Discussion 10

Query Execution & Optimization

EECS 484

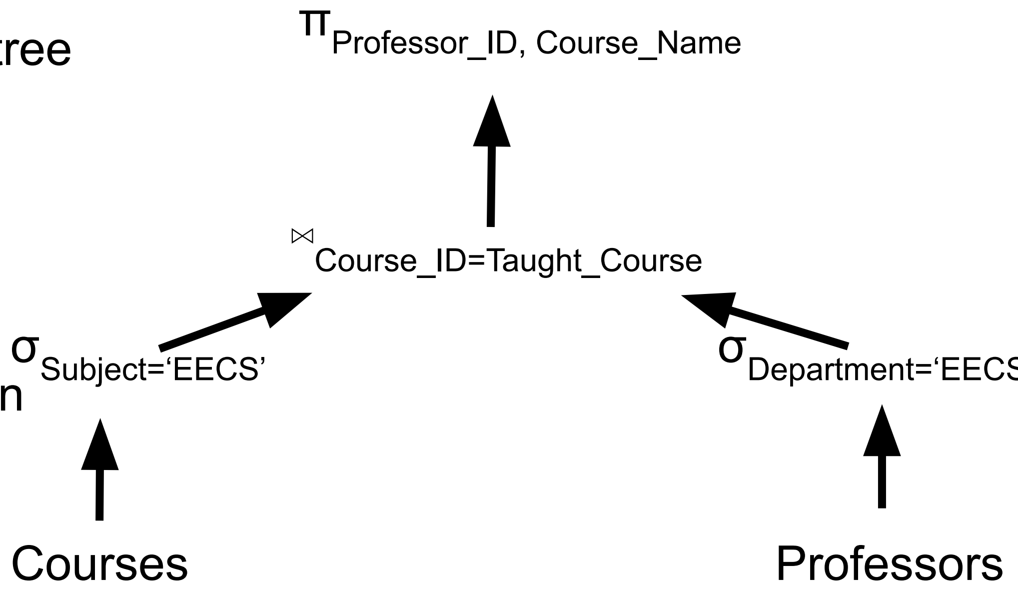
Logistics

- **HW 5** due **Today** at 11:45 PM ET
- **Project 4** due **Nov 22nd** at 11:45 PM ET
- **HW 6** released**,** due **Dec 6th** at 11:45 PM ET
- Feedback on Maizey
    - Have you used it? What was your experience?
    - Did you find it helpful? In what ways?
    - Any suggestions or feedback?

Query Execution

Query Plan

- The operators are arranged in a tree
- Data flows from the leaves of the tree up towards the root
- The output of the root node is the result of the query
- Use to create a query plan that can be optimized



Processing Models

- Define how a DBMS executes a query plan
- How do you pass data from one operator to the next?
- 3 approaches
  - Iterator Model
  - Materlization Model
  - Vectorization Model

Iterator Model

- Query operators are implemented as iterators
- Each iterator implements a **Next()** function
  - Returns the next tuple from its output
- Each operator implements a loop that calls Next() on its children until there are no more tuples to process
- Process **one tuple at a time**
- Used in almost every DBMS. Allows for tuple pipelining

Iterator Model

Suppose we have a relation R, and we want to find all the IDs with YOB = 2002.

R

ID	YOB
1	2000
2	2002
3	2002

$\pi\_ID(\sigma\_YOB=2002(R))$

π_ID: Next()
```
for t in child.Next():
  emit(projection(t, ID))
```

σ_YOB: Next()
```
for t in child.Next():
  if t.YOB == 2002: emit(t)
```

R: Next()
```
for t in R:
  emit(t)
```

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

Project iterator calls child.Next()

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))

for t in child.Next():
  if t.YOB == 2002: emit(t)

for t in R:
  emit(t)
```

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

Select iterator calls child.Next()

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))

for t in child.Next():
  if t.YOB == 2002: emit(t)

for t in R:
  emit(t)
```

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))
```

```
for t in child.Next():
  if t.YOB == 2002: emit(t)
```

R's iterator
emit the first tuple

