Select R.text from Report R, Weather W where W.image.rain() and W.city = R.city and W.date = R.date and R.text.matches("insurance claims")
Recall the Netflix Schema

### Ratings

<table>
<thead>
<tr>
<th>RatingID</th>
<th>Stars</th>
<th>RateDate</th>
<th>UID</th>
<th>MID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Users

<table>
<thead>
<tr>
<th>UID</th>
<th>Name</th>
<th>Age</th>
<th>JoinDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>Alice</td>
<td>23</td>
<td>01/10/13</td>
</tr>
<tr>
<td>80</td>
<td>Bob</td>
<td>41</td>
<td>05/10/13</td>
</tr>
</tbody>
</table>

### Movies

<table>
<thead>
<tr>
<th>MID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
</tbody>
</table>
### Example SQL Query

```sql
SELECT M.Year, COUNT(*) AS NumBest
FROM Ratings R, Movies M
WHERE R.MID = M.MID
AND R.Stars = 5
GROUP BY M.Year
ORDER BY NumBest DESC
```

Suppose, we also have a B+Tree Index on Ratings (Stars)
SELECT R.stars = 5
FROM Ratings Table

JOIN R.MID = M.MID
ON NumBest

GROUP BY M.Year, COUNT(*)

SORT

CALL AGGREGATE

SELECT No predicate
FROM Movies Table

Called “Logical” Operators

From extended RA

Each one has alternate “physical” implementations
Physical Query Plan

Indexed Access
Use Index on Stars

File Scan
Read heapfile

Result Table

External Merge-Sort
In-mem quicksort; B=50

Sort-based Aggregate

Index-Nested Loop Join

Indexed Access

Sort-based Aggregate

Movies Table

Called “Physical” Operators

Specifies exact algorithm/code to run for each logical operator, with all parameters (if any)

Aka “Query Evaluation Plan”
This is also a correct PQP for the given LQP!

**Q:** Which PQP is faster?

This is a key job of the RDBMS Query Optimizer!
So, what is query optimization and how does it work?
Meet Query Optimization

Basic Idea: A given LQP could have several possible PQPs with very different runtime performance

Goal (Ideal): Get the optimal (fastest) PQP for a given LQP

Goal (Realistic): Fine, just avoid the "clearly awful" PQPs!

Jeff Naughton

GOOD LUCK WITH THAT

Query optimization is a metaphor for life itself! It is often hard to even know what an optimal plan would be, but it is feasible to avoid many obviously bad plans!
Query Optimization

- Overview of Query Optimizer
- Physical Query Plan (PQP)
  - Concept: Pipelining
  - Mechanism: Iterator Interface
- Enumerating Alternative PQPs
  - Logical: Algebraic Rewrites
- Costing PQPs
Overview of Query Optimizer

SQL Query

Parser

Logical Query Plan

Plan Enumerator

Optimizer

Plan Cost Estimator

Catalog

Physical Query Plan (Optimized)

To Scheduler/Executor
System Catalog

❖ Set of pre-defined relations for metadata about DB (schema)
❖ For each **Relation**:
  Relation name, File name
  File structure (heap file vs. clustered B+ tree, etc.)
  Attribute names and types; Integrity constraints; Indexes
❖ For each **Index**:
  Index name, Structure (B+ tree vs. hash, etc.); Index key
❖ For each **View**:
  View name, and View definition
Statistics in the System Catalog

- RDBMS periodically collects stats about DB (instance)
- For each Table R:
  - Cardinality, i.e., number of tuples, \( NTuples (R) \)
  - Size, i.e., number of pages, \( NPages (R) \), or just \( N_R \)
- For each Index X:
  - Cardinality, i.e., number of distinct keys \( IKeys (X) \)
  - Size, i.e., number of pages \( IPages (X) \) (for a B+ tree, this is the number of leaf pages only)
  - Height (for tree indexes) \( IHeight (X) \)
  - Min and max keys in index \( ILow (X), IHigh (X) \)
Query Optimization

- Overview of Query Optimizer

- Physical Query Plan (PQP)
  - Concept: Pipelining
  - Mechanism: Iterator Interface

- Enumerating Alternative PQPs
  - Logical: Algebraic Rewrites

- Costing PQPs
**Q:** Does the hash-based aggregate have to wait till the entire output of the “upstream” hash join is available?

No! We can “pipeline” the output of the join – pass on a join output tuple as soon as it is obtained!
Concept: Pipelining

**Basic Idea:**
Do not force “downstream” physical operators to wait till the entire output is available.

**Benefits:**
Display output to the user incrementally.

**CPU Parallelism** in multi-core systems!

- Tuples
- File Scan
- Hash Join
- Hash-based Aggregate
- Aggregate
Concept: Pipelining

- Crucial for PQPs with workflow of many phy. ops.
- Common feature of almost all RDBMSs
- Works for many operators: SCAN, HASH JOIN, etc.

**Q:** *Are all physical operators amenable to pipelining?*

No! Some may “stall” the pipeline: “**Blocking Op**”

A blocking op. requires its output to be **Materialized** as a temporary table

Usually, any phy. op. involving **sorting** is blocking!
This phy. op. is blocking because we need to sort Movies and sort Ratings (materialize the output) before we can start any aggregate computations!
Query Optimization

❖ Overview of Query Optimizer
❖ Physical Query Plan (PQP)
   Concept: Pipelining
   Mechanism: Iterator Interface
❖ Enumerating Alternative PQPs
   Logical: Algebraic Rewrites
❖ Costing PQPs
Mechanism: Iterator Interface

- Software API to process PQP; makes pipelining easy to impl.
- Enables us to abstract away individual phy. op. impl. details
- Three main functions in usage interface of each phy. op.:
  - **Open()**: Initialize the phy. op. “state”, get arguments
    Allocate input and output buffers
  - **GetNext()**: Ask the phy. op. impl. to “deliver” next
    output tuple; pass it on; if blocking, wait
  - **Close()**: Clear phy. op. state, free up space
Query Optimization

- Overview of Query Optimizer
- Physical Query Plan (PQP)
  - Concept: Pipelining
  - Mechanism: Iterator Interface
- Enumerating Alternative PQPs
  - Logical: Algebraic Rewrites
- Costing PQPs
Overview of Query Optimizer

SQL Query

Parser

Logical Query Plan

Plan Enumerator

Plan Cost Estimator

Catalog

Physical Query Plan (Optimized)

To Scheduler/Executor
Enumerating Alternative PQPs

- Plan Enumerator explores various PQPs for a given LQP
- **Challenge**: Space of plans is huge! How to make it feasible?
- RDBMS Plan Enumerator has **Rules** to help determine what plans to enumerate, and also consults **Cost models**
- Two main sources of Rules for enumerating plans:
  - **Logical: Algebraic Rewrites**: Use relational algebra **equivalence** to rewrite LQP itself!
  - **Physical: Choosing Phy. Op. Impl.**: Use different phy. op. impl. for a given log. op. in LQP
Query Optimization

- Overview of Query Optimizer
- Physical Query Plan (PQP)
  - Concept: Pipelining
  - Mechanism: Iterator Interface
- Enumerating Alternative PQPs
  - Logical: Algebraic Rewrites
- Costing PQPs
Algebraic Rewrite Rules

- Rewrite a given RA query in to another that is *equivalent* (a logical property) but might be *faster* (a physical property)
- RA operators have some formal properties we can exploit
- We will cover only a few rewrite rules:
  
  **Single-operator** Rewrites
  - **Unary** operators
  - **Binary** operators
  
  **Cross-operator** Rewrites
Unary Operator Rewrites

- Key unary operators in RA: $\sigma \pi$
- Commutativity of $\sigma$
  \[
  \sigma_{p_1} (\sigma_{p_2} (R)) = \sigma_{p_2} (\sigma_{p_1} (R))
  \]
- Cascading of $\sigma$
  \[
  \sigma_{p_1} (\sigma_{p_2} (\ldots \sigma_{p_n} (R) \ldots )) = \sigma_{p_1 \lor p_2 \lor \ldots \lor p_n} (R)
  \]
- Cascading of $\pi$
  \[
  \pi_{A_1} (\pi_{A_2} (\ldots \pi_{A_n} (R) \ldots )) = \pi_{A_1} (R)
  \]

**Q:** Why are cascading rewrites beneficial?
Binary Operator Rewrites

- Key binary operator in RA: \( \otimes \)
- Commutativity of \( \otimes \): \( R \otimes S = S \otimes R \)
- Associativity of \( \otimes \): \( (R \otimes S) \otimes T = R \otimes (S \otimes T) \)

**Q:** Why are these properties beneficial?

**Q:** What other binary operators in RA satisfy these?
Cross-Operator Rewrites

- Commuting \( \sigma \) and \( \pi \)
  \[
  \sigma_p(A)(\pi_B(R)) = \pi_B(\sigma_p(A)(R))
  \]

- Combining \( \sigma \) and \( \times \)
  \[
  \sigma_p(R \times S) = R \Join_p S
  \]

- “Pushing the select”
  \[
  \sigma_p(A)(R \Join S) = \sigma_p(A)(R) \Join S
  \]
  \[
  \sigma_p(A)(R \times S) = \sigma_p(A)(R) \times S
  \]

- Commuting \( \pi \) with \( \times \) and \( \Join \)
  \[
  \pi_A(R \times S) = \pi_{A \cap R.*}(R) \times \pi_{A \cap S.*}(S)
  \]
  \[
  \pi_A(R \Join_{p(B)} S) = \pi_{A \cap R.*}(R) \Join_{p(B)} \pi_{A \cap S.*}(S)
  \]

\( A \subseteq B \)
\( A \subseteq S.* \)
\( B \subseteq A \)
Review Question

❖ Which of the following hold?

\[ \pi_A(R \times S) = \pi_A(R) \times S \quad A \subseteq R.* \]

\[ \pi_A(R \bowtie_{p(B)} S) = \pi_A(\pi_{C \cap R.*}(R) \bowtie_{p(B)} \pi_{C \cap S.*}(S)) \]

\[ \sigma_{p_1 \land p_2 \lor p_3}(R) = \sigma_{p_1}(R) \cap \sigma_{p_2}(R) \cup \sigma_{p_3}(R) \]

\[ \sigma_{p(A) \land q(B)}(R \bowtie S) = \sigma_{p(A)}(R) \bowtie \sigma_{p(B)}(S) \quad A \subseteq R.* \text{ and } B \subseteq S.* \]
Query Optimization

❖ Overview of Query Optimizer
❖ Physical Query Plan (PQP)
   Concept: Pipelining
   Mechanism: Iterator Interface
❖ Enumerating Alternative PQPs
   Logical: Algebraic Rewrites
❖ Costing PQPs

- Given a (rewritten) LQP, pick phy. op. impl. for each log. op.
- Recall various RA op. impl. with their I/O (and CPU costs)

\[ \sigma \quad \text{File scan vs Indexed (B+ Tree vs Hash)} \]
\[ \pi \quad \text{Hashing-based vs Sorting-based vs Indexed} \]
\[ \bowtie \quad \text{BNLJ vs INLJ vs SMJ vs HJ} \]

etc.

\[ \pi_B(\sigma_{p(A)}(R) \bowtie S) \]

3 options 3 options 4 options = 36 PQPs!

Q: With algebraic rewrites?!

❖ Are the indexes clustered or unclustered?
❖ Are there multiple matching indexes? Use multiple?
❖ Are index-only access paths possible for some ops?
❖ Are there “interesting orderings” among the inputs?
❖ Would sorted outputs benefit downstream ops?
❖ Estimation of cardinality of intermediate results!
❖ How best to reorder multi-table joins?

Query optimizers are complex beasts! Still a hard, open research problem!

