowly

**What to do?**

1. use (disk-based) data structures for efficient searching
2. use better algorithms to implement the operators
   commonly based on *SORTING* or *HASHING*
3. rewrite the RA expression to an equivalent, but more efficient one
   remove unnecessary operations (e.g., duplicate elimination)
   enable the use of better algorithms/data structures
Parallelism in Query Execution

Another approach to improving performance: take advantage of parallelism in hardware

- mass storage usually reads/writes data in blocks
- multiple mass storage units can be accessed in parallel
- relational operators amenable to parallel execution

Summary

Relational Algebra is the basis for efficient implementation of SQL

- provides a connection between conceptual and physical level
- breaks query execution to (easily) manageable pieces
- allows the use of efficient algorithms/data structures
- provides mechanism for query optimization based on logical transformations (including simplifications based on integrity constraints, etc.)

Performance of database operations depends on the way queries (and updates) are executed against a particular physical schema/design.

. . . understanding basics of query processing is necessary to making physical design decisions

. . . performance also depends on transaction management (later)