```
for t in R:
  emit(t)
```

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

t.ID == 1
t.YOB == 2000

→

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))

for t in child.Next():
  if t.YOB == 2002: emit(t)

for t in R:
  emit(t)
```

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))
```

```
for t in child.Next():
  if t.YOB == 2002: emit(t)
```

R's iterator
emit the 2nd tuple

```
for t in R:
  emit(t)
```

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

t.ID == 2
t.YOB == 2002

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))

for t in child.Next():
  if t.YOB == 2002: emit(t)

for t in R:
  emit(t)
```

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

t.ID == 2
t.YOB == 2002

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))

for t in child.Next():
  if t.YOB == 2002: emit(t)

for t in R:
  emit(t)
```

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

project iterator
emit t.ID

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))
```

```
for t in child.Next():
  if t.YOB == 2002: emit(t)
```

```
for t in R:
  emit(t)
```

Results: 2

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

Project iterator calls child.Next()

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))
```

```
for t in child.Next():
  if t.YOB == 2002: emit(t)
```

```
for t in R:
  emit(t)
```

Results: 2

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

Select iterator
calls child.Next()

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))

for t in child.Next():
  if t.YOB == 2002: emit(t)

for t in R:
  emit(t)
```

Results: 2

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))
```

```
for t in child.Next():
  if t.YOB == 2002: emit(t)
```

R's iterator
emit the 3rd tuple

```
for t in R:
  emit(t)
```

Results: 2

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

t.ID == 3
t.YOB == 2002

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))
```

```
for t in child.Next():
  if t.YOB == 2002: emit(t)
```

```
for t in R:
  emit(t)
```

Results: 2

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

t.ID == 3
t.YOB == 2002

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))
```

```
for t in child.Next():
  if t.YOB == 2002: emit(t)
```

```
for t in R:
  emit(t)
```

Results: 2

Iterator Model

R

ID	YOB
1	2000
2	2002
3	2002

project iterator
emit t.ID

π_ID(σ_YOB=2002(R))

```
for t in child.Next():
  emit(projection(t, ID))

for t in child.Next():
  if t.YOB == 2002: emit(t)

for t in R:
  emit(t)
```

Results: 2, 3

Materialization Model

- Each operator **processes its input all at once** and **emits its output all at once**
- Results of the intermediate steps are stored on disk or in memory before starting the next operation
- Easy to understand and implement
- Better for OLTP workloads because queries only access a small number of tuples at a time
- Not good for OLAP queries with large intermediate results

Materialization Model

Consider the same example:

R

ID	YOB
1	2000
2	2002
3	2002

π_ID(σ_YOB=2002(R))

π_ID:

```
p_results = []
for t in child.Output():
  p_results.add(projection(t, ID))
return p_results
```

σ_YOB:

```
s_results = []
for t in child.Output():
  if t.YOB == 2002: s_results.add(t)
return s_results
```

R:

```
results = []
for t in R:
  results.add(t)
return results
```

Materialization Model

R

ID	YOB
1	2000
2	2002
3	2002

Project operator calls child.Output()

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Output():
  p_results.add(projection(t, ID))
return p_results
s_results = []
for t in child.Output():
  if t.YOB == 2002: s_results.add(t)
return s_results
results = []
for t in R:
  results.add(t)
return results
```

Materialization Model

R

ID	YOB
1	2000
2	2002
3	2002

Select operator
calls child.Output()

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Output():
  p_results.add(projection(t, ID))
return p_results
s_results = []
for t in child.Output():
  if t.YOB == 2002: s_results.add(t)
return s_results
results = []
for t in R:
  results.add(t)
return results
```

Materialization Model

R

ID	YOB
1	2000
2	2002
3	2002

return entire
relation R

$\pi\_ID(\sigma\_YOB=2002(R))$

