MySQL's Concrete Architecture (Query Processor)

Tabs vs. Spaces

Varsha Ragavendran
Nisha Sharma
Davood Anbarnam
Ayman Abualsunun
Anton Sitkovets
Glib Sitiugin
Kevin Arindaeng

Introduction

Query Processor is within the Logical Layer of MySQL

- Processes queries
- Determines how to execute the query
- Query must be semantically and syntactically valid
- Executes the Query in optimized manner

Overview

Derivation Process

System Architecture

Overall Query Processor Architecture Analysis

Subsystem Analysis

Use Cases

Conclusions

Lessons Learned

Derivation Process

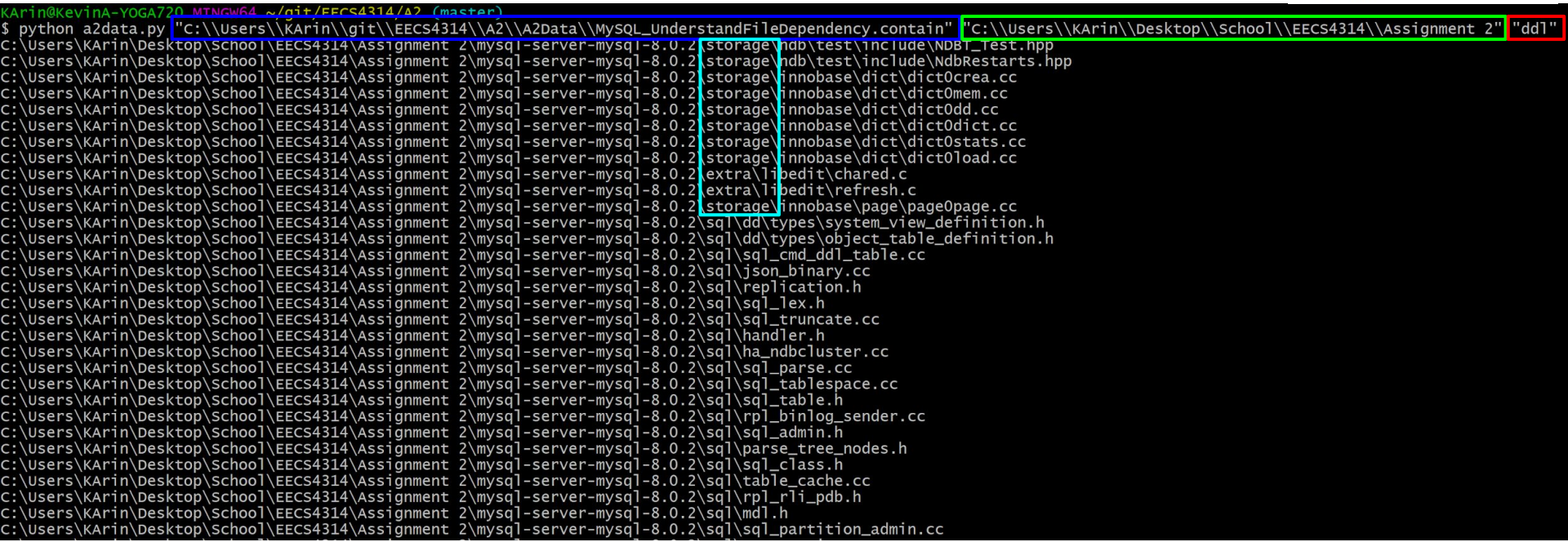
https://github.com/azkevin/EECS4314/blob/master/A2/a2data.py

```
sys.argv[1] = .contain file
sys.argv[2] = directory of MySQL source
sys.argv[3] = keyword to search for

'''
import sys
import re
import string
import os

# open the .contain file
f = open(sys.argv[1], 'r')
for line in f.readlines():
    # get the absolute path of each file in the .contain file
    mysql_file_path = os.path.join(sys.argv[2], os.path.normpath(line.split()[-1]))
    # make sure it's a file before opening; some of them are directories
    if os.path.isfile(mysql_file_path):
        with open(mysql_file_path) as mysql_file:
            try:
                for mysql_file_line in mysql_file:
                    # if the keyword is in your file, print out file to system out, then go onto the next file.
                    if sys.argv[3] in mysql_file_line:
                        print(mysql_file_path)
                        break
            # some files give a UnicodeDecodeError when analyzing its text; skip them
            except UnicodeDecodeError:
                pass;
```

Derivation Process



Look for **keywords, comments**, **name of the file**, and **file directory structure** then add relevant files of the subsystem to the .contain file

