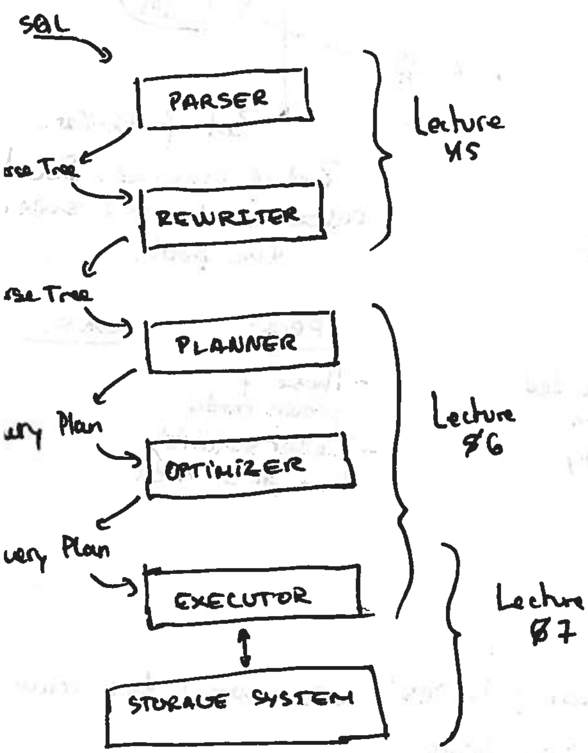
LECTURE 4

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?
  - ↳ 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).
  - ↳ In both cases, the RDBMS works as a server. Clients can be local or remote.
    - ↳ When they are remote, consider the problems of interconnecting different languages to the DB. ORM (Object Relational Mapper), Django, Hibernate, ActiveRecord, SQLAlchemy
    - ↳ Consider the problem of shipping data through the network.

- General architecture of a RDBMS; specifically critical path from SQL to results:



- RDBMS architecture 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) enough resources to evolve these systems.

Lecture 4 Outline:

→ Process Model. How/Who executes the SQL query?

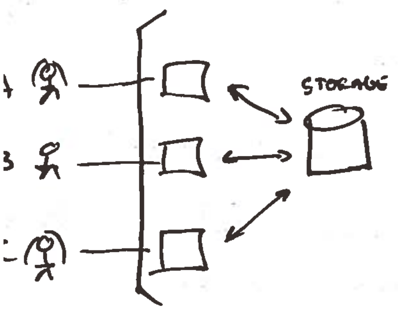
→ Admission Control / Authorization. "Always think of security".

→ Parser and Rewriter.

Process Models

- Who takes the 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-:

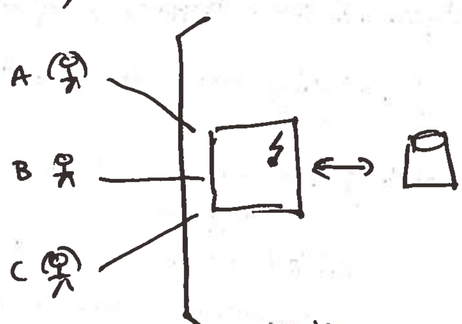


Process per Worker

A full OS-process with its own address space (memory)
OS Kernel in charge of scheduling

PROS:
- Easy to implement
- OS isolation, security

CONS:
- Dealing with shared data structures.
- Scalability. Fixed, constant mem. per worker
- Process switch is expensive
- DOS?

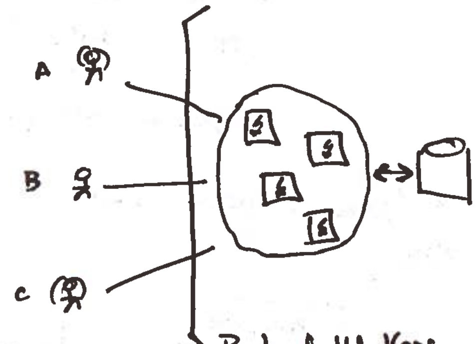


Thread per Worker

All threads share the same address space.
Threads scheduled by Kernel.

PROS:
- Scalability
  - ↳ memory
  - ↳ context switch

CONS:
- Multi-threaded application
- Portability
- DOS?



Pool of Workers.

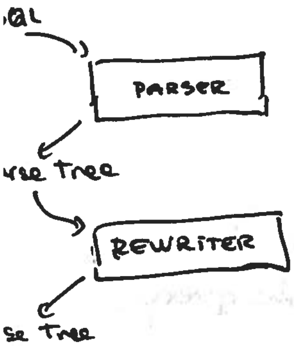
Pool of processes. Bound resources to the size of the pool.

PROS:
- Those of process model
- Better scalability than process model.

CONS:

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

Rewriter:

- Goal: Simplify query without changing its semantics and without accessing to the actual data.
  - ↳ 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:
  - ↳ View Expansion: If an input query is defined over a view, express it in terms of the underlying tables.
  - ↳ Example: $\boxed{\text{SCHEMA}}$ EMP: (id, salary, age, dept)

$\boxed{\text{VIEW}}$
```
create view SALS as (
    select dept, avg(salary) as sal from EMP group by dept;
)
```

$\boxed{\text{QUERY}}$
```
select sal from SALS where dept = 'eecs'
```

$\boxed{\text{REWRITTEN QUERY}}$
```
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

↳ some queries can be answered without touching any data.

↳ Example:

select id from EMP where age > $\underbrace{40+5}_{\text{arithmetic eval.}}$ AND age < 43;

logical rewriting (Boolean logic)

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

↳ Example:

A.x < 10 AND B.y = A.x

↳ It is possible to determine that B.y < 10 for this to evaluate true.

↳ We only need to scan B.y < 10 instead of all B.y.

↳ Subquery flattening

↳ Goal is to facilitate the query optimizer's job.

↳ Flatten nested queries.

DON'T INCLUDE

↳ Example:

~~QUERY select avg(sal) as sal from EMP where dept = 'eecs' group by dept;~~

VIEW create view SALS as (
select distinct dept, sal from emp;
)

QUERY select avg(sal) from SALS;

eid	dept	sal	...
1	eecs	100k	
2	eecs	100k	
3	me	50k	
4	arch	50k	

Select avg(sal) from (
select distinct dept, sal from emp;
)

select avg(distinct sal) from emp;

↳ Example: