Recap — Tons of Options

Physical Layout - Records in Page / Fields in Record

- **Delimited — Separator character splits fields (‘,’) /records (‘\n’)**
- **Fixed Width — Each field/record has a predictable / known size**
- **Directory — Each field/record has a fixed-size header/footer indicating where each field begins**

Indexing

- **Primary Hash — Put full records into a hash table (O(1) lookup, but only for == predicates)**
  - Static vs Dynamic
- **Primary Tree — Put full records into a tree-structure (O(log(N)) lookup, works for any ==, >, < predictate)**
  - B+Tree
  - LSM Tree
- **Secondary (Hash or Tree) — Index just record IDs in to avoid multiple copies of the entire record**

Sorting

- **In Memory**
- **External**

Group-By Aggregation

- **1-Pass Hash — Build a hash-table in memory to store each group and its current aggregate value**
- **Sort First — After sorting on group-by columns, all elements in a group adjacent (O(Nlog(N)) time)**
- **2-Pass Hash — Organize data into hash buckets, then do a 1-pass hash for each bucket**

Joins

- **Nested Loop Join — Foreach s in S : Foreach r in R : if test(s, r) : emit(s, r)**
- **Block-Nested Loop Join — Same, but add 2 more layers of loop, loading in blocks**
- **Index-Nested Loop Join — Replace inner loop with an index lookup based on the outer loop**
- **Sort/Merge Join — Sort both sides of the join first, then scan over the two lists in parallel**
- **2-Pass Hash Join — Group data from both sides into parallel buckets, then do an in-memory join on each bucket.**
- **1-Pass Hash Join — Build an in-memory hash table for one side, then use it for an index-nested loop join ewith the other.**
- **1-Pass Tree Join — Build an in-memory tree index for one side, then use it for an index-nested loop join with the other.**

Messy!

- **Assuming you make each choice exactly once, 864 options!**
  - Generally more!
- **Violating separation of concerns**
  - Programmers need to think about <u>what</u> they want to compute AND <u>how</u> to compute it, all at the same time
- **Can we fix it? Yes, but we need two things:**
  - We need a way to reason about “equivalent” options.
  - We need a way to evaluate which option is “best”.

Reasoning about Equivalent Options

Basic idea: Create a language (or “Algebra”) to describe computations

- **Common theme: Every expression in this language defines a table**
  - Like Math: 1 + 1 ≠ “Bob”… it’s a number instead
  - X,Y are tables, X (?) Y is also a table (if we decide on ‘(?)’ correctly)
- **What are the elements of this language (a “Relational Algebra”)?**
  - Need some sort of atomic, leaf value… just “a table” with an explicit value

- The basic operations we discussed at the start:
  - Filter (also called Select) — σ_c
  - Map (also called [Generalized] Projection) — π_A
  - Union — U
- The stuff we talked about in the last few classes seemed useful
  - Sort — τ
  - Aggregation (and Group-By Aggregation) — ɣ
  - Cross Products (and Joins) - x (and ⋈)
- Some other useful tools:
  - Convert Bags to Sets (Distinct) — δ
  - Take the first k records (Limit) — L
- **Let's try a few things:**
  - If R is a table, then so is σ(R)
    - … and so is $\pi(\sigma_c(R))$
    - … and so is $\pi(\sigma_c(R \times S))$
  - The "join" pattern $\sigma_c(R \times S)$ occurs often — and we have more efficient algorithms for it
    - … so we give it a shorthand: $R \bowtie_c S$
    - … Also a few other common shorthands:
      - $R \bowtie_{(R.ship\ =\ S.ship)} S \rightarrow R \bowtie_{ship} S$
      - $R \bowtie_{(R.ship\ =\ S.ship)} S \rightarrow R \bowtie S$ (if 'ship' is the only attribute name in common between R and S)
        - Also called a 'natural join': And of equality predicates on all columns with the same name
  - **Example**: Come up with 2-3 separate queries for the Last Names of all Captains of a Ship Located at Bajor.
    - $\pi_{Last\ Name}(\sigma_{Loc=\text{'Bajor'}}(Locations \bowtie_{ship} Captains))$
    - $\pi_{Last\ Name}((\sigma_{Loc=\text{'Bajor'}}(Locations)) \bowtie_{ship} Captains)$
    - $\pi_{Last\ Name}((\pi_{Last\ Name,Ship}(\sigma_{Loc=\text{'Bajor'}}(Locations))) \bowtie_{ship} Captains)$
    - These are all equivalent queries!

What is Equivalent?

- **Two expressions are equivalent if they're guaranteed to produce the same output**

Equivalent Expressions

They look the same, but one is good, one is evil

$\neq$

$=$

Two different expressions of the “same” character

RA Equivalencies

Selection

$$\sigma_{c_1 \wedge c_2}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(R)) \quad \text{(Decomposable)}$$

$$\sigma_{c_1 \vee c_2}(R) \equiv \delta(\sigma_{c_1}(R) \cup \sigma_{c_2}(R)) \quad \text{(Decomposable)}$$

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R)) \quad \text{(Commutative)}$$

Projection

$$\pi_a(R) \equiv \pi_a(\pi_{a \cup b}(R)) \quad \text{(Idempotent)}$$

Cross Product (and Join)

$$R \times (S \times T) \equiv (R \times S) \times T \quad \text{(Associative)}$$

$$(R \times S) \equiv (S \times R) \quad \text{(Commutative)}$$

Try It: Show that $R \times (S \times T) \equiv T \times (R \times S)$

Selection and Projection

$$\pi_a(\sigma_c(R)) \equiv \sigma_c(\pi_a(R))$$

Selection commutes with Projection
(but only if attribute set **a** and condition **c** are *compatible*)

a must include all columns referenced by **c**

Show that

$$\pi_a(\sigma_c(R)) \equiv \pi_a(\sigma_c(\pi_{a \cup \mathtt{cols}(c)}(R)))$$

When is this rewrite a good idea?

Join

$$\sigma_c(R \times S) \equiv R \bowtie_c S$$

Selection combines with Cross Product
to form a Join as per the definition of Join
(Note: This only helps if we have a join algorithm for conditions like **c**)

Show that

$$\sigma_{(R.B=S.B)\wedge(R.A>3)}(R \times S) \equiv \sigma_{(R.A>3)}(R \bowtie_{(R.B=S.B)} S)$$

When is this rewrite a good idea?

Selection and Cross Product

$$\sigma_c(R \times S) \equiv (\sigma_c(R) \times S)$$

Selection commutes with Cross Product
(but only if condition **c** references attributes of R exclusively)

Show that

$$\sigma_{(R.B=S.B)\wedge(R.A>3)}(R \times S) \equiv \sigma_{(R.A>3)}(R) \bowtie_{(R.B=S.B)} S$$

When is this rewrite a good idea?

Projection and Cross Product

$$\pi_a(R \times S) \equiv (\pi_{a_1}(R)) \times (\pi_{a_2}(S))$$

Projection <u>commutes</u> (distributes) over Cross Product
(where **a_1** and **a_2** are the attributes in **a** from R and S respectively)

<u>Show that</u>

$$\pi_a(R \bowtie_c S) \equiv (\pi_{a_1}(R)) \bowtie_c (\pi_{a_2}(S))$$

(under what condition)

How can we work around this limitation?

$$\pi_a((\pi_{a_1 \cup (\mathtt{cols}(c) \cap \mathtt{cols}(R)}(R)) \bowtie_c (\pi_{a_2 \cup (\mathtt{cols}(c) \cap \mathtt{cols}(S)}(S)))$$

When is this rewrite a good idea?

RA Equivalencies

Union and Intersections are <u>Commutative</u> and <u>Associative</u>

Selection and Projection both commute with both Union and Intersection

When is this rewrite a good idea?

Example

```
SELECT R.A, T.E
  FROM R, S, T
 WHERE R.B = S.B
   AND S.C < 5
   AND S.D = T.D
```