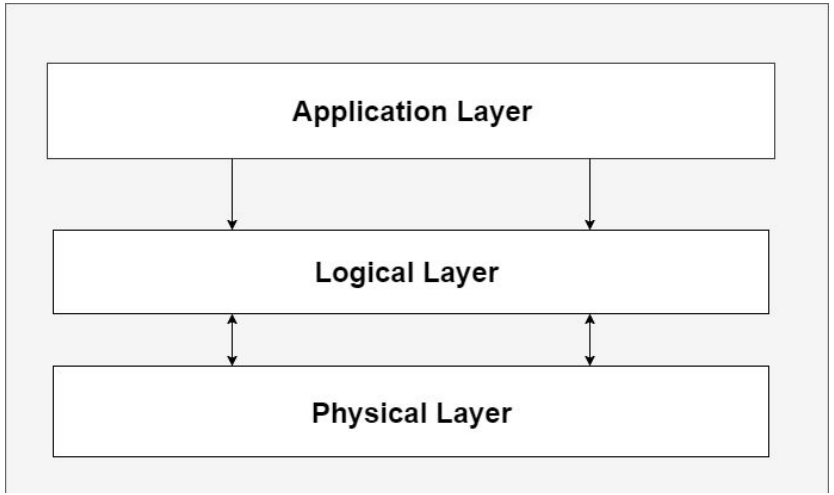
System Architecture

Key Findings:

- Concrete Architecture reveals that there Exist
  - Two way dependencies between different components and subsystems
  - Instead of One way dependencies that were seen in the conceptual Architecture

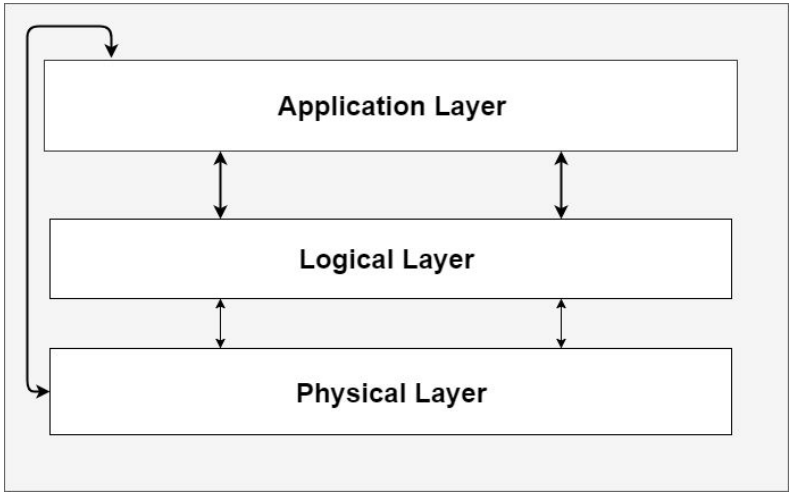
System Architecture

Key Findings:

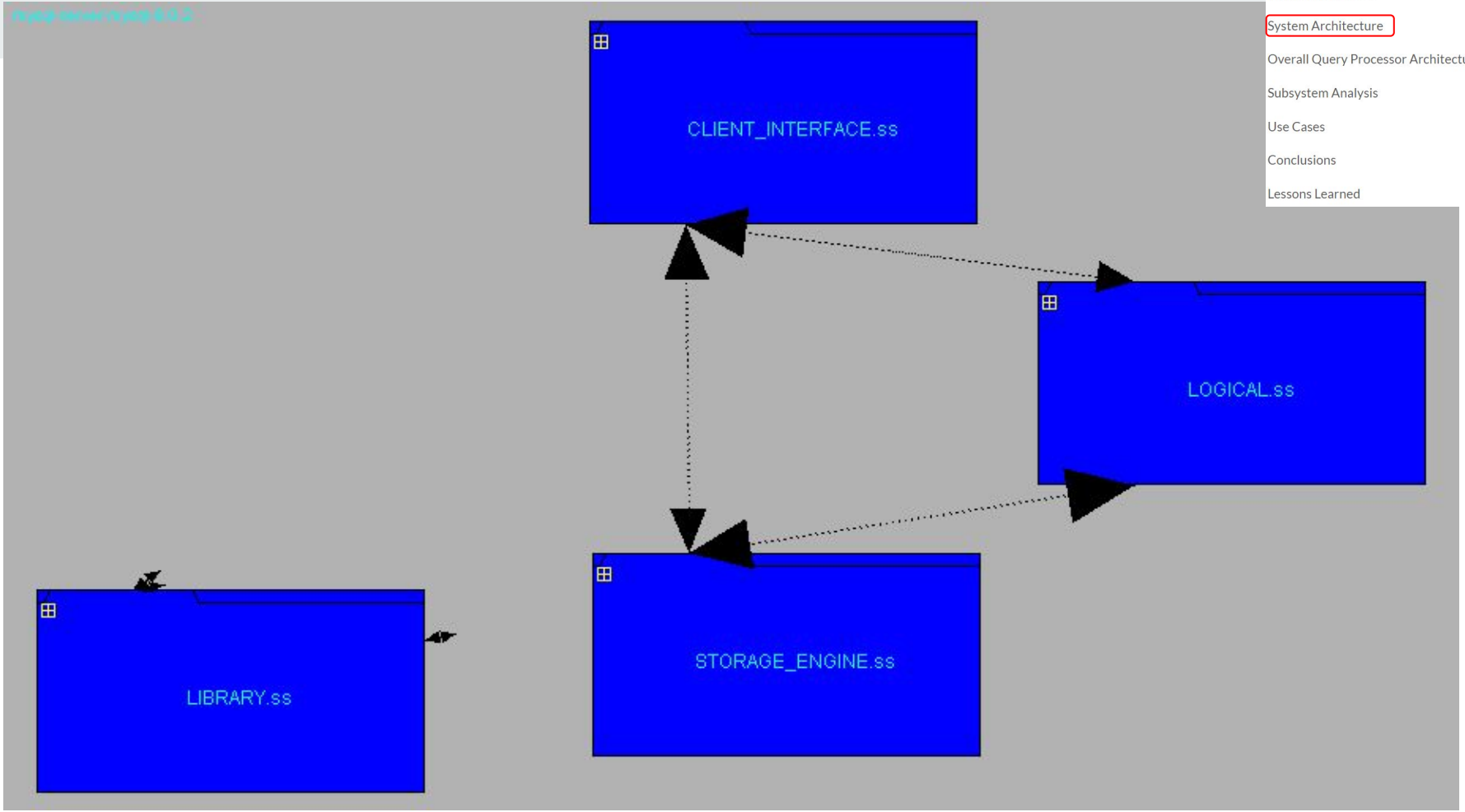


Simpler Conceptual Architecture

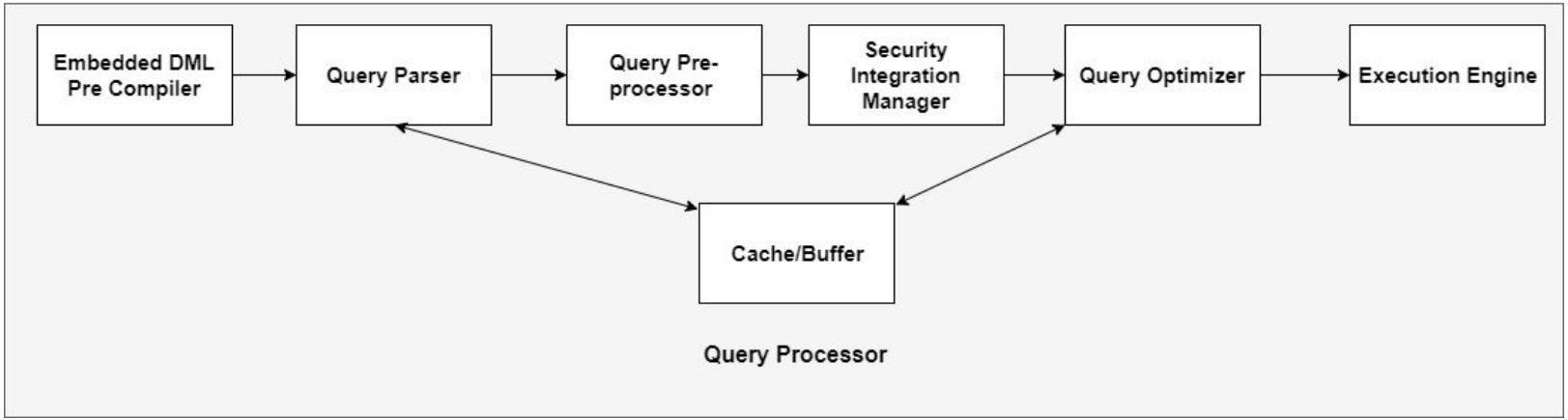
VS.



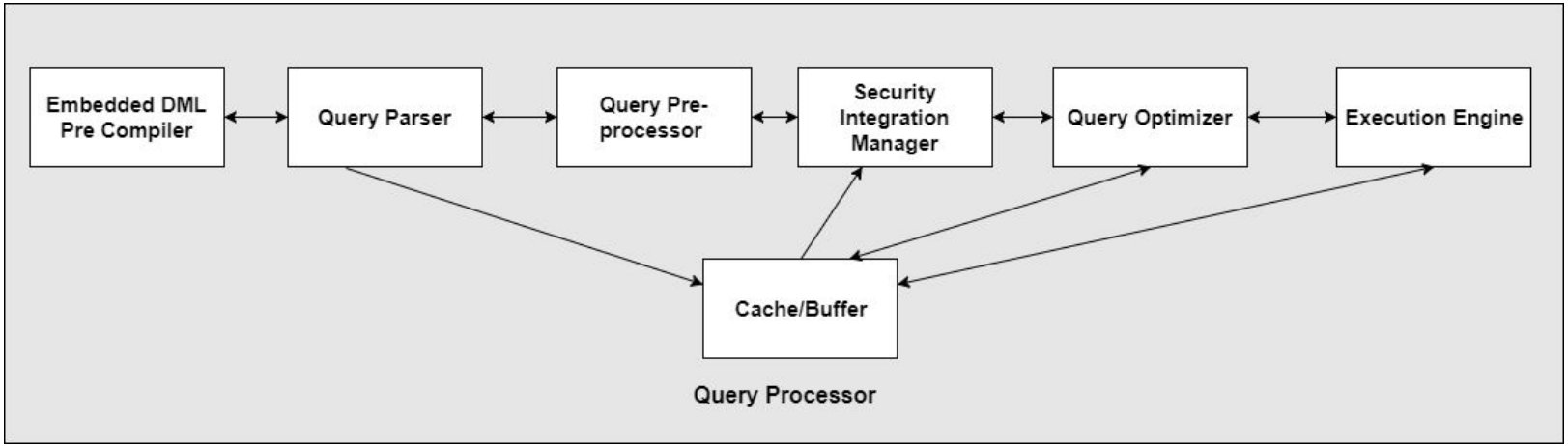
Simpler Concrete Architecture



Query Processor Conceptual Architecture



Query Processor Concrete Architecture



DDL Compiler

- Data Description Language (DDL): Syntax to create or alter the structure of the database.