```
p_results = []
for t in child.Output():
  p_results.add(projection(t, ID))
return p_results
s_results = []
for t in child.Output():
  if t.YOB == 2002: s_results.add(t)
return s_results
results = []
for t in R:
  results.add(t)
return results
```

Materialization Model

R

ID	YOB
1	2000
2	2002
3	2002

[(1, 2000), (2, 2002), (3, 2002)]

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Output():
  p_results.add(projection(t, ID))
return p_results
s_results = []
for t in child.Output():
  if t.YOB == 2002: s_results.add(t)
return s_results
results = []
for t in R:
  results.add(t)
return results
```

Materialization Model

R

ID	YOB
1	2000
2	2002
3	2002

return [(2, 2002), (3, 2002)]

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Output():
  p_results.add(projection(t, ID))
return p_results
s_results = []
for t in child.Output():
  if t.YOB == 2002: s_results.add(t)
return s_results
results = []
for t in R:
  results.add(t)
return results
```

Materialization Model

R

ID	YOB
1	2000
2	2002
3	2002

return [2, 3]

$\pi\_ID(\sigma\_YOB=2002(R))$

```
p_results = []
for t in child.Output():
  p_results.add(projection(t, ID))
return p_results
s_results = []
for t in child.Output():
  if t.YOB == 2002: s_results.add(t)
return s_results
results = []
for t in R:
  results.add(t)
return results
```

Results: 2, 3

Vectorization Model

- Combination of *iterator model* and *materizaition model*
  - each operator implements a Next() function
  - Each operator emits **a batch of tuples** instead of a single tuple
- Ideal for OLAP queries because it greatly reduces the number of invocations per operator

Vectorization Model

Consider a similar example, let batch size = 3:

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

π_ID(σ_YOB=2002(R))

π_ID:

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
```

σ_YOB:

```
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
```

R:

```
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

Project operator calls child.Next()

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

Select operator calls child.Next()

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

add to results until
the length of it >= 3

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

emit [(1, 2000), (2, 2002), (3, 2002)] →

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

Check selection condition

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

when child.Next() is empty, |s_results|=2

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

call child.Next() again

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

emit [(4, 2002), (5, 2002), (6, 2003)]

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

emit [(2, 2002), (3, 2002), (4, 2002)] →

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

perform projection

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

Results: 2, 3, 4

$A \xrightarrow{\text{emit [2, 3, 4]}} B$

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

Results: 2, 3, 4

Finish remaining data, emit [(5, 2002)]

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
results = []
for t in R:
  results.add(t)
  if |results| >= 3: emit(results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

Results: 2, 3, 4

Finish remaining data, emit [(5, 2002)]

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: or child.Next() is
empty: emit(s_results)
```

Vectorization Model

R

ID	YOB
1	2000
2	2002
3	2002
4	2002
5	2002
6	2003

Results: 2, 3, 4, 5

Finish remaining data, emit [5]

π_ID(σ_YOB=2002(R))

```
p_results = []
for t in child.Next():
  p_results.add(projection(t))
  if |p_results| >= 3 or child.Next() is empty: emit(out)
s_results = []
for t in child.Next():
  if t.YOB == 2002: s_results.add(t)
  if |s_results| >= 3: emit(s_results)
```

Access Methods

Access Methods

- An access method is the way that the DBMS *accesses the data* stored in a table
- Three basic approaches:
  - Sequential Scan
  - Index Scan
  - Multi-Index / "Bitmap" Scan

Sequential Scans

- For each page in the table:
  - Retrieve it from the buffer pool
  - Iterate over each tuple and check whether to include it
- Almost always the worst thing that the DBMS can do to execute a query
- Sequential Scan Optimizations:
  - Zone Maps
  - Late Materialization (refer to the lecture slides)

Zone Maps

- A way we can *slightly* improve sequential scan
- *Pre-computed* aggregates for the attribute values in a page (min, max, etc)
- DBMS checks the zone map first to decide whether to access the page

Zone Maps

- Suppose we have a relation R spans two pages, P1 and P2
- We want to find those rows with Val = 2002

