CS660: Grad Intro to Database Systems

Final Exam Review

Instructor: Manos Athanassoulis

https://bu-disc.github.io/CS660/

Course Evaluation

12:30-12:45 course evaluation

https://tinyurl.com/CS660-F23-CourseEval

if the above does not work:

https://go.blueja.io/inAWTDZkT0CDuuMCUTba5g

What to study for Final

From the Book (focus on the 2nd half of the semester)

```
Chapter  4: 4.1–4.2,   Relational Algebra
Chapter 12: 12.1–12.6, Overview of Query Evaluation
Chapter 14: 14.1–14.7, Evaluating Relational Operators
Chapter 15: 15.1–15.5, A Typical Relational Optimizer
Chapter 16: 16.1–16.7, Transaction management
Chapter 17: 17.1–17.6, Concurrency control
Chapter 18: 18.1–18.6, Crash recovery
```

- The 1st half of the semester is assumed knowledge
- Lecture Slides from Oct 24, 2023 until December 7, 2022
  - Including in-class guest lectures from 11/30 and 12/5
- Homeworks

Exam Date & Time

Wednesday, December 20, 2023 at noon

12:00pm until 2:00 pm in CAS 313

Relational Algebra: 5 Basic Operations

Selection (σ) Selects a subset of *rows* from relation (horizontal).

Projection (π) Retains only wanted *columns* from relation (vertical).

Cross-product ($\times$) Allows us to combine two relations.

Set-difference ($-$) Tuples in R_1, but not in R_2.

Union ($\cup$) Tuples in R_1 and/or in R_2.

each operation returns a relation : ***composability*** (Algebra is “closed”)

Compound Operator: Join

Joins are compound operators : $\times$, σ, (sometimes) π

frequent type is "*natural join*" (often called "join")

R ⋈ S conceptually is:

compute R×S

select rows where attributes in both **R**, **S** have equal values

project all unique attributes and one copy of the common ones

Note: Usually done much more efficiently than this

Useful for putting ***normalized*** relations back together

Reserves (sid, bid, day) **Sailors** (sid, sname, rating, age)

Boats (bid, bname, color)

Find names of sailors who have reserved a red boat

boat color only available in Boats; need an <u>extra</u> join:

$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie \text{Reserves} \bowtie Sailors)$$

a <u>*more efficient*</u> solution:

why more efficient?

$$\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'} Boats) \bowtie \text{Res}) \bowtie Sailors)$$

a <u>query optimizer</u> can find this given the first solution!

Reserves (sid, bid, day) **Sailors** (sid, sname, rating, age)

Boats (bid, bname, color)

Find all pairs of sailors with the same rating

$$\rho(S_1(1 \longrightarrow sid_1, 2 \longrightarrow sname_1, 3 \longrightarrow rating_1, 4 \longrightarrow age_1), Sailors)$$

$$\rho(S_2(1 \longrightarrow sid_2, 2 \longrightarrow sname_2, 3 \longrightarrow rating_2, 4 \longrightarrow age_2), Sailors)$$

$$\pi_{sname_1, sname_2}(S_1 \bowtie_{rating_1 = rating_2 \wedge sid_1 \neq sid_2} S_2)$$

is this ok?

$sid_1 < sid_2$

Reserves (sid, bid, day) **Sailors** (sid, sname, rating, age)
Boats (bid, bname, color)

Find the names of sailors who have reserved all boats

use division; schemas of the input relations to / must be carefully chosen (**why?**)

$$\rho\,(Tempsids, (\pi_{sid,bid}\text{Reserves}) / (\pi_{bid} Boats))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