❖ Since joins are associative, exponential number of orderings!

\[ R \bowtie S \bowtie T \bowtie U \]

❖ Left Deep tree

❖ Right Deep tree

❖ “Bushy” tree

❖ Almost all RDBMSs consider only left and right deep join trees

   Enables easy pipelining! Why?

❖ “Interesting orderings” idea from System R optimizer paper

❖ Dynamic program to combine enumeration and costing

“Access Path Selection in a Relational Database Management System” SIGMOD’79
Query Optimization

❖ Overview of Query Optimizer
❖ Physical Query Plan (PQP)
  Concept: Pipelining
  Mechanism: Iterator Interface
❖ Enumerating Alternative PQPs
  Logical: Algebraic Rewrites
❖ Costing PQPs
Overview of Query Optimizer

SQL Query

Parser

Logical Query Plan

Plan Enumerator

Plan Cost Estimator

Optimizer

Catalog

Physical Query Plan (Optimized)

To Scheduler/Executor
Costing PQPs

❖ For each PQP considered by the Plan Enumerator, the Plan Cost Estimator computes “Cost” of the PQP
   Weighted sum of I/O cost and CPU cost
   (Distributed RDBMSs also include Network cost)

❖ Challenge: Given a PQP, compute overall cost

❖ Issues to consider:
   Pipelining vs. blocking ops; cannot simply add costs!
   **Cardinality estimation** for intermediate tables!

Q: *What statistics does the catalog have to help?*
Costing PQPs

❖ Most RDBMSs use various heuristics to make costing tractable; so, it is approximate!

❖ Example: Complex predicates

\[ \sigma_{p_1 \land p_2}(R) \]

Suppose selectivity of \( p_1 \) is 5% and selectivity of \( p_2 \) is 10%

Q. What is the selectivity of \( p_1 \land p_2 \)? Not enough info!

But, most RDBMSs use the independence heuristic!

Selectivity of conjunction = Product of selectivities

Thus, \( \approx 0.05 \times 0.1 = 0.005 \), i.e., 0.5%
Query Optimization: Summary

- Plan Enumerator and Cost Estimator work in lock step:
  - **Rules** determine what PQPs are enumerated
    - Logical: Algebraic rewrites of LQP
    - Physical: Op. Impl. and ordering alternatives
  - **Cost models** and **heuristics** help cost the PQPs

- Still an active research area!
  - Parametric Q.O., Multi-objective Q.O.,
  - Multi-objective parametric Q.O., Multiple Q.O.,
  - Online/Adaptive Q.O., Dynamic re-optimization, etc.
Query Optimization

❖ Overview of Query Optimizer
❖ Physical Query Plan (PQP)
  Concept: Pipelining
  Mechanism: Iterator Interface
❖ Enumerating Alternative PQPs
  Logical: Algebraic Rewrites
❖ Costing PQPs
Review Question

```
SELECT COUNT(DISTINCT Date) FROM Ratings
```

Page size 8KB; Buffer memory 8GB; 8B for each field

Propose an efficient physical plan and compute its I/O cost.

<table>
<thead>
<tr>
<th>RatingID</th>
<th>Stars</th>
<th>Date</th>
<th>UID</th>
<th>MID</th>
<th>20m pages</th>
</tr>
</thead>
</table>

Hash-based Project on Date

COUNT Aggregate

Read

<table>
<thead>
<tr>
<th>NT = 4m</th>
</tr>
</thead>
</table>

Write T to disk

<table>
<thead>
<tr>
<th>NT = 4m</th>
</tr>
</thead>
</table>

Read

<table>
<thead>
<tr>
<th>NR = 20m</th>
</tr>
</thead>
</table>

Write T to disk

<table>
<thead>
<tr>
<th>Cannot fit hash table on T entirely; partition it</th>
</tr>
</thead>
</table>

Pipeline COUNT with deduplication phase

<table>
<thead>
<tr>
<th># buffer frames = 1m</th>
</tr>
</thead>
</table>

Total I/O = 28m pages
Page size 8KB; Buffer memory 8GB; 8B for each field

We are given a clustered AltRID B+ tree index on Date. Propose an efficient physical plan and compute its I/O cost. (Assume RecordID is 8B too)

SELECT COUNT(DISTINCT Date) FROM Ratings

ProjectionList is prefix of IndexKey
No GROUP BY; trivial pipelining
Overall, just one scan of leaf level

Leaf level size = \( \frac{2}{5} \times N_R = 8m \) pages
Propose an efficient physical plan and compute its I/O cost.

- Query plan is a Project on UID, followed by counting. 
- Size of intermediate file $T = 2m$ pages, i.e., $16$ GB. 
- Size of hash table on $T$: $F \times 16$ GB $> 4$ GB RAM. 
- But RAM is enough for regular hash-based Project. 
- Pipeline counting on top of output of Project. 
- I/O cost = $10m + 2 \times 2m = 14m$ pages.
Propose an efficient physical plan that does not materialize any intermediate data (fully pipelined) and compute its I/O cost.
SELECT \( \text{AVG(Stars)} \) FROM Ratings R, Movies M
WHERE R.MID = M.MID AND M.Director = "Christopher Nolan" AND R.UID = 1234;

8 KB pages; 4 GB RAM

\( \sigma_{\text{Director}=...} \) \( \sigma_{\text{UID}=1234} \)

\( \gamma_{\text{AVG(Stars)}} \)

Trivially pipelined
Hash table size < F * 100k * 8KB

Group by
Easy case
Hash join

Fully pipelined; no partitioning
I/O cost = 10.1m pages

Filescan
Director = ...

Filescan
UID = 1234
Propose an efficient physical plan that does not materialize any intermediate data (fully pipelined) and compute its I/O cost.

```
SELECT M.Year, AVG(R.Stars)
FROM Ratings R, Movies M, Users U
WHERE R.MID = M.MID AND R.UID = U.UID
    AND U.Age <= 30
GROUP BY M.Year
```
Review Question

SELECT M.Year, AVG(R.Stars)
FROM Ratings R, Movies M, Users U
WHERE R.MID = M.MID AND R.UID = U.UID
   AND U.Age <= 30
GROUP BY M.Year

Logical Query Plan

Should we rewrite the LQP? How? Why?
Review Question

\[ \gamma_{Year, AVG(Stars)} \]

LQP with right deep join tree

What physical op to use for each logical op? Pipelined?

Can hash tables fit in RAM?

What about this group by?

\[ \sigma_{Age \leq 30} \]

Movies

Users

Ratings

100k pages

10m pages

1b pages

\[ = F \times 100k \times 8KB \]

\[ = 1.12 \text{ GB} \]

\[ < F \times 10m \times 8KB \]

\[ = 112 \text{ GB} \]

\[ \text{Page size 8KB} \]

256GB RAM

Both hash tables fit in RAM together; fully pipelined join!
Review Question

\[ \gamma_{Year, \text{AVG}(\text{Stars})} \]

What about this group by?

Can the incremental statistics of all groups fit in RAM?

How many groups are there?

\[ \sigma_{\text{Age} \leq 30} \]

Movies

Users

Ratings

100k pages

10m pages

1b pages

\[ = F \times 100k \times 8\text{KB} \]

\[ < F \times 10m \times 8\text{KB} \]

\[ = 1.12 \text{ GB} \]

\[ = 112 \text{ GB} \]

Page size 8KB

256GB RAM

Need not bother; definitely smaller than Movies!

So, hash-based group by aggregate; partitioning not needed.
Review Question

\[ \gamma Year, AVG(Stars) \]

Hash-based group by

Final PQP

Hash join

Fully pipelined!
No intermediate data materialized

What is the total I/O cost?

Just 1 pass over each table

= 1010.1 million pages

Filescan

Hash join

Hash join

Ag \[ Age \leq 30 \]

Filescan; select Age <= 30

Users

100k pages

Movies

10m pages

Filescan

Ratings

1b pages