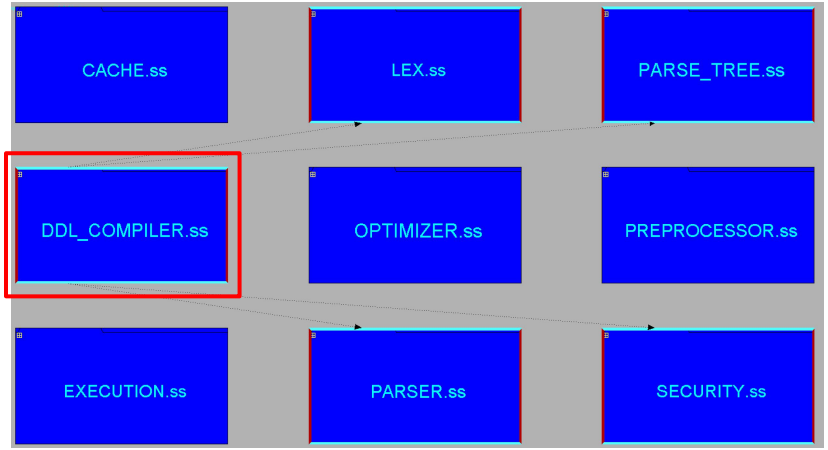
Example SQL Statements:

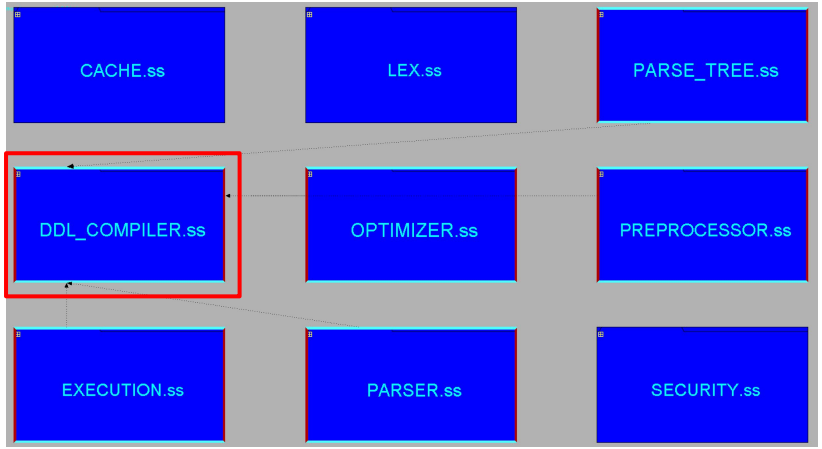
- CREATE
- ALTER
- DROP
- RENAME
- TRUNCATE

```
2
3   CREATE TABLE `data_log` (
4     `REC_ID` int(11) NOT NULL AUTO_INCREMENT,
5     `DATESTAMP` datetime DEFAULT NULL,
6     `SCANNER_ID` varchar(25) NOT NULL,
7     `BARCODE` varchar(255) DEFAULT NULL,
8     `WEIGHT` DOUBLE DEFAULT NULL,
9     PRIMARY KEY (`REC_ID`)
10  );
11
```

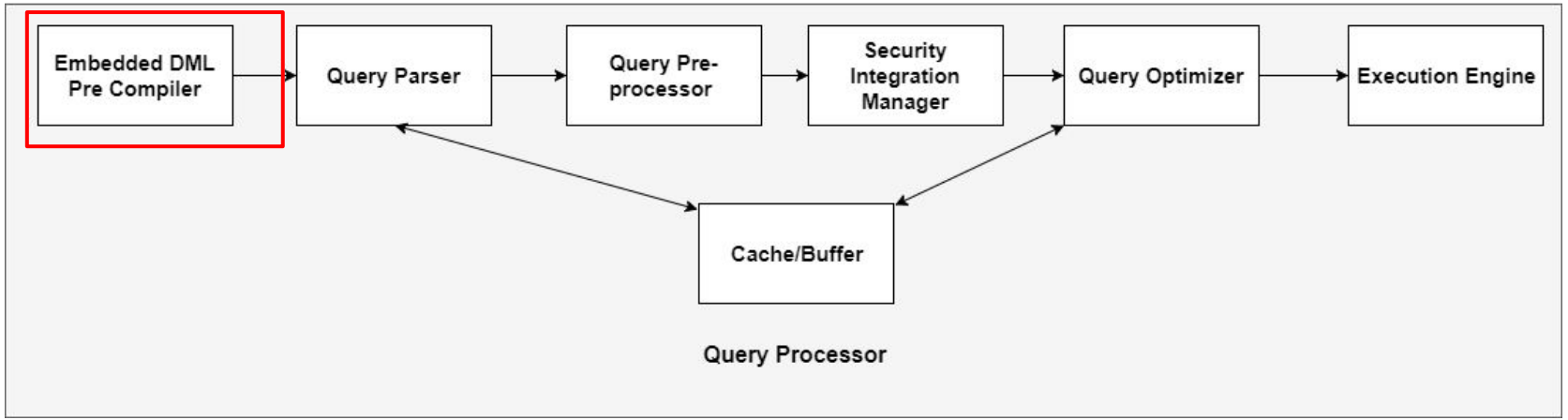
The DDL Compiler converts these statements into machine/object code for MySQL usage.

