Introduction to Database Internals

The core question we want to answer in the next 2-3 lectures is: what happens from the moment a user writes a SQL query to the moment the results are returned?

1. Consider 2 types of clients, an application and an analyst with a SQL terminal. Queries can be triggered by an action in the application (1st case) or by submitting the query in the terminal (2nd case).

2. In both cases, the RDBMS works as a server. Clients can be local or remote.

3. When they are remote, consider the problems of interconnecting different languages to the DB; ORM (Object Relational Mapper), Django, Hibernate, ActiveRecord, SQLAl<...>my.

4. Consider the problem of shipping data through the network.

General architecture of a RDBMS:

- Specifically critical path from SQL to results:

  - RDBMS is by now very mature. Multiple generations of researchers and practitioners over decades.
  - However still changing due to new workloads and hardware.
  - Commercial success means they were (and are) thought resources to evolve these systems.

**Lecture 4 Outline:**

- Process Model. How/Who executes the SQL query?
- Parser and Rewriter.
Process Models

- Who takes responsibility of executing the query end-to-end?

- Naturally, this depends on the hardware characteristics of the underlying platform.
  - Single-node or cluster (parallel databases)?
  - Single processor or multiprocessor architecture?

- For this discussion, we assume single-processor, single-node. We will be relaxing these assumptions once we understand the basic "process models".

**Process-per-Worker, Thread-, Pool-**:

- **Process per Worker**
  - A full OS-process with its own address space (memory)
  - OS kernel in charge of scheduling

- **Thread per Worker**
  - All threads share the same address space
  - Threads scheduled by kernel

- **Pool of Workers**
  - Pool of processes
  - Bound resources to the size of the pool

**Pros**:
- Easy to implement
- Dealing with shared data structures
- Scalability, fixed, constant man. per worker
- Process switch is expensive
- Dos?

**Cons**:
- - Scalability
  - Multi-threaded application
  - Portability
  - DOS?

**Pros**:
- Those of process model
- Better scalability than process model

- For any process model, one has to consider 'admission policies' to control the concurrent # users/queries running in the system at a given time.

  - Goal: should I run the query now or wait? Achieve graceful degradation.

  - One way of implementing a basic admission policy is to constrain the max. number of concurrent connections.

  - A more precise implementation would involve estimating the 'cost' of the query (CPU, memory, disk accesses) and decide whether the query runs now or not.

- Different systems choose different process models.
Let's assume there's a chosen process model, this does not affect the next discussion.

- We discuss each component as a logically distinct piece of software with clear inputs, outputs and goals. In practice the implementation of some of these components may be intertwined.

**Parser:**

- Is the query correct and is it valid?
  1. There's a SQL standard, but nobody follows it and each SQL version/system is different.
  2. The parser must check if the SQL is correct for that system.
  3. Resolves names and references to determine if it's valid.
    1. Canonicalizes each table/column name.
    2. Use the 'catalog' to check attribute names and types.
  4. Security checks
    1. Is the user allowed to read this table?
    2. When is this not possible? Consider row-level security.

**Rewriter:**

- Goal: Simplify query without changing its semantics and without accessing to the actual data.
  1. This module/component usually operates on an internal representation of the query and not the SQL string. Sometimes implemented with the 'optimizer.'

- There are different kinds of transformations the rewriter typically performs:
  1. **View Expansion**: If an input query is defined over a view, express it in terms of the underlying tables.

**Example:**

```sql
create view SALS as
(SELECT dept, avg(salary) as sal
 FROM EMP
 GROUP BY dept;
)

SELECT sal FROM SALS
 WHERE dept = 'Eecs';
```

```sql
REWRITTEN QUERY
SELECT sal FROM (
SELECT dept, avg(salary) as sal
 FROM EMP
 GROUP BY dept
)
 WHERE dept = 'Eecs';
```
Constant arithmetic evaluation and logical rewriting of predicates

Example:

\[
\text{select id from EMP where age} > 40 + 5 \quad \text{AND} \quad \text{age} < 43; \\
\]

\[
\text{arithmetic eval.} \\
\text{logical rewriting (Boolean logic)}
\]

Logical rewriting can also help by adding more information to the query.

Example:

\[
A \cdot x < 10 \quad \text{AND} \quad B \cdot y = A \cdot x
\]

It is possible to determine that \( B \cdot y < 10 \) for this to evaluate true.

We only need to scan \( B \cdot y < 10 \) instead of all \( B \cdot y \).

Subquery flattening

Goal is to facilitate the query optimizer's job.

Flatten nested queries.

Example:

```sql
CREATE VIEW SALS AS (  
  SELECT distinct dept, sal from emp;
)

SELECT avg(sal) FROM SALS;
```

```
<table>
<thead>
<tr>
<th>dept</th>
<th>sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>eecs</td>
<td>50k</td>
</tr>
<tr>
<td>eecs</td>
<td>50k</td>
</tr>
<tr>
<td>me</td>
<td>50k</td>
</tr>
<tr>
<td>order</td>
<td>50k</td>
</tr>
</tbody>
</table>
```

Example:

```sql
SELECT avg(sal) FROM (  
  SELECT distinct dept, sal from emp;
)
```

```sql
SELECT avg(distinct sal) FROM emp;
```