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?

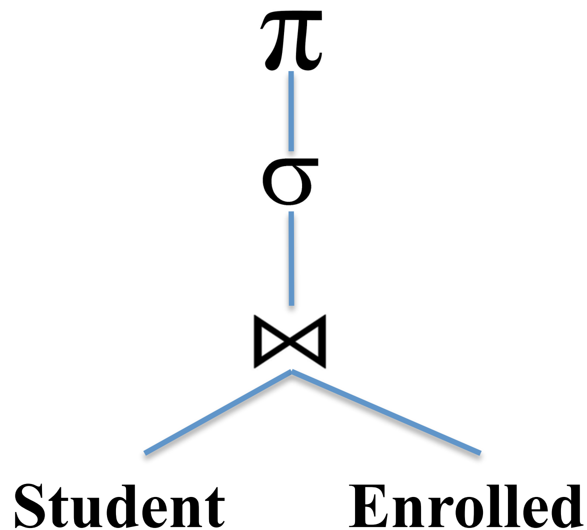
- SQL is declarative.
  - SQL only says ***what*** to retrieve from the DB, not the details of ***how***.
  - Unlike most programming languages (though there are other declarative languages).
- Good query optimization can make a big difference.

Example

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

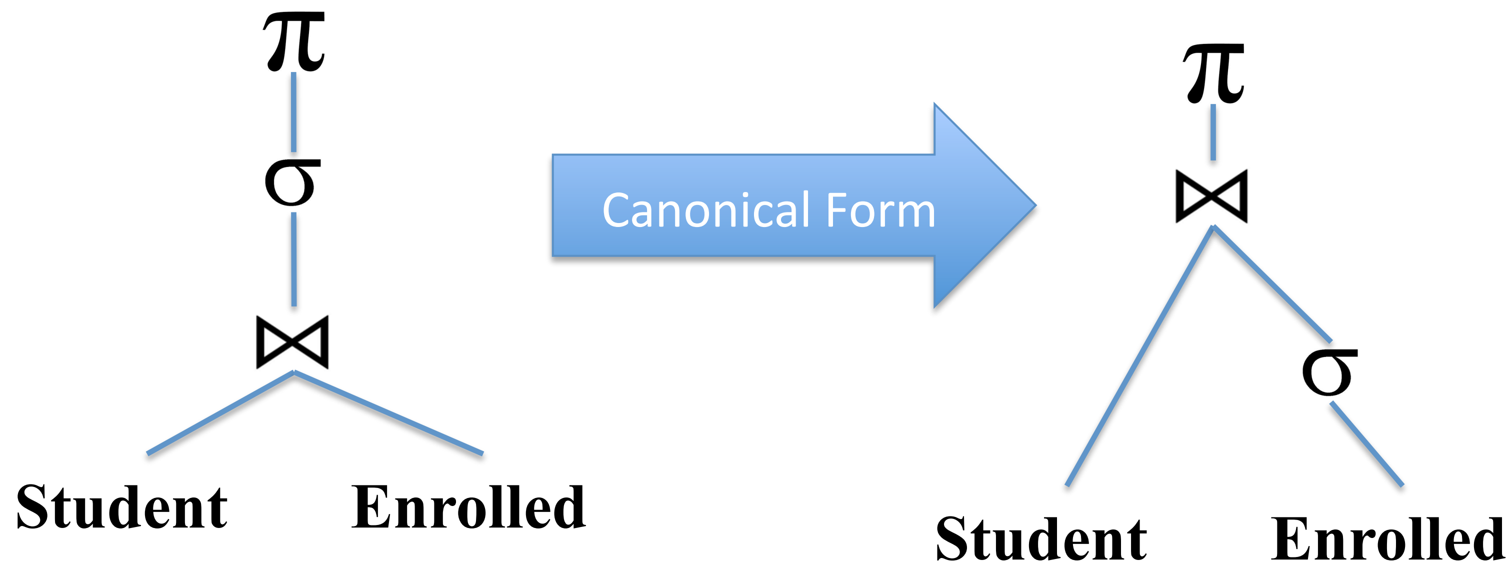
Example

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



Example

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

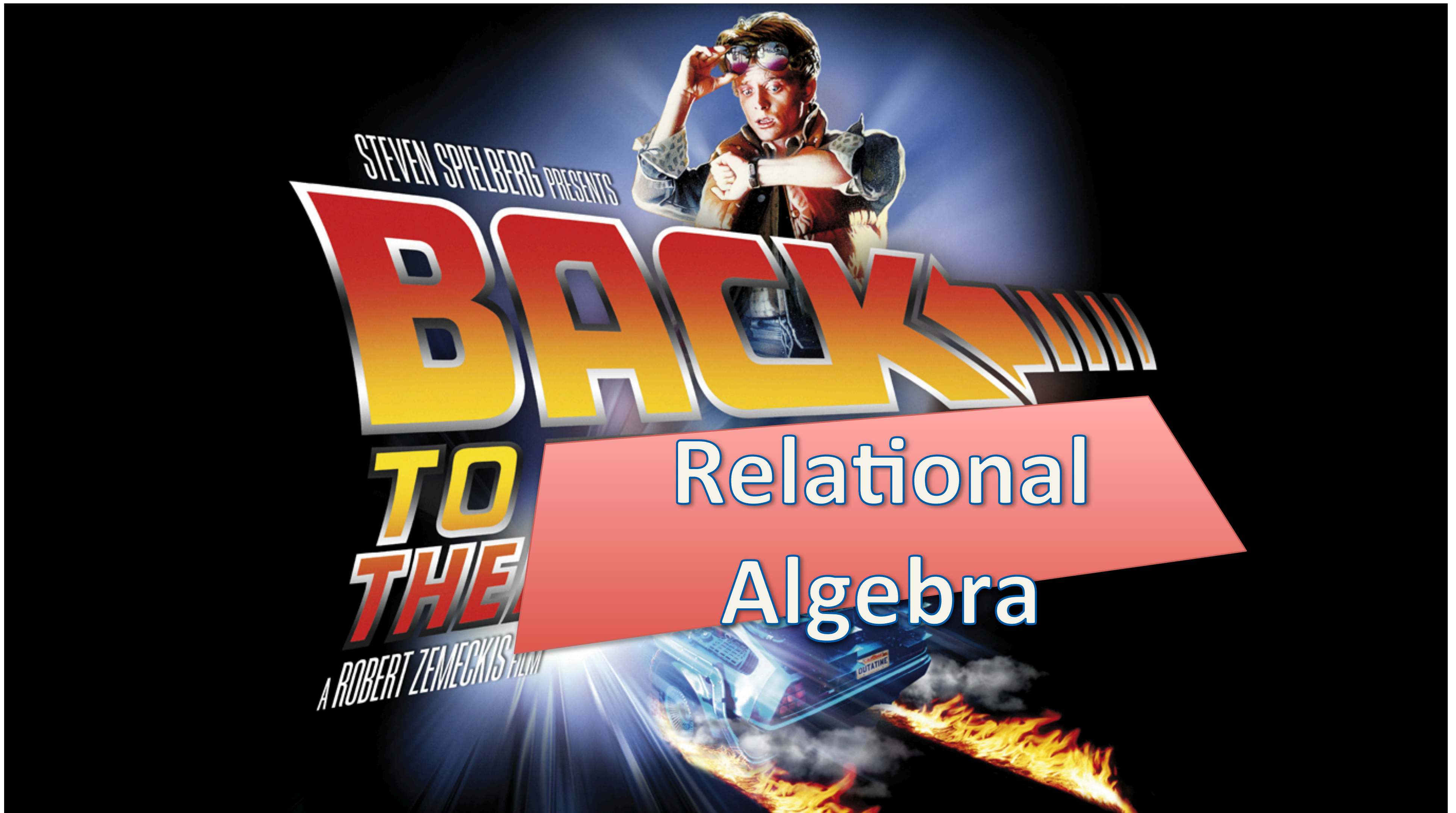


Canonical Form

- Make all JOINs explicit with WHERE clauses.
  - S NatJoin T == S Join T WHERE…
  - S Join T ON … == 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


STEVEN SPIELBERG PRESENTS
BACK
TO
THE
A ROBERT ZEMECKIS FILM
Relational
Algebra

Relational algebra

- How do we know
  $\pi_{F,L}$ ($\sigma_{CRN=12345}$ (S $\bowtie$ E))

is equal to

  $\pi_{F,L}$ (S $\bowtie$ $\sigma_{CRN=12345}$ (E)) ?

- Yay 172 proofs!

Example

- Prove

$$\sigma_P(R1 \cup R2) \stackrel{?}{=} \sigma_P(R1) \cup \sigma_P(R2)$$

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 linear scans of R and S.
  - Works well with no duplicate values of A.

