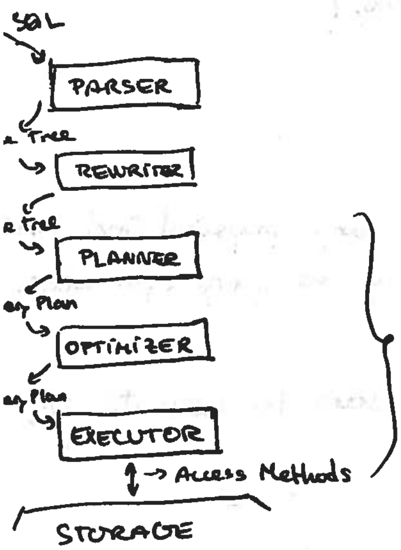
LECTURE ~~8~~/6

DB OPERATORS AND QUERY PROCESSING

1

→ We have seen so far how an input SQL query is parsed and what kind of rewrite rules can be used to simplify it and normalize it.

→ In this lecture we are going to see how do we express the SQL query in a representation that is used by the query executor to finally run the query and obtain results.



- The query planner will take the Parsed SQL query and build a query plan.
  - ↳ A query plan is a DAG where nodes are DB ops. and edges indicate data flow.
- The optimizer takes a query plan QP and returns a QP' which is semantically equivalent to QP but faster to execute in that machine and with that database.

- The executor knows the query plan and the access methods available, and is responsible for computing the result.

QUERY PLANS :

2

Given SQL ⟶ relational algebra ⟶ Query Plan (DAG)

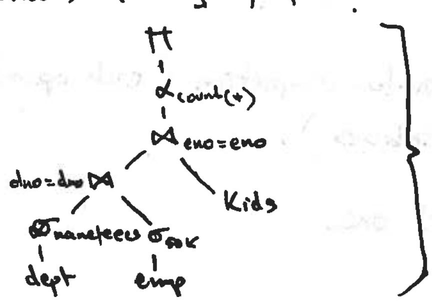
Schema: emp(<u>eno</u>, ename, sal, dno) dept(<u>dno</u>, dname, bldg) Kids(<u>Kno</u>, eno, kname, bday)

Query: select ename, count(*) from emp, dept, Kids ~~and~~ where emp.dno = dept.dno and
kids.eno = emp.eno and
emp.sal > 50000; and
dept.name = 'eecs'
group by ename
having count(*) > 7

Find the relational algebra expression for the query.

$$\pi\left(\alpha_{count(*),\, ename,\, count(*)>7}\left(Kids \bowtie_{eno=eno} \left(\sigma_{sal>50K}\, emp \bowtie_{dno=dno} \left(\sigma_{name=eecs}\, dept\right)\right)\right)\right)$$

What's the query plan?



As long as query plan remains semantically equivalent, we can perform two transformations, logical and physical.

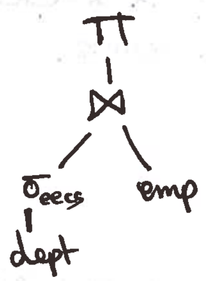
- ↳ Logical → Reordering of operators
- ↳ Physical → Choose specific implementation for operators.

Logical transformation. Predicate pushdown.

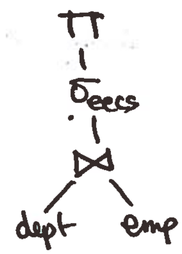
Query: select ename, count(*) from emp, dept where emp.dno = dept.dno and dept.dname = 'eecs'.

English: return all employee information for EECS employees.

Query Plan 1 :



Query Plan 2 :



- Are they equivalent?
- Which one do you prefer and why?

- In addition to these kind of logical transformations there are many physical impl. possib. Choosing the best plan possible in a given time budget is the task of the query optimizer.
- There will be an entire lecture dedicated to the query optimizer.
- Let's now assume we have selected one query plan and we want to execute it.

Query Execution:

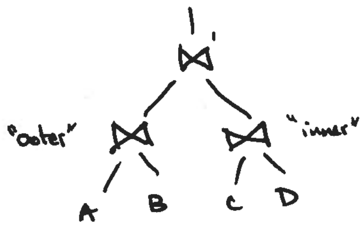
- The query executor takes a query plan and executes it to obtain the results, ~~given~~ How the query executes depends (~~in part~~) on the 'Physical Storage'.
  - ↳ In fact how data is organized on disk has a big effect on query performance.
  - ↳ For now, we are going to make the assumption that data, records (tuples), are simply stored in a file unordered. We have different files for different relations.
    - ↳ We will break at the end this assumption, and will get into details of physical storage and access methods in the next lecture.
    - ↳ So, assume you can read all (unordered) tuples from disk (each table corresponds to 1 file)

- How can we execute any (arbitrary) query plan?
  - ↳ There are different strategies. One that works well and that is broadly implemented is the "iterator" model. Also called "Volcano" or "Pipeline" model.
    - ↳ Each operator implements (at least) void open() ; Tuple next(); void close()
    - ↳ This model "pulls" tuples from the top.
  - ↳ Tuple-at-a-time has several advantages:
    - ↳ Easy to control intermediate results.
    - ↳ Allows stopping early when done.
  - ↳ Other alternatives: batch-at-a-time (reduce function call) take advantage of 'vectorization'., run-to-completion (each operator finishes before next runs. Query compilation (In-memory databases).

- Notion of pipelining. Results of one operator can be fed to ~~the~~ next one.

Query Plan Types:

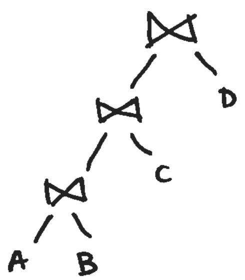
- Left deep vs bushy




```
for t1 in outer:
    for t2 in inner:
        if p(t1, t2):
            emit join(t1, t2)
```

- The biggest disadvantage of this plan is that we need to either run ⋈ and store in memory while ⋈' executes, or recompute its result every time.

- Left deep plan



- Notice in this case it's not necessary to materialize or recompute results.

---

1) CONTEXT: WHY DO WE CARE ABOUT ALL THIS?

- Advantages of relational model are logical and physical independence. A potential disadvantage was that "it seems hard to implement it and make it work fast".
- RDBMSs are all designed to make managing data practical. Declarative aspect of relational algebra and SQL means it is possible to optimize the query to achieve good performance.
    - ↳ ~~One~~ goal of the class is to understand how these systems are built. (and why like this

2) RELATIONAL ALGEBRA and EXTENSIONS.

- Relational algebra defines select (σ), projection (π), rename (ρ), set-union (∪ set-difference ($-$), cartesian product ($\times$), set-intersection ($\cap$)
    - ↳ ~~set intersection of X, Y is like (X ∪ Y) −~~
    - ↳ join is applying a selection predicate to a cartesian product.
        - ↳ $\sigma_{pred}(A \times B)$ usually represented as $A \bowtie_{pred} B$
    - ↳ group by (aggr). ${}_{gr\,attr}\alpha_{aggr\,f.}$ → also G is used. Also γ. [Not part of relational algebra, just an extension]. → 'Having' represented as condition on the functio
    - ↳ order by must appear top downstream. Or else results will be unordered. Same with 'limit'
    - ↳ generalized projection: allows arithmetic operations in the projection list.

3) LOGISTICS

- Lab 1 due Wed. PSET 2 due Wed. Final Projects teams due Friday (then 1 week to pre-proposal, short abstract). Then 1 more week to Final Project proposal.