Query Optimization

Query optimization

- Given an SQL query, the query optimizer tries to figure out the order of operations that will make the query run the fastest.
- Possible because usually there is more than one way to run a query.

Why query optimization?

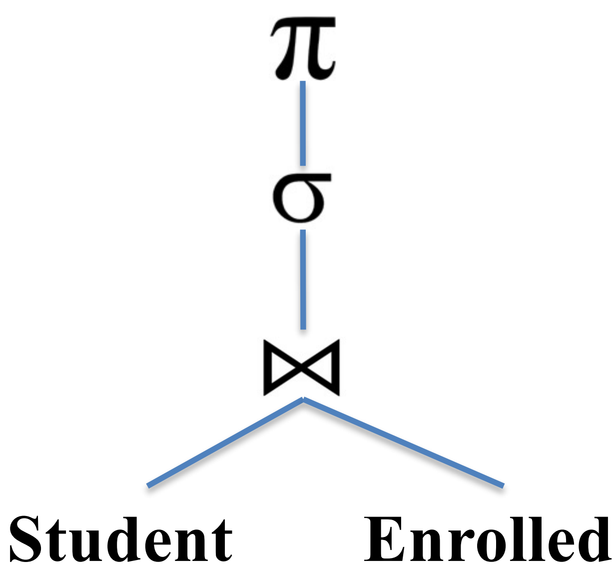
- SQL is a ***declarative*** language.
  - SQL only says ***what*** to retrieve from the DB, not the details of ***how***.
  - Unlike most programming languages (though there are other declarative languages).
- Good query optimization can make a big difference.

Example

- Students(R#, First, Last)
- Enrolled(R#, CRN)
- SELECT First, Last
  FROM Students NATURAL JOIN Enrolled
  WHERE CRN=12345
- $\pi_{First,Last}\ (\sigma_{\ CRN=12345}\ (S \bowtie E))$