To find sailors who have reserved all ”Interlake” boats:

$$\text{.....} \;\; / \; \pi_{bid}(\sigma_{bname='Interlake'} Boats)$$

Query Processing Overview

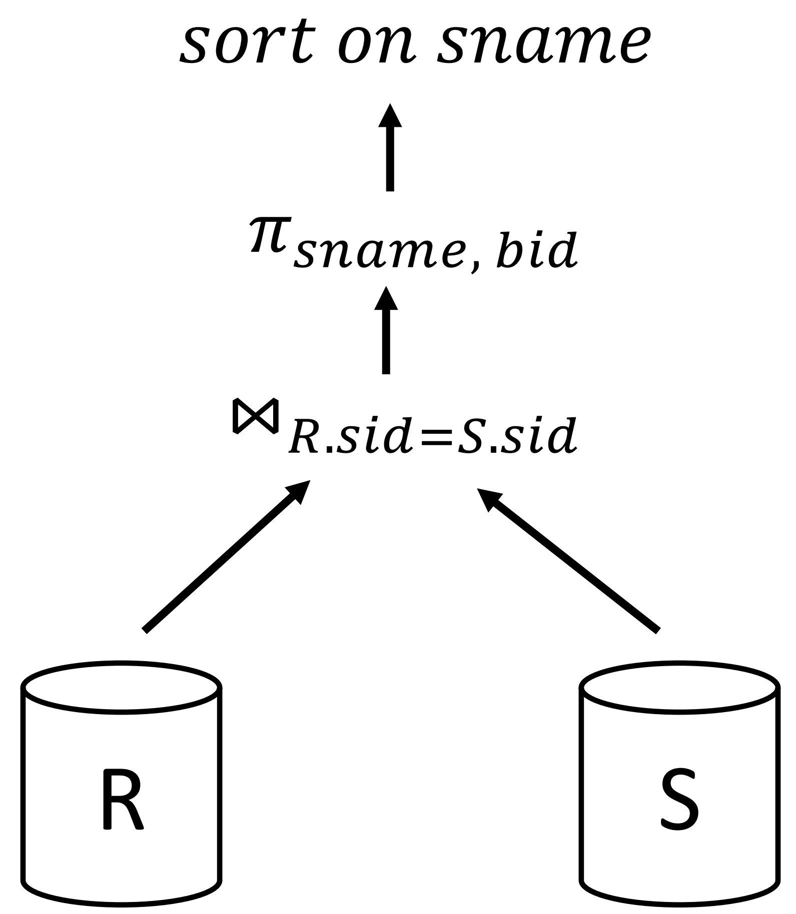
```
SELECT    sname, bid
FROM      R, S
WHERE     R.sid=S.sid
ORDER BY  sname
```

- The *query parser and optimizer* translates SQL to a special internal “language”
  - Query Plans
- The *query executor* is an *interpreter* for query plans
- Think of query plans as “box-and-arrow” *dataflow* diagrams
  - Each **box** implements a ***relational operator***
  - **Edges** represent a **flow of tuples** (columns as specified)
  - For **single-table queries**, these diagrams are **straight-line graphs**

How to evaluate query operators?

- Two general ideas: **sorting** and **hashing**

- Used for Group by, aggregates, joins, distinct
- For selection: Linear scan or Index based
  - When using Index:
    - Important if it is clustered or unclustered



