Query Processing and Optimization

Jennifer Widom
Ramakrishnan/Gehrke Chapters 10, 12

Steps in Database Query Processing

Parser – Checker - Views - Logical plan – Optim1 - Physical plan – Optim2 - Execution

Query string

→ **Parser** →

Query tree

→ Checker →

Valid query tree

→ View expander →

Valid tree w/o views

→ **Logical query plan generator** →

Logical query plan

→ Query rewriter (heuristic) →

Better logical plan

→ Physical query plan generator (cost-based)

Selected physical plan

→ Code generator →

Executable code

→ **Execution engine**

Running Example

Parser – Checker - Views - Logical plan – Optim1 - Physical plan – Optim2 - Execution

- Tables (what are the keys?):

  Student(ID, Name, Major)

  Course(Num, Dept)

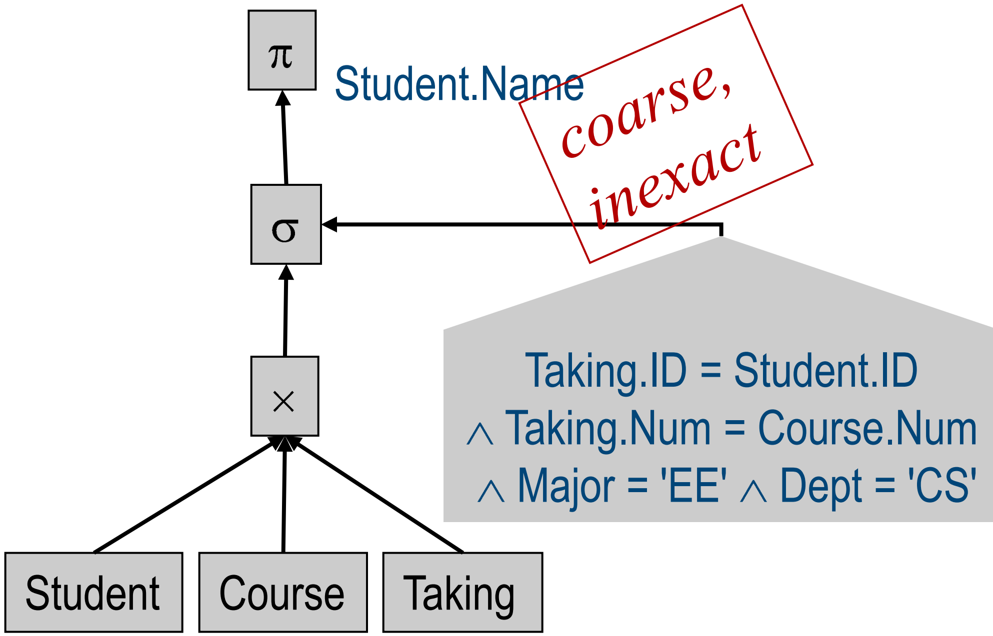
  Taking(ID, Num)

- Query to find all EE students taking at least one CS course:

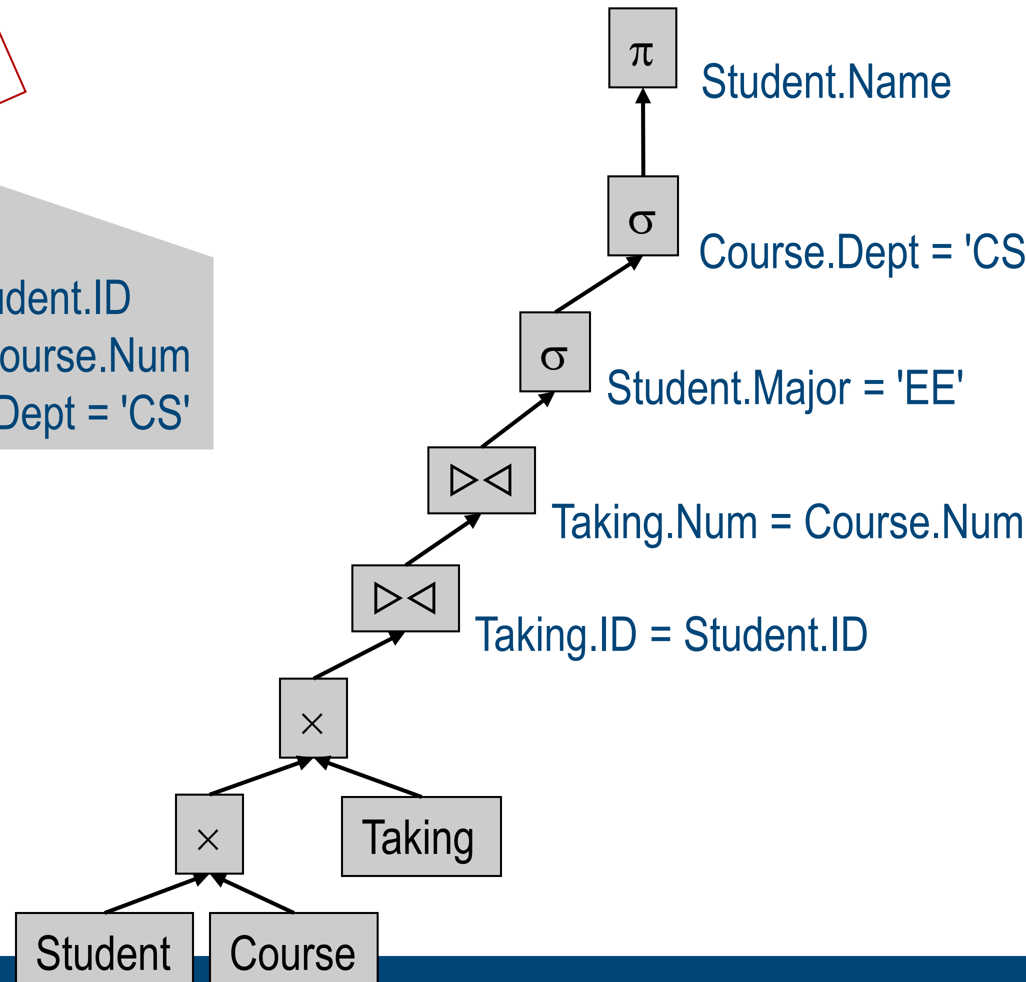
SELECT	Name	π
FROM	Student, Course, Taking	$\times$
WHERE	Taking.ID = Student.ID	$\bowtie$
AND	Taking.Num = Course.Num	$\bowtie$
AND	Major = 'EE'	σ
AND	Dept = 'CS'	σ

Logical Query Plan

Parser – Checker - Views - **Logical plan** – Optim1 - Physical plan – Optim2 - Execution

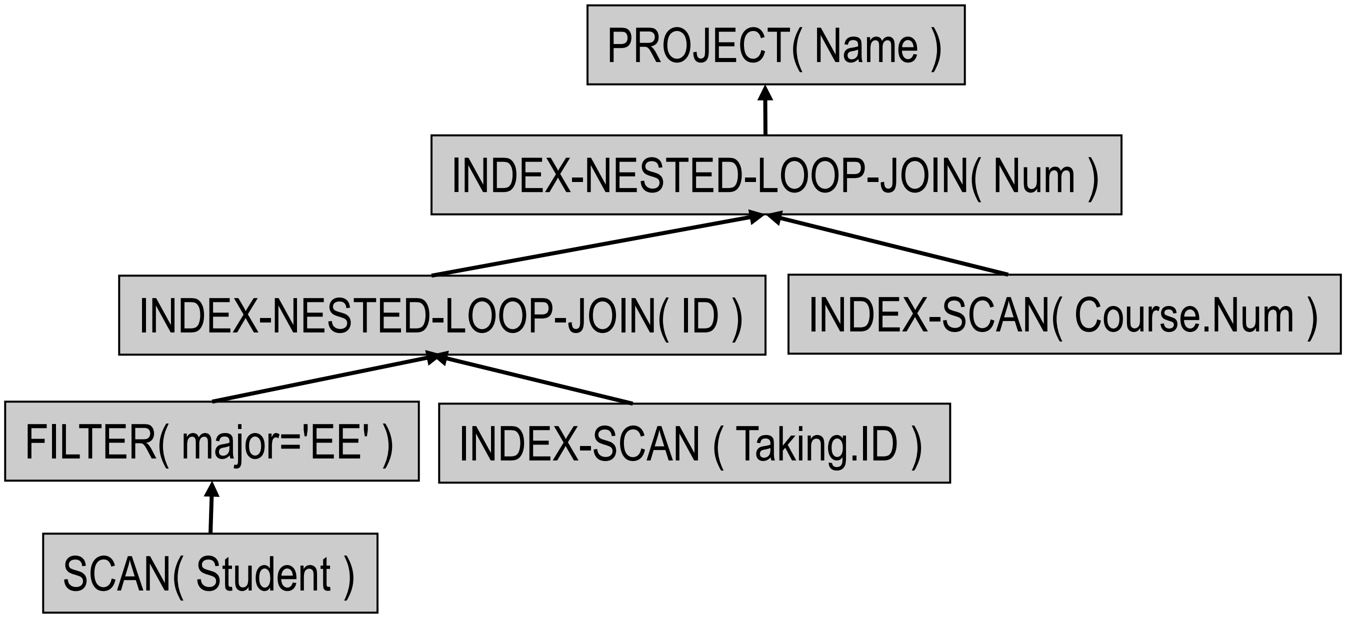



```
SELECT  Name
FROM    Student, Course, Taking
WHERE   Taking.ID = Student.ID
    AND Taking.Num = Course.Num
    AND Major = 'EE'
    AND Dept = 'CS'
```



Physical Query Plan

Parser – Checker - Views - Logical plan – Optim1 - **Physical plan** – Optim2 - Execution




```
SELECT  Name
FROM    Student, Course, Taking
WHERE   Taking.ID = Student.ID
   AND  Taking.Num = Course.Num
   AND  Major = 'EE'
   AND  Dept = 'CS'
```

one of manyManyMany possible plans, assumes particular index situation.

Sample Operator: Nested Loop Join

- Consider this equi-join query:

```
SELECT   *
FROM     Sailor S, Reserves R
WHERE    S.sid = R.sid
```

- Naïve, straightforward approach: combine all tuples, pick good ones

```
foreach tuple r in R do
    foreach tuple s in S do
        if r_i == s_j then add <r,s> to result
```

  - Assume there is no index, R small, S big: better R inner or S?
  - What if hash index on S?
  - *…this is what cost-based optimization considers!*

Physical Plan Generation

- ManyManyMany possible physical query plans for a given logical plan
- physical plan generator tries to select "optimal" one
  - Optimal wrt. response time, throughput
- How are intermediate results passed from children to parents?
  - Temporary files
    - *Evaluate tree bottom-up*
    - *Children write intermediate results to temporary files*
    - *Parents read temporary files*
  - Iterator interface (next)

Sample Query Plan

Parser – Checker - Views - Logical plan – Optim1 - Physical plan – Optim2 - Execution

```
SET EXPLAIN ON AVOID_EXECUTE;
SELECT   C.customer_num, O.order_num
FROM     customer C, orders O, items I
WHERE    C.customer_num = O.customer_num
         AND O.order_num = I.order_num
```

```
for each row in the customer table do:
  read the row into C
  for each row in the orders table do:
    read the row into O
    if O.customer_num = C.customer_num then
      for each row in the items table do:
        read the row into I
        if I.order_num = O.order_num then
          accept the row and send to user
        end if
      end for
    end if
  end for
end for
```

IBM Informix Dynamic Server

Ex: Iterator for Table Scan

Parser – Checker - Views - Logical plan – Optim1 - Physical plan – Optim2 - Execution

- open( )
  - Allocate buffer space
- getNext( )
  - If no block of R has been read yet:
    read first block from disk
    return (R==empty ? null : first tuple in block)
  - If no more tuple left in current block:
    read next block of R from disk
    return (R exhausted ? null : first tuple in block)
  - Return next tuple in block
- close( )
  - Deallocate buffer space

Ex: Iterator for Nested-Loop Join

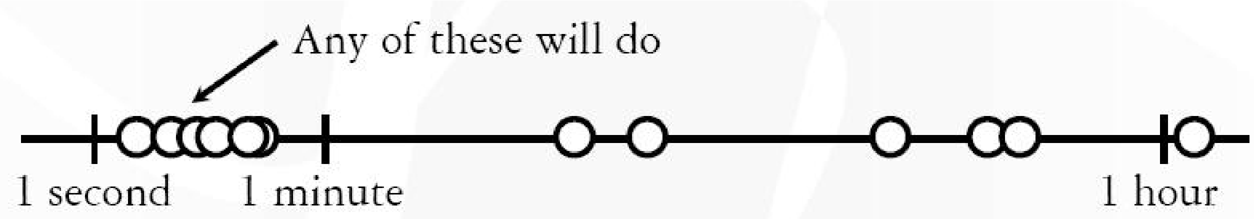
- open()
  - R.open(); S.open();
  - r = R.getNext();
- getNext()
  - repeat until r and s join:

```
s = S.getNext();
if (s = = null)
{       S.close(); S.open(); s = S.getNext();
        if (s = = null) return null;
        r = R.getNext();
        if (r = = null) return null;
}
```

  - return <r,s>;
- close()
  - R.close(); S.close();

Query Optimization

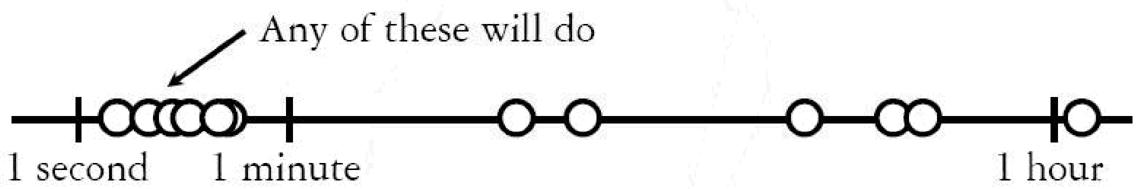
- Optimization = find better, equivalent plan
  - Equivalent = produces same result
  - Logical level optimization = aka heuristic optimization
  - Physical level optimization = aka cost-based optimization
- Two main issues:
  - For a given query, how to find cheapest plans?
  - How is cost of a plan estimated?



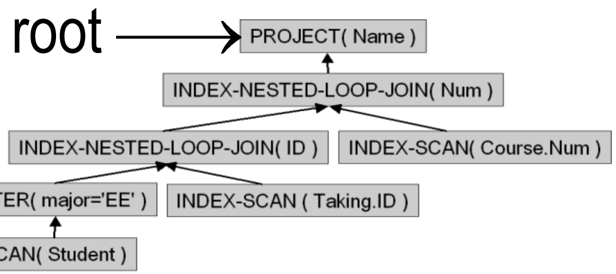
(II) Cost-Based Optimization

Parser – Checker - Views - Logical plan – Optim1 - Physical plan – Optim2 - Execution

- logical plan → (efficient) physical plan
- "Cost based" = driven by (estimated) costs of physical situation
  - concrete table sizes, indexes, data distribution, …
- Approach:
  - enumerate all (?) possible physical plans that can be derived from given logical plan
  - estimate cost for each plan
  - pick best (i.e., least cost) alternative
- Ideally: Want to find best plan; practically: Avoid worst plans!



Finale: Execution of Tree




```
result = {};
root.open();
do
{
        tmp = root.getNext();
        result += tmp;
} while (tmp != NULL);
root.close();
return result;
```

- Recursive evaluation of tree
  - Requests go down
  - Intermediate result tuples go up
- Often instead: compile into "database machine code" program