P1

Val
2000
2002
2003

Zone Map for P1

Type	Val
MIN	2000
MAX	2003
AVG	2001.67
SUM	6005
COUNT	3

P2

Val
2000
2001
2001

Zone Map for P2

Type	Val
MIN	2000
MAX	2001
AVG	2000.67
SUM	6002
COUNT	3

Zone Maps

- Since 2002 is greater than the max value of P2, we don't need to access it
- Only need to access P1 to get the results

P1

Val
2000
2002
2003

Zone Map for P1

Type	Val
MIN	2000
MAX	2003
AVG	2001.67
SUM	6005
COUNT	3

P2

Val
2000
2001
2001

Zone Map for P2

Type	Val
MIN	2000
MAX	2001
AVG	2000.67
SUM	6002
COUNT	3

2002 > max value of the page

Index Scan

- Even with optimization methods, sequential scans can still be *slow*
- Solution: Index Scan / Multi-Index Scan
  - The DBMS **picks an index** to find the tuples that the query needs
  - How to know which index(es) to use? Index matching! (discussed earlier)
  - If there are *multiple indexes* that the DBMS can use for a query
    - Compute sets of Record IDs using each matching index
    - Combine these sets based on the query's predicates (union vs. intersect)
    - Retrieve the records and apply any remaining predicates

Query Optimization

Query Optimization

- There can be different ways to plan the same query
- There can be a big difference in performance based on which plan is used
- **Goal:** Organize order of steps in query to be most efficient

Logical Query Optimization

- Selection
  - Predicate Pushdown
    - Perform filters (selections) as early as possible
    - Break a complex predicate and push down
- Projection
  - Projection Pushdown
    - Perform them early to create smaller tuples and reduce intermediate results
    - Project out all attributes except the ones requested (in SELECT clause) or required (in WHERE clause)
- Join
  - The number of different join orderings for an n-way join ≈ 4^n
  - Too slow if we use logical query optimization
  - Use cost-based query optimization

Steps for Logical Query Optimization

1. Decompose predicates into their simplest forms to make it easier for the optimizer to move them around (Split Conjunctive Predicates)
2. Move the predicate to the lowest applicable point in the plan (Predicate Pushdown)
3. Replace all Cartesian Products with inner joins using the join predicates
4. Eliminate redundant attributes before pipeline breakers (ex: joins) to reduce materialization cost (Projection Pushdown)

Example for Logical Query Optimization

- Create the optimized RA tree for the following query
  - ```
    SELECT U1.User_ID, U2.User_ID
    FROM Users U1, Users U2
    WHERE U1.age = U2.age AND U1.MOB = ’7’ AND U2.MOB = ’9’
    ```
  - Split Conjunctive Predicates
  - Predicate Pushdown
  - Replace all Cartesian Products
  - Projection Pushdown
    ```

Example for Logical Query Optimization

- Create the optimized RA tree for the following query
  - ```
    SELECT U1.User_ID, U2.User_ID
    FROM Users U1, Users U2
    WHERE U1.age = U2.age AND U1.MOB = '7' AND U2.MOB = '9'
    ```
  - Split Conjunctive Predicates
  - Predicate Pushdown
  - Replace all Cartesian Products
  - Projection Pushdown
- Convert to RA

$\pi_{U1.User\_ID,\ U2.User\_ID}(\sigma_{U1.MOB='7'\ \wedge\ U2.MOB='9' \wedge U1.Age=U2.Age}(\text{Users U1 X Users U2}))$
    ```

Example for Logical Query Optimization

- Create the optimized RA tree for the following query
  - ```
    SELECT U1.User_ID, U2.User_ID
    FROM Users U1, Users U2
    WHERE U1.age = U2.age AND U1.MOB = '7' AND U2.MOB = '9'
    ```
  - **Split Conjunctive Predicates**
  - Predicate Pushdown
  - Replace all Cartesian Products
  - Projection Pushdown

$\pi_{U1.User\_ID,\ U2.User\_ID}(\sigma_{U1.MOB='7'}(\sigma_{U2.MOB='9'}(\sigma_{U1.Age=U2.Age}(\text{Users U1 X Users U2}))))$
    ```

Example for Logical Query Optimization

- Create the optimized RA tree for the following query
  - ```
    SELECT U1.User_ID, U2.User_ID
    FROM Users U1, Users U2
    WHERE U1.age = U2.age AND U1.MOB = '7' AND U2.MOB = '9'
    ```
  - Split Conjunctive Predicates
  - **Predicate Pushdown**
  - Replace all Cartesian Products
  - Projection Pushdown