Hash join

- Join R and S on 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.

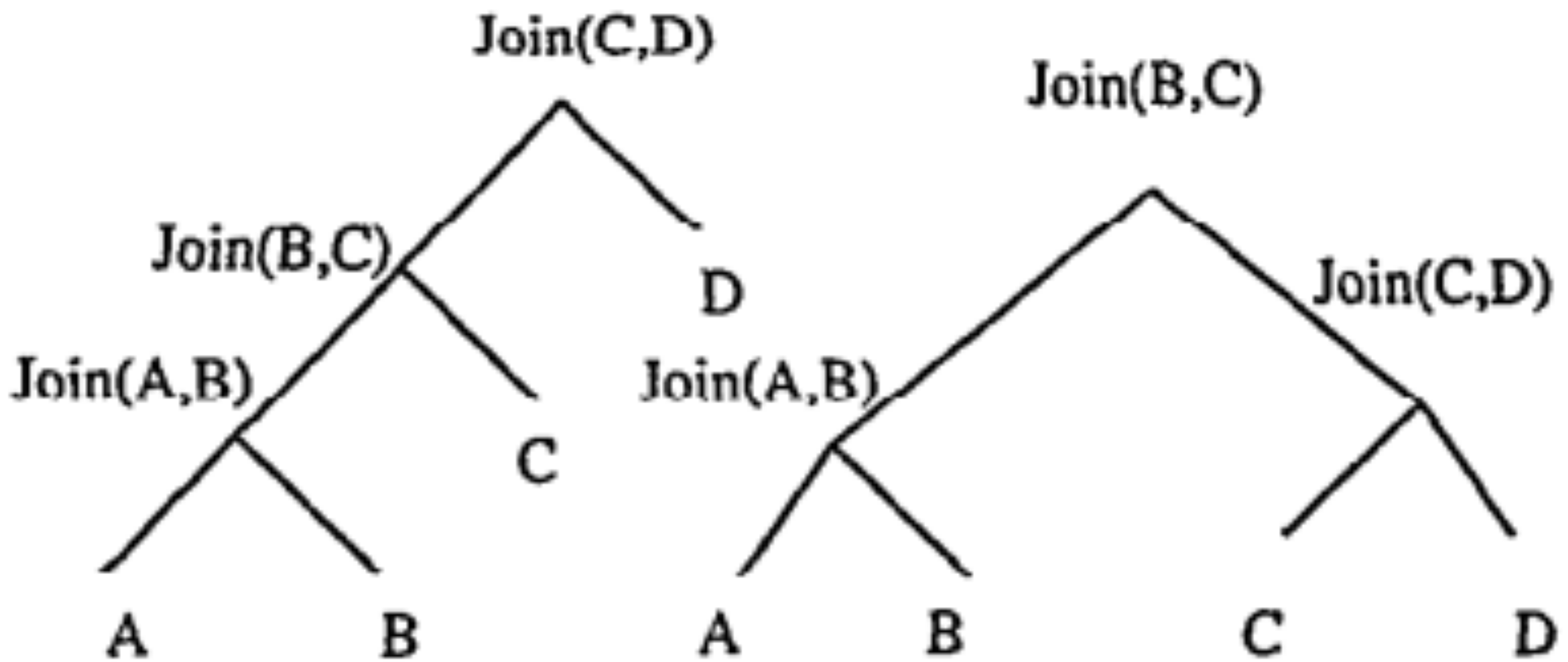
Equivalence of expressions

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

- How many different join orderings are there for n relations?

Equivalence of expressions

- Natural joins:
  - commutative $R \bowtie S = S \bowtie R$
  - associative $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- How many different join orderings are there for n relations?
  - Catalan number = O(4^n)



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.
    - How many left-deep trees are there for n relations?

- 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.

Views

- A ***view*** is a stored SQL query that can be used as if it were a relation.
- Only the query itself is stored, not the results.
  - Results are re-computed whenever the view is used.
- Saves typing, but not time.
- CREATE VIEW name AS SELECT…FROM…WHERE

Materialized Views

- A ***materialized view*** stores the results of the query rather than the query itself.
- Results are re-computed as needed.
- Saves typing and usually time, at the cost of space.
- CREATE MATERIALIZED VIEW name AS SELECT…FROM…WHERE
  - In many RDBMSs, but not SQLite.