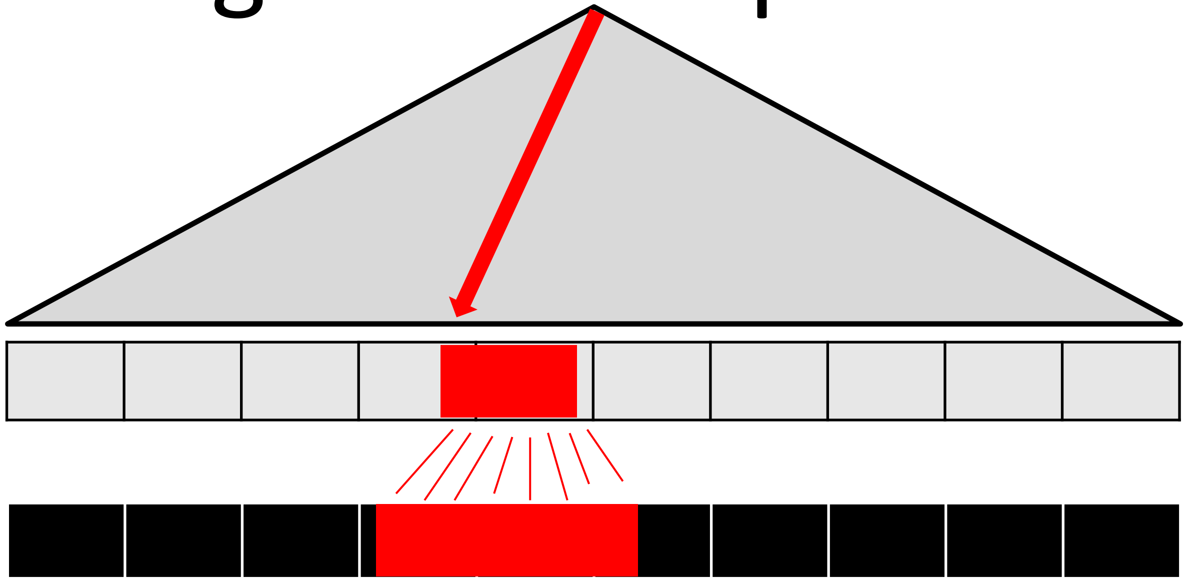
Selections using Index– Explained

R: M=1000, p_R=100, ts=40b

A) clustered

data entries:

data records:



index search: $\log_B M$

$\lceil f \cdot M \rceil =$

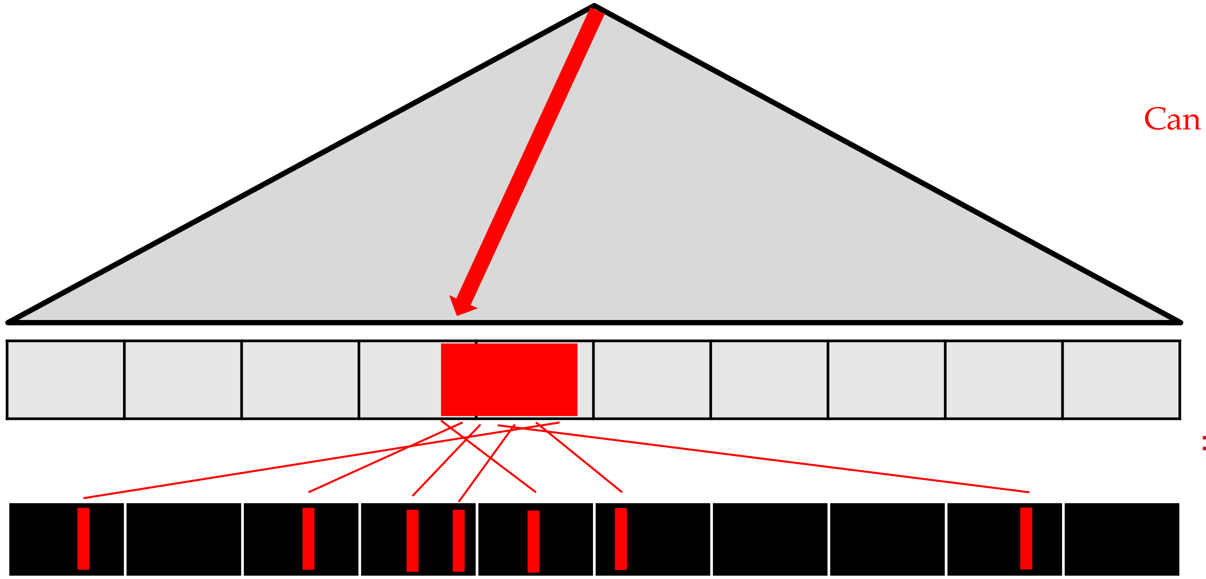
$= 10\% \cdot 1000 = 100$

Can we do better?

B) unclustered

data entries:

data records:



index search: $\log_B M$

$\lceil f \cdot M \cdot p_R \rceil =$

$= 10\% \cdot 1000 \cdot 100 = 10000$

Query Evaluation: Join

- A number of different approaches to evaluate join:
  - Page Oriented Nested Loop Join
  - Indexed Nested Loop Join
  - Block Nested Loop Join
  - Sort-Merge Join
  - Hash Join

- Formulas to estimate the cost of operators!
  - Important for query optimization

Costs of Join R ⋈ S, R has M pages, S has N pages, buffer B

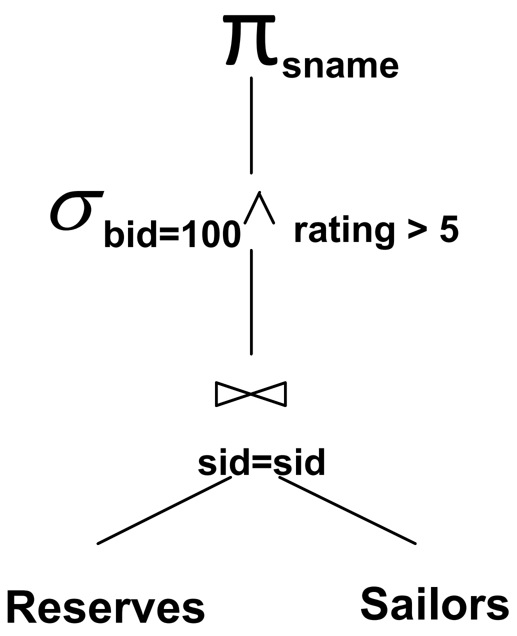
- PNLJ: M+M*N
- BNLJ: $M + \lceil \frac{M}{B-2} \rceil * N$
- Indexed NL: M + M*p_R* cost of index for match
- Sort-Merge: (best case) Sort R + Sort S + M+N
  - If B > $\sqrt{M}$, if M is larger than N (R larger relation) then 3*(M+N)
- Hash-join: partition until every partition is smaller than B-1.
  - if B > $\sqrt{N}$, if N is smaller than M (S smaller relation) then 3*(M+N)
  - Otherwise, re-partition until each partition fits in memory
    - Each partition or repartition divides the previous partitions in B-1 equal new partitions

Recall: Query Optimization Overview

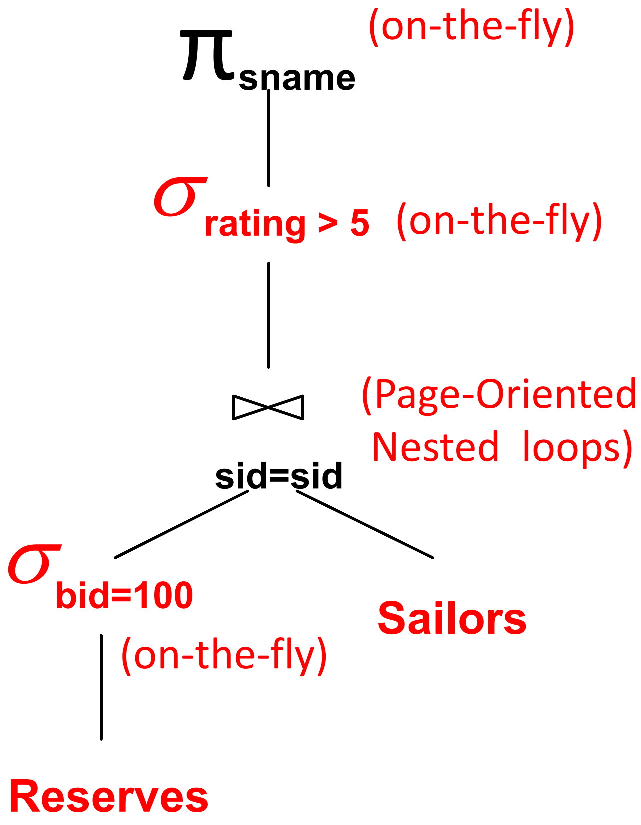
1. Query first broken into "blocks"
2. Each block converted to relational algebra
3. Then, for each block, several alternative query plans are considered
4. Plan with lowest estimated cost is selected (ops can be pushed)

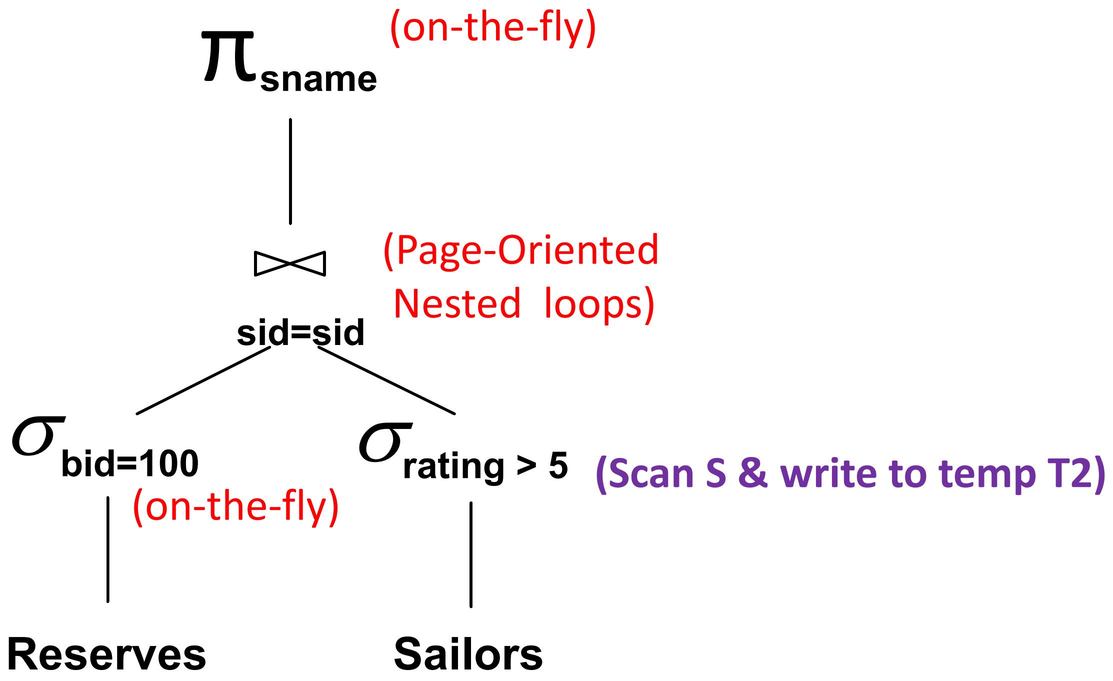
```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```

$$\pi_{(sname)}\sigma_{(bid=100 \wedge rating > 5)} (\text{Reserves} \bowtie \text{Sailors})$$



Example of a plan:





Query Optimization

Query Plan: *Tree of R.A. ops (and others) with choice of algo.*

- `pull' interface: when we `pull' for next tuple, op `pulls' on its inputs

Two Main Issues

1. For a given query, what plans are considered?

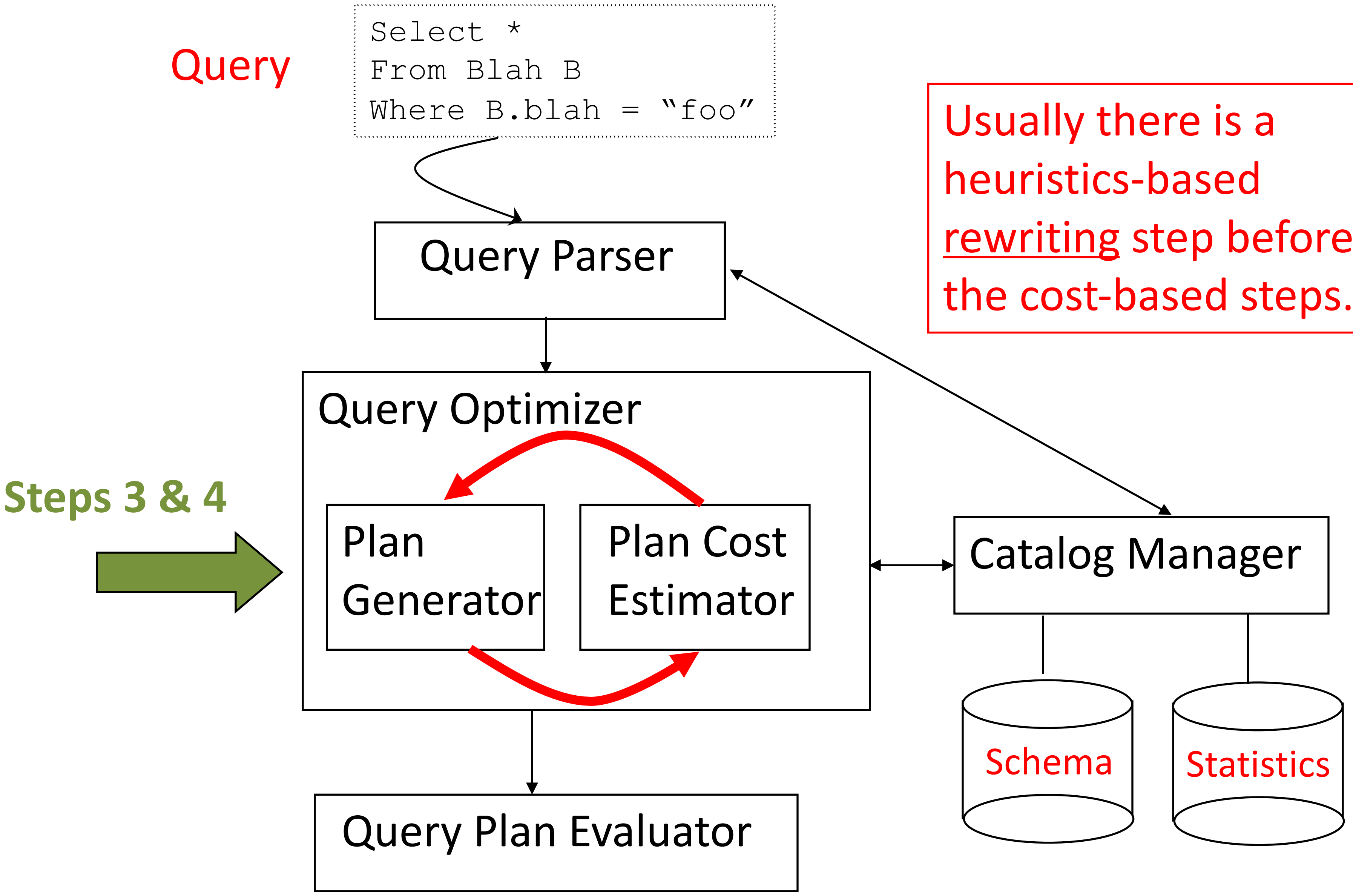
   Algorithm to search plan space for cheapest (estimated) plan.

2. How is the cost of a plan estimated?

Ideally: Want to find best plan.

Reality: Avoid worst plans!

Cost-based Query Sub-System



Usually there is a heuristics-based rewriting step before the cost-based steps.

Highlights of System R Optimizer

Impact:

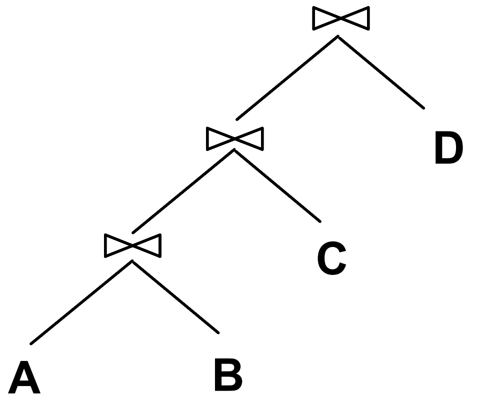
- Most widely used currently; works well for < 10 joins

Cost estimation:

- Very inexact, but works okay in practice
- Statistics, maintained in system catalogs, used to estimate cost of operations and result sizes
- Considers combination of CPU and I/O costs
- More sophisticated techniques known now

Plan Space: Too large, must be pruned

- Only the space of *left-deep plans* is considered
- Cross products are avoided



System R Strategy

Shared sub-plan observation suggests a better strategy:

Enumerate plans using N passes (N = # relations joined):

- Pass 1: Find best 1-relation plans for each relation
- Pass 2: Find best ways to join result of each 1-relation plan as outer to another relation *(All 2-relation plans.)*
- Pass N: Find best ways to join result of a (N-1)-relation plan as outer to the Nth relation *(All N-relation plans.)*

For each subset of relations, retain only:

- Cheapest subplan overall (possibly unordered), plus
- Cheapest subplan for each *interesting order* of the tuples

For each subplan retained, remember cost and result size estimates

A Note on ”Interesting Orders”

An intermediate result has an “interesting order” if it is sorted by any of:

- ORDER BY attributes
- GROUP BY attributes
- Join attributes of other joins

Transactions and Concurrency control

an atomic sequence of database actions (reads/writes)

takes DB from one consistent state to another

transaction - DBMS's abstract view of a user program:

- a sequence of reads and writes.

Correctness: The **ACID** properties

A tomicity: All actions in the transaction happen, or none happen

C onsistency: If each transaction is consistent, and the DB starts consistent, it ends up consistent

I solation: Execution of one transaction is isolated from that of other transactions

D urability: If a transaction commits, its effects persist

Concurrency Control

- We first attack Isolation, then address the rest
- Schedule, equivalent schedule, serializable schedule

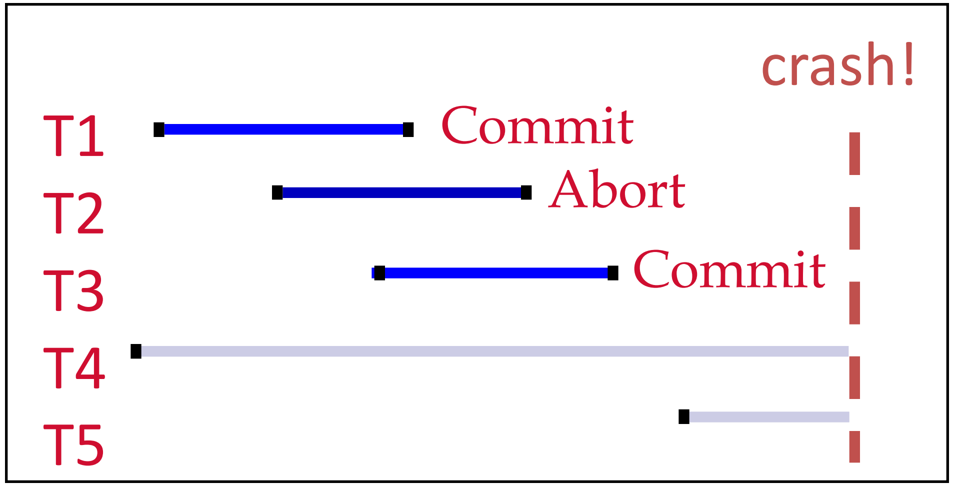
- We can use locking to guarantee conflict serializable schedule
  - Conflict equivalent to a serial schedule
  - We can check if a schedule is c.s.

- 2PL and Strict 2PL
- Optimistic CC
  - Kung-Robinson Model (Read, Validate, Write phases)
  - Timestamp based
  - MVCC

Crash recovery - Motivation

- Atomicity:
  - Transactions may abort ("Rollback").
- Durability:
  - What if DBMS stops running? (Causes?)

- Desired state after system restarts:
  - T1 & T3 should be durable.
  - T2, T4 & T5 should be aborted (effects not seen).