DDL Compiler



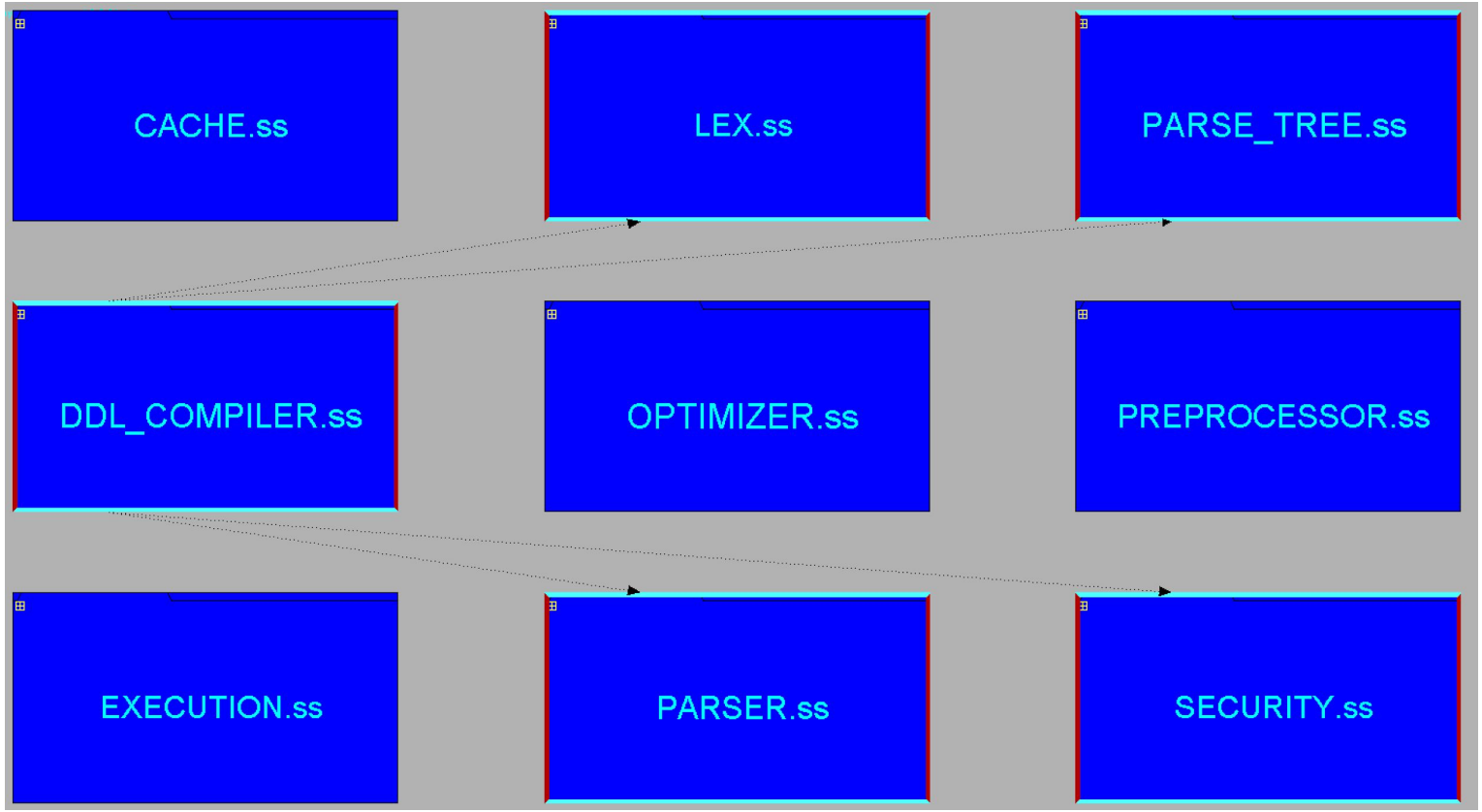


Concrete



Conceptual

DDL Compiler



LEX

PARSE_TREE

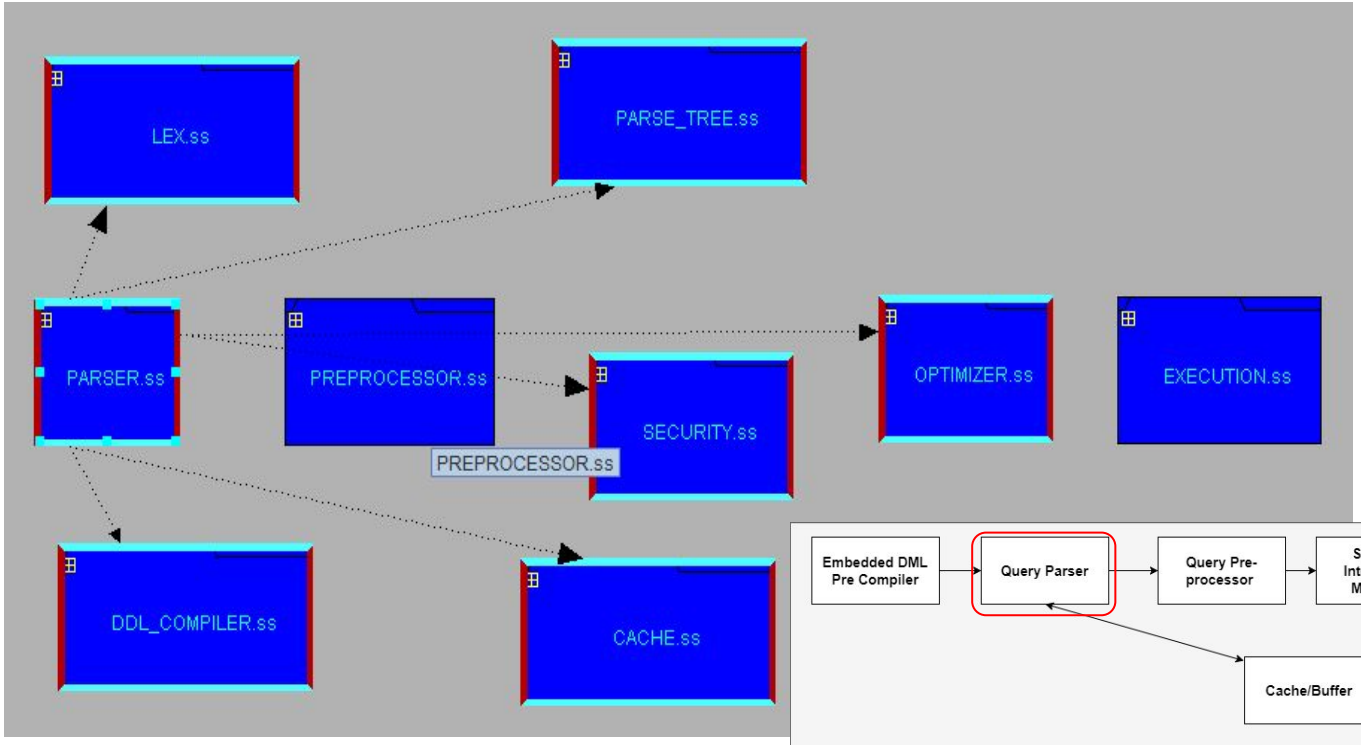
PARSER

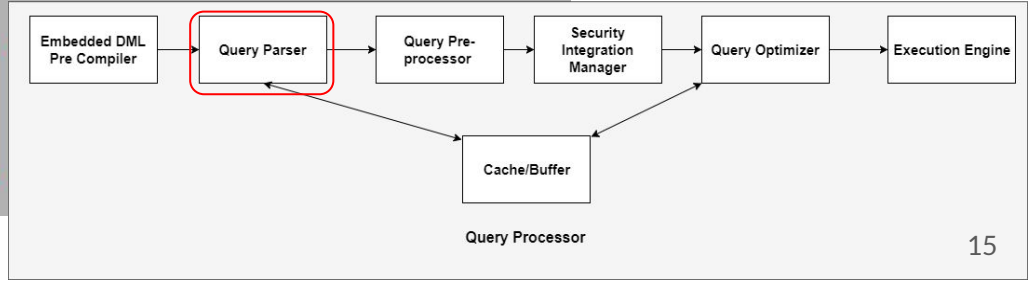
SECURITY

Query Parser

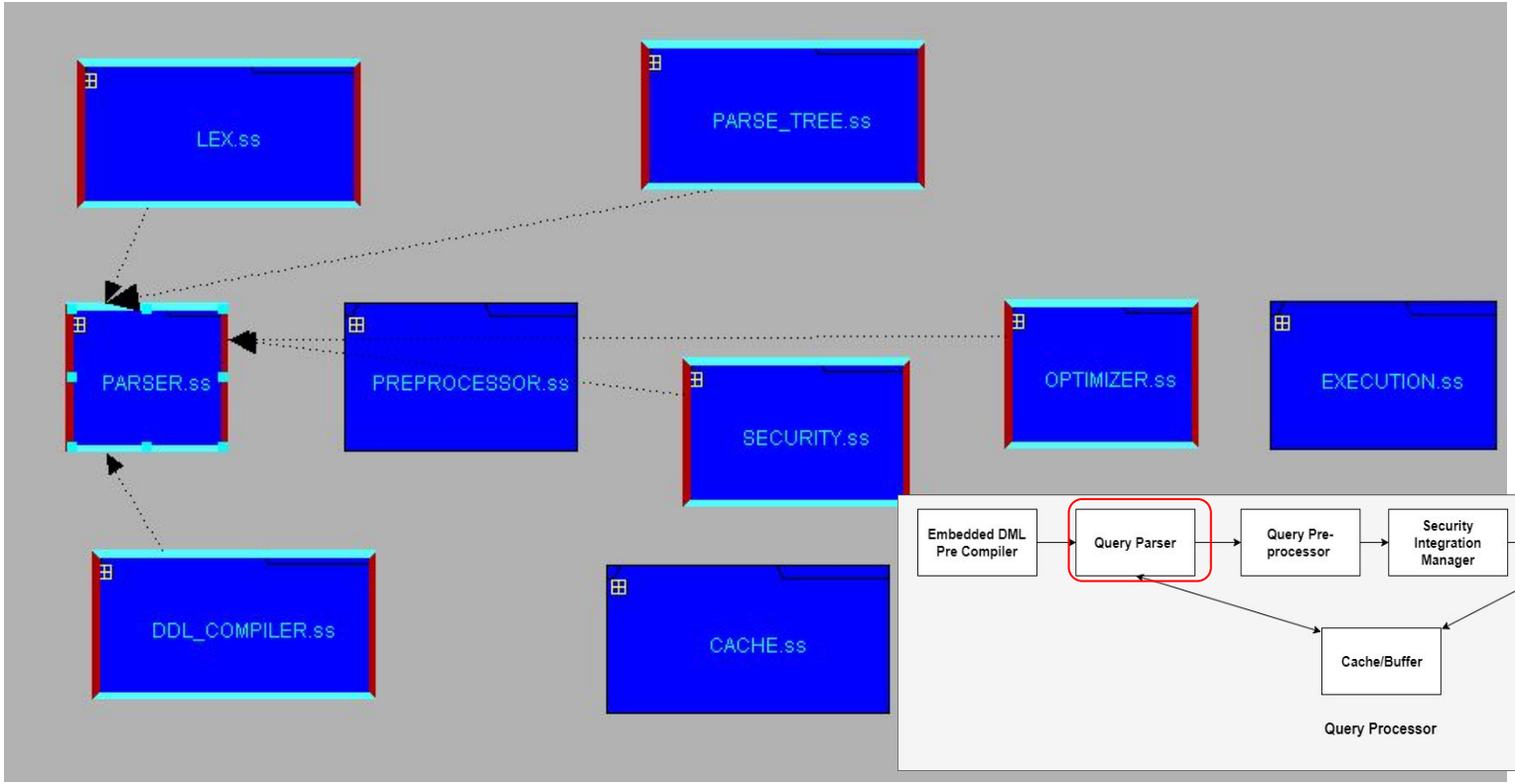
1) Interpreter and visitor design patterns.
2) Conceptual vs concrete architecture.
3) Parser to other subsystems (security, DD_compiler, Cache , Parse Tree, LEX, Optimizer).
4) Other Subsystems to Parser (Optimizer, Security, DD_compiler , Parse Tree, LEX).
5) New subsystems.

Parser to other systems dependencies





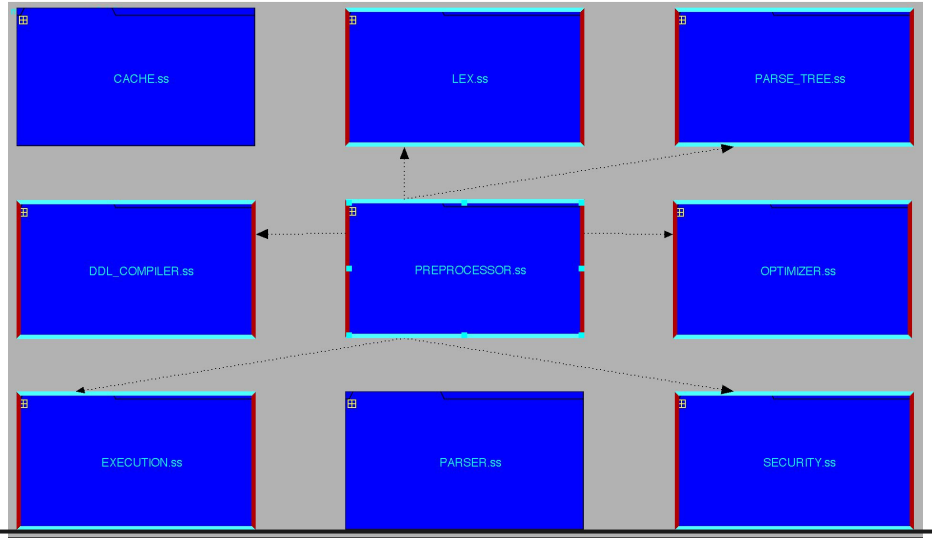
Other systems to Parser dependencies



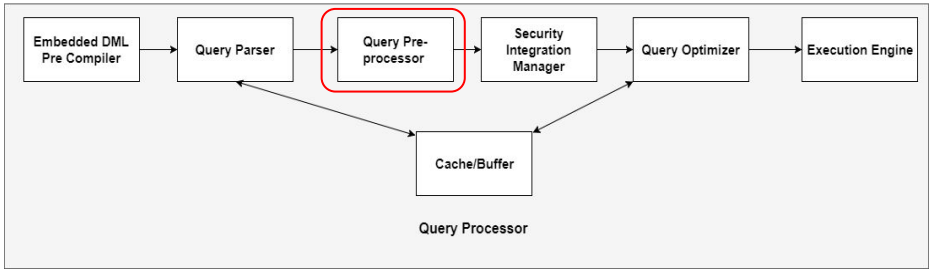
Query Preprocessor

- Performs semantic validation
  - Verify that the relations and views are in the database schema
  - Verify that attributes have a corresponding relation specified
  - Verify attribute types
- Parse tree is valid only if it is syntactically and semantically valid

Preprocessor depending on other systems

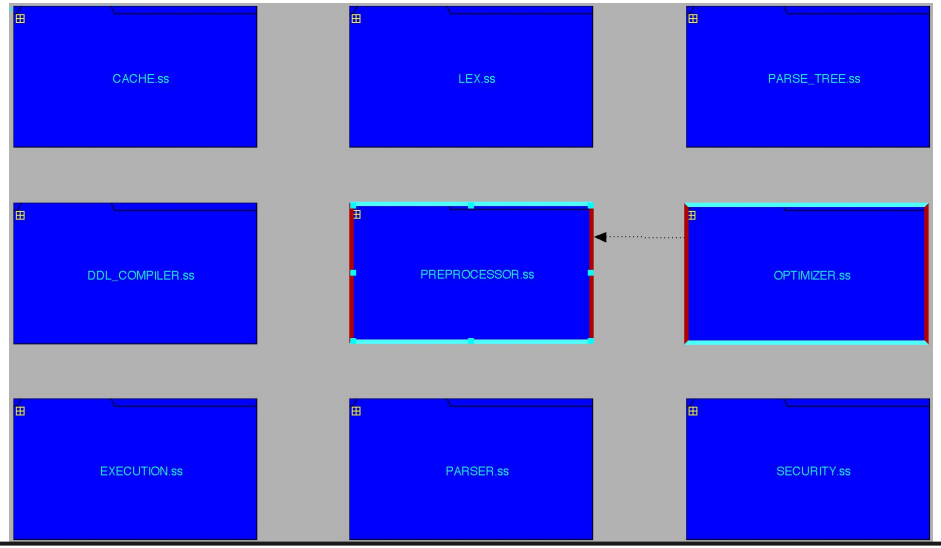


Concrete

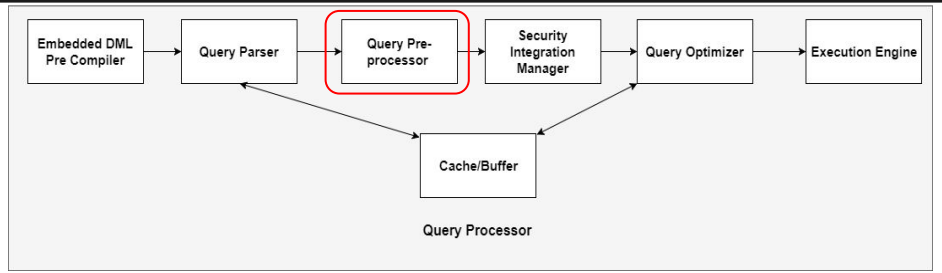


Conceptual

Other Systems depending on Preprocessor



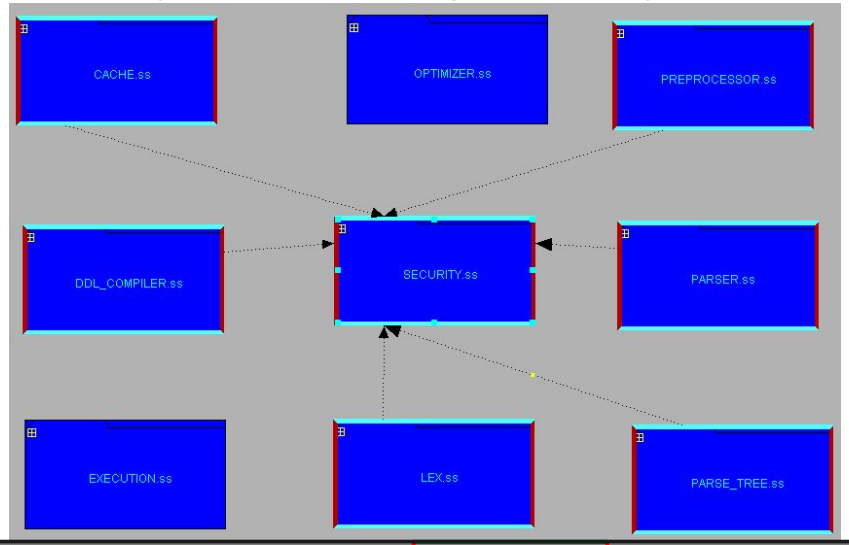
Concrete