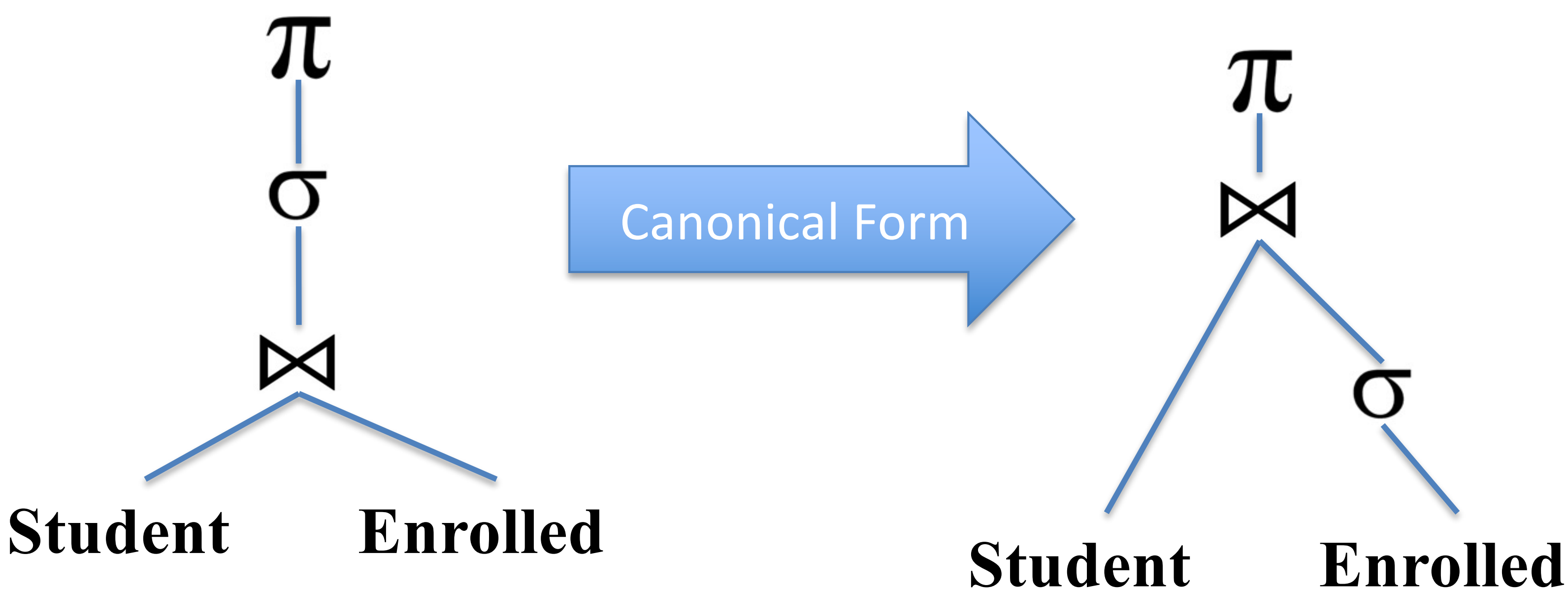
Example

- SELECT First, Last
  FROM Students NATURAL JOIN Enrolled
  WHERE CRN=12345



Example

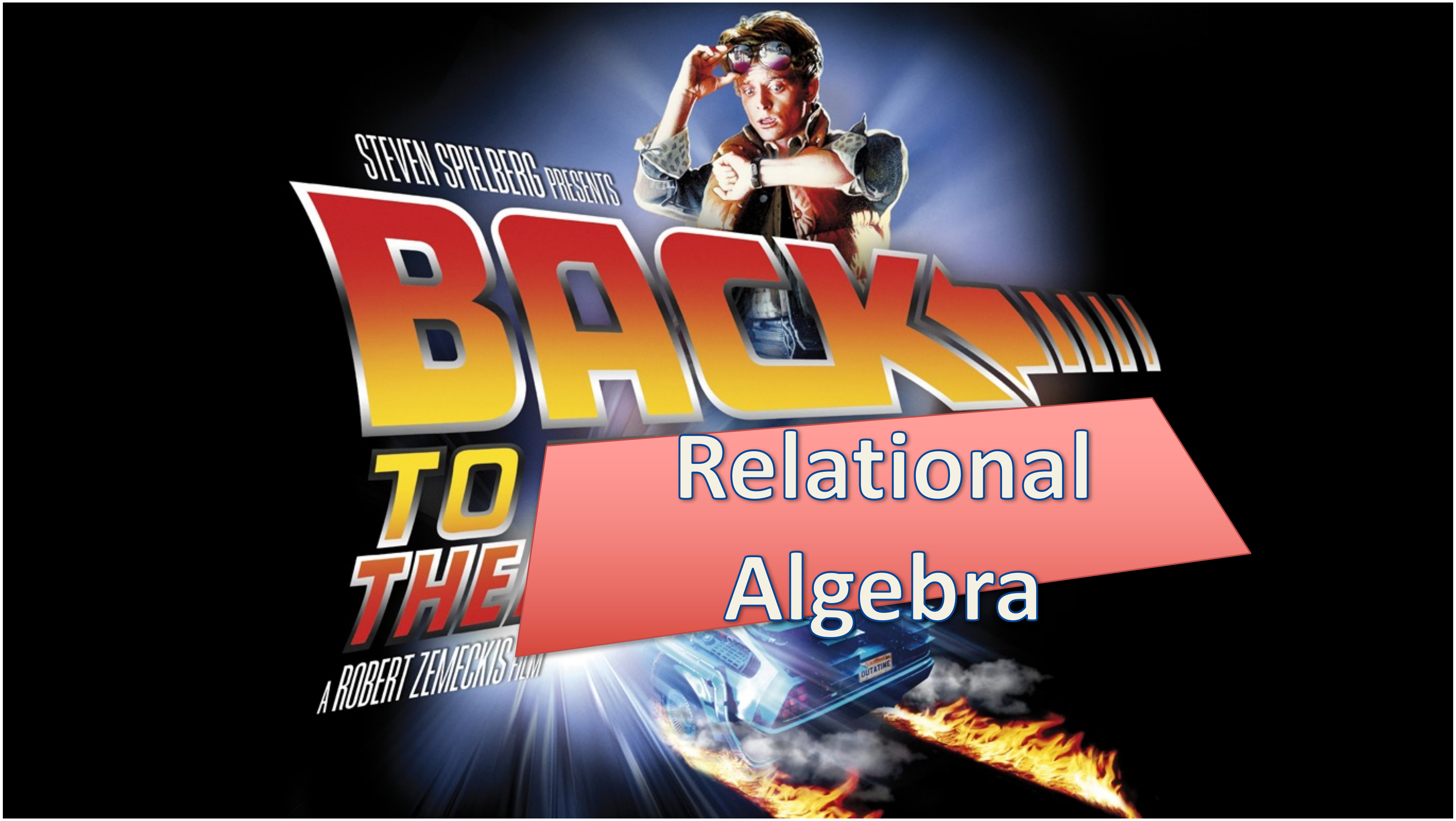
- SELECT First, Last
  FROM Students NATURAL JOIN Enrolled
  WHERE CRN=12345



Canonical Form

- Make all JOINs explicit with WHERE clauses.
  - "S NatJoin T" convert to: "S Join T WHERE…"
  - "S Join T ON" convert to: "S Join T WHERE…"
- Perform selections and projections as early as possible.


STEVEN SPIELBERG PRESENTS
BACK TO THE FUTURE
PG
A ROBERT ZEMECKIS FILM
OUTATIME



Relational Algebra

Relational algebra

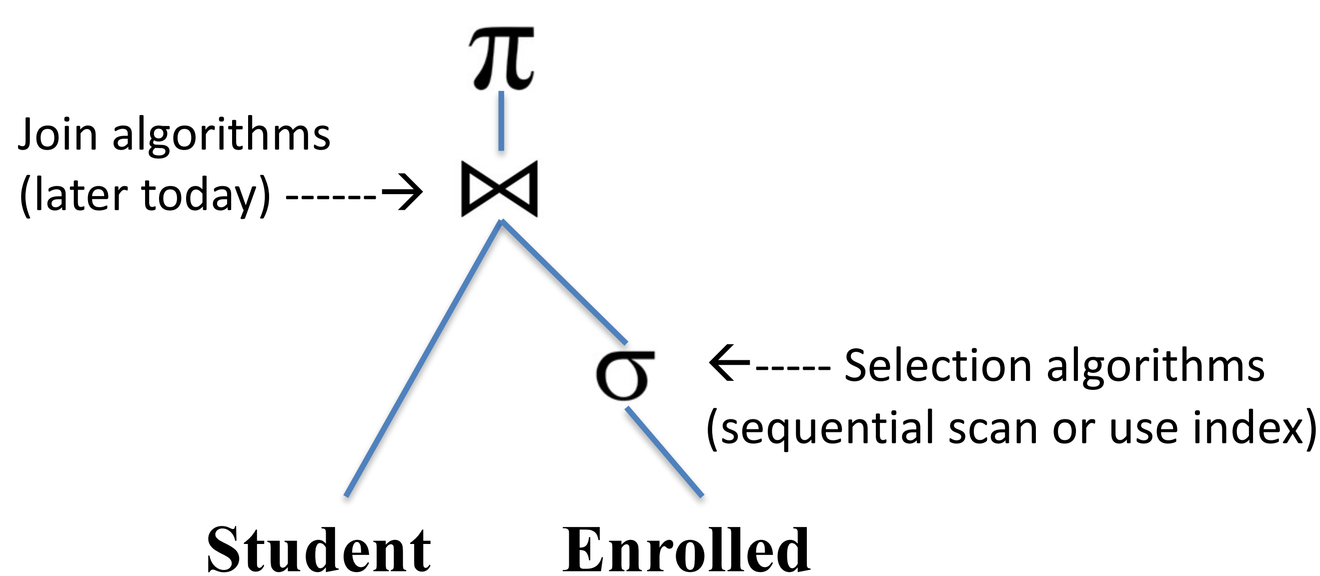
- How do we know
  $\pi_{First,Last}$ (σ $_{CRN=12345}$ ($S \bowtie E$))

is equal to
  $\pi_{First,Last}$ ($S \bowtie$ (σ $_{CRN=12345}$ (E))) ?

- Yay 172 proofs!

What are the algorithms used?

- SELECT First, Last
  FROM Students NATURAL JOIN Enrolled
  WHERE CRN=12345



Query optimization steps

- Parse query into internal form (e.g., parse tree)
- Convert to canonical form
- Generate a set of ***query plans*** (an ordering of steps and algorithms for answering the query)
- Estimate the cost of each query plan.
- Pick the best one.

PostgreSQL query plan demo

- EXPLAIN *<SQL statement here>*

Back to query optimization

- Projections and selections
  - Perform them early (but carefully) to reduce
    - number of tuples
    - size of tuples (remove attributes)
  - Project out (remove) all attributes except those requested or required (e.g., needed for joins)

How does a join work?

- Three main algorithms:
  - Nested loop join
  - Sort-merge join
  - Hash join

Nested loop join

```
For each tuple r in R do
  For each tuple s in S do
    If r and s satisfy the join condition
      Then output the tuple <r,s>
```

Sort-Merge join

- Assume we want to join R and S on some attribute A.
- Sort both R and S by A.
- Perform two simultaneous linear scans of R and S.
  - Works well assuming no duplicate values of A in both R and S (duplicates in one table are OK).

Hash join

- Assume we want to join R and S on some attribute A.
- Make a hash table of the smaller relation, mapping A to the appropriate row(s) of R (or S).
- Scan the larger relation to find the relevant rows using the hash table.
  - Only useful if smaller relation maps A to >1 rows of R (or S).

Equivalence of expressions

- Natural joins:
  - commutative $R \bowtie S = S \bowtie R$
  - associative $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

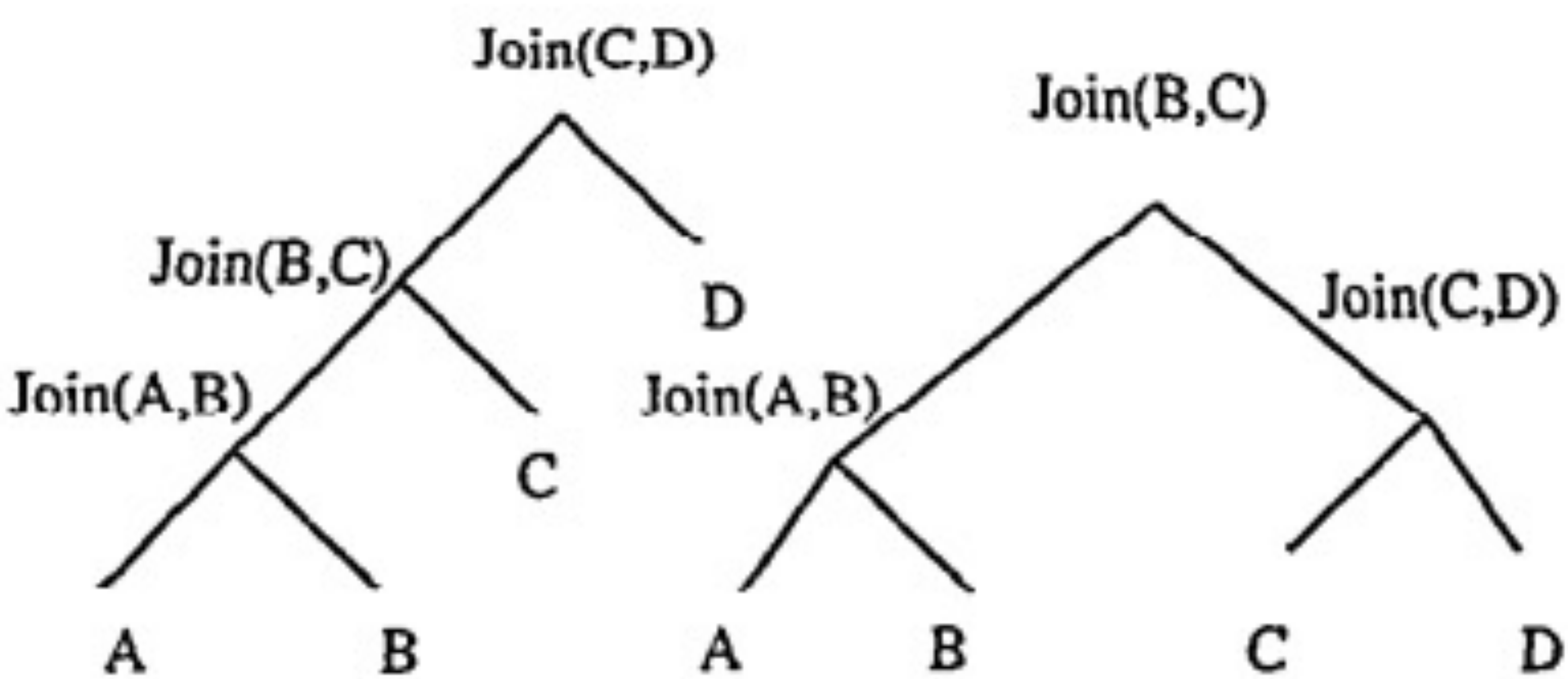
- How can we figure out how many possible orderings there are to join the tables?

Equivalence of expressions

- Natural joins:
  - commutative $$R \bowtie S = S \bowtie R$$
  - associative $$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$
- How can we figure out how many possible orderings there are to join the tables?
  - Each join is a binary tree.

Equivalence of expressions

- Natural joins:
  - commutative $$R \bowtie S = S \bowtie R$$
  - associative $$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$
- How can we figure out how many possible orderings there are to join the tables?
  - Each join is a binary tree.
  - # of binary trees with n nodes = O(4^n) = Catalan numbers. (This only considers associativity).



Why care?

Picking good join orders

- Query optimizer generates a few potential orders
  - Doesn't evaluate all O(4^n) possibilities.
  - Prefers deep trees over bushy trees. (Why?)
    - Bushy trees require lots of extra temporary tables to store intermediate results. A maximally-deep tree only requires one (or maybe two) temporary tables that we can keep overwriting.
    - How many left-deep trees are there for n relations? (left-deep means the tree is as deep as it can be in the left child).

- Query optimizer tries to estimate the cost for each ***query plan***, relying on
  - Statistics maintained for relations and indexes (size of relation, size of index, number of distinct values in columns, etc)
  - Formulas to estimate selectivity of predicates (the probability that a randomly-selected row will be true for a predicate)
  - Formulas to estimate CPU and I/O costs of selections, projections, joins, aggregations, etc.