Buffer Management summary

	No Steal	Steal
No Force		**Fastest**
Force	**Slowest**	

Performance Implications

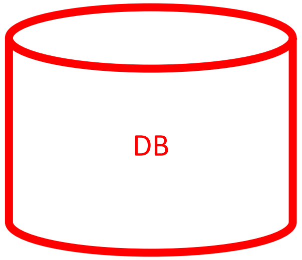
	No Steal	Steal
No Force	**No UNDO REDO**	**UNDO REDO**
Force	**No UNDO No REDO**	**UNDO No REDO**

Logging/Recovery Implications

Crash Recovery: What’s Stored Where

LogRecords

- prevLSN
- XID
- type
  - update
  - commit
  - abort
  - checkpoint
  - CLR
  - end
- pageID
- length
- offset
- before-image
- after-image

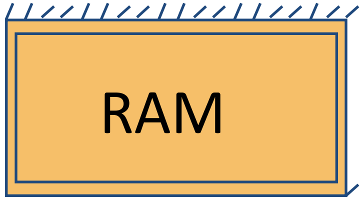


Data pages

each with a pageLSN

master record

LSN of most recent checkpoint



Xact Table

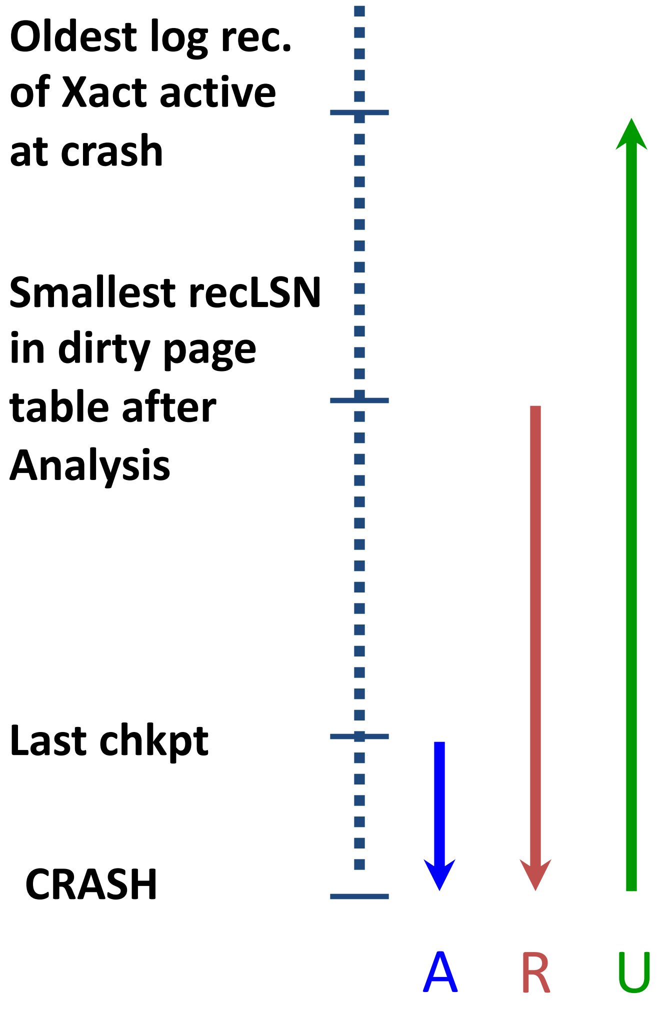
- lastLSN
- status

Dirty Page Table

- recLSN

flushedLSN

Crash Recovery: Big Picture



- Start from a checkpoint (found via master record).
- Three phases. Need to do:
  - Analysis - Figure out which transactions committed since checkpoint, which failed.
  - REDO ***all*** actions.

    (repeat history)
  - UNDO effects of failed transactions.

"Repeats History" in order to simplify the logic of recovery.

Must handle arbitrary failures

Even during recovery!