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 as input and returns a query plan which is semantically equivalent to the input 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:**

Given SQL \rightarrow \text{relational algebra} \rightarrow \text{Query Plan (DAG)}

Schema: \begin{align*}
\text{emp} & : (\text{eno}, \text{ename}, \text{sal}, \text{dno}) \\
\text{dept} & : (\text{id}, \text{dname}, \text{bdby})
\end{align*}

Query: \text{select ename, count(*) from emp, dept, kids where emp.dno = dept.dno and}

\begin{align*}
\text{kids.eno} & = \text{emp.eno} \\
\text{emp.sal} & > 50000; \text{ and}
\end{align*}

\text{dept.name} = \text{'eecs'}

\text{group by ename}

\text{having count(*) > 7}

Find the relational algebra expression for the query.

\[ \sigma_{\text{ename}, \text{count(*) > 7}} (\text{Kids} \pi_{\text{eno} = \text{eno}} (\sigma_{\text{sal} > 50000} \text{emp} \pi_{\text{dno} = \text{dno}} (\sigma_{\text{name} = \text{'eecs'} \text{dept}}))) \]

What's the query plan?

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

- Logical \rightarrow \text{Reordering of operators}
- Physical \rightarrow \text{Choose specific implementation for operators}
Logical transformation: Predicate pushdown.

Query: `select name, count(*) from emp, dept where emp.dno = dept.dno and dept.dept like 'ees'.`

English: return all employee information for 'ees' employees.

\[ \text{Query Plan 1:} \]

\[ \text{Query Plan 2:} \]

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

In addition to these kinds of logical transformations, there are many physical implementations. 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 query optimizers.

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. 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 a 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 calls), 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

```
  (outer)
   A   B
   / \  /
  C   D
```

- Left deep plan

```
   /  /
  A   B
```

- The biggest disadvantage of this plan is that we need to either run $P_1$ and store in memory while $P_2$ executes, or recompute its result every time.

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

---

**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.

The goal of the class is to understand how these systems are built, and why they look like they do.

**Relational Algebra and Extensions.**

- Relational algebra defines select ($\sigma$), projection ($\Pi$), rename ($\pi$), set-union ($\cup$), set-difference ($\setminus$), cartesian product ($\times$), set-intersection ($\cap$).

- $\sigma_{\text{pred}} (R \times B)$ is like $(R \cup B)$.

- Join is applying a selection predicate to a cartesian product. $\text{group by (aggr)}$.

- Group by (aggr). $\text{group by (aggr)}$.[Not part of relational algebra, just an extension].

- Order by must appear top-downstream. Or else results will be unordered. Same with "limit".

- Generalized projection: allows arithmetic operations in the projection list.

---

**Logistics**

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