$\pi_{U1.User\_ID,\ U2.User\_ID}(\sigma_{U1.Age=U2.Age}(\sigma_{U1.MOB='7'}(\text{ Users U1}) \text{ X } \sigma_{U2.MOB='9'}(\text{Users U2})))$
    ```

Example for Logical Query Optimization

- Create the optimized RA tree for the following query
  - ```
    SELECT U1.User_ID, U2.User_ID
    FROM Users U1, Users U2
    WHERE U1.age = U2.age AND U1.MOB = ’7’ AND U2.MOB = ’9’
    ```
  - Split Conjunctive Predicates
  - Predicate Pushdown
  - **Replace all Cartesian Products**
  - Projection Pushdown

$\pi_{U1.User\_ID,\ U2.User\_ID}((\sigma_{U1.MOB='7'}(\text{Users U1}) \bowtie_{U1.Age=U2.Age} (\sigma_{U2.MOB='9'}(\text{Users U2})))$
    ```

Example for Logical Query Optimization

- Create the optimized RA tree for the following query
  - `SELECT U1.User_ID, U2.User_ID`

    `FROM Users U1, Users U2`

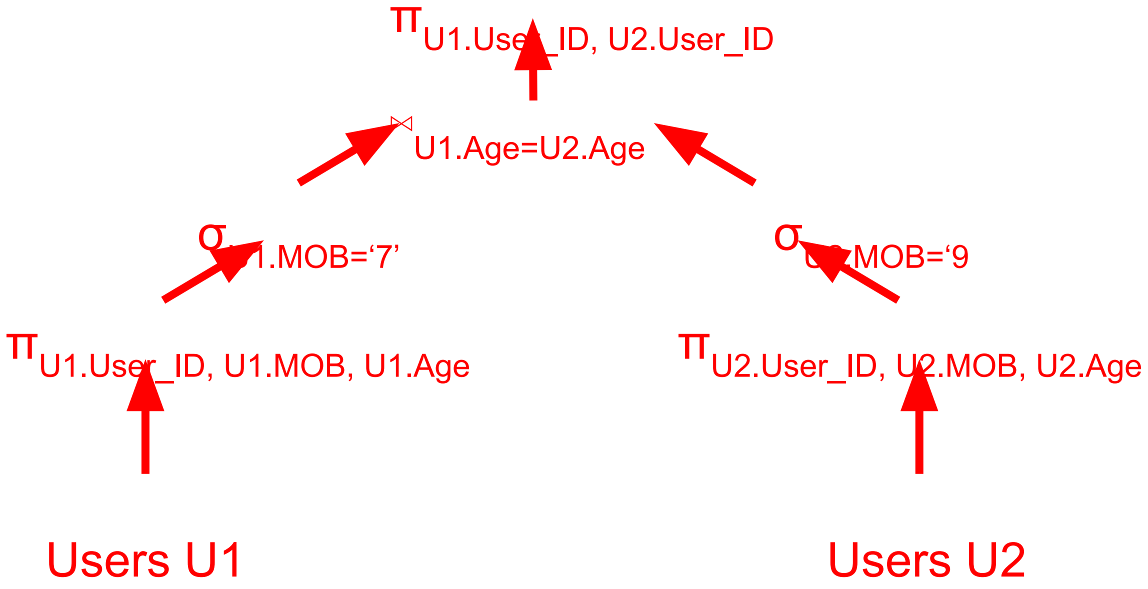
    `WHERE U1.age = U2.age AND U1.MOB = '7' AND U2.MOB = '9'`
  - Split Conjunctive Predicates
  - Predicate Pushdown
  - Replace all Cartesian Products
  - **Projection Pushdown**

$$\pi_{U1.User\_ID,\ U2.User\_ID}((\sigma_{U1.MOB='7'}(\pi_{U1.User\_ID,\ U1.MOB,\ U1.Age}(Users\ U1))) \bowtie_{U1.Age=U2.Age} (\sigma_{U2.MOB='9'}(\pi_{U2.User\_ID,\ U2.MOB,\ U2.Age}(Users\ U2))))$$

Example for Logical Query Optimization

- Tree Representation

$\pi_{U1.User\_ID,\ U2.User\_ID}((\sigma_{U1.MOB='7'}(\pi_{U1.User\_ID,\ U1.MOB,\ U1.Age}(\text{Users U1}))) \bowtie_{U1.Age=U2.Age}(\sigma_{U2.MOB='9'}(\pi_{U2.User\_ID,\ U2.MOB,\ U2.Age}(\text{Users U2}))))$



Cost-Based Query Optimization

- Generate an estimate of the cost of executing a particular query plan for the current state of the database

Selection Cardinality

- $SC(A,R) = N_R / V(A,R)$
  - $SC(A,R)$ – the average number of records with a value for an attribute A in R
  - N_R – number of tuples in R
  - $V(A,R)$ – number of distinct values for attribute A
- The number of tuples returned *on average* if we have a equality predicate on A
- Assumes equal distribution of values

Selection Cardinality

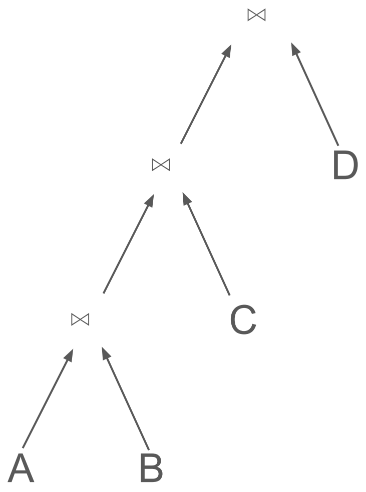
- Given a table Users with 600 records, we execute the following query:
  - ```
    SELECT User_ID
    FROM Users
    WHERE MOB = '7'
    ```
- What is the selection cardinality for MOB?
    ```

Selection Cardinality

- Given a table Users with 600 records, we execute the following query:
    - ```
      SELECT User_ID
      FROM Users
      WHERE MOB = '7'
      ```
- What is the selection cardinality for MOB?
- $N_{Users} = 600$
- V(MOB,Users) = 12
- SC(MOB,Users) = 600/12 = 50
- We estimate the number of tuples returned for this query is 50
      ```

Left Deep Plans

- As number of joins increases, number of alternative plans grows rapidly
  - n-way join $\approx 4^n$
  - Need to restrict search space
- Solution: only consider left-deep join trees



Left Deep Plans

- Steps for finding the optimal left deep plan
  - Enumerate possible orderings of relations
    - List out all the possible permutations of left deep plans
    - Prune plans with cross-products
  - Enumerate the plans for each operator in the plan
    - List out all the possible types of joins in each plan
  - Enumerate the access methods for each table
    - E.g. B+ Tree, Hash Index, File Scan
  - Estimate costs for the plans
  - Pick the cheapest one!
    - Can use dynamic programming to lower search space

HW5 Q2.2

For this question, suppose that you are given two relations, **R** and **S**, and that you want to perform an inner equijoin between the two. Suppose that you have B = 650 memory pages available, and suppose the following is also true of the two relations:

- Number of records in R = 20,000
- Number of pages in R = 2,500
- Number of records in S = 16,531
- Number of pages in S = 3,600

2.2) (4 points) **Sort Merge Join**
How many (a) total reads and (b) total writes are required?

Get started on Project 4!

We're here if you need any help!!

- Office Hours: Schedule is here, both virtual and in person offered
- Piazza
- Next week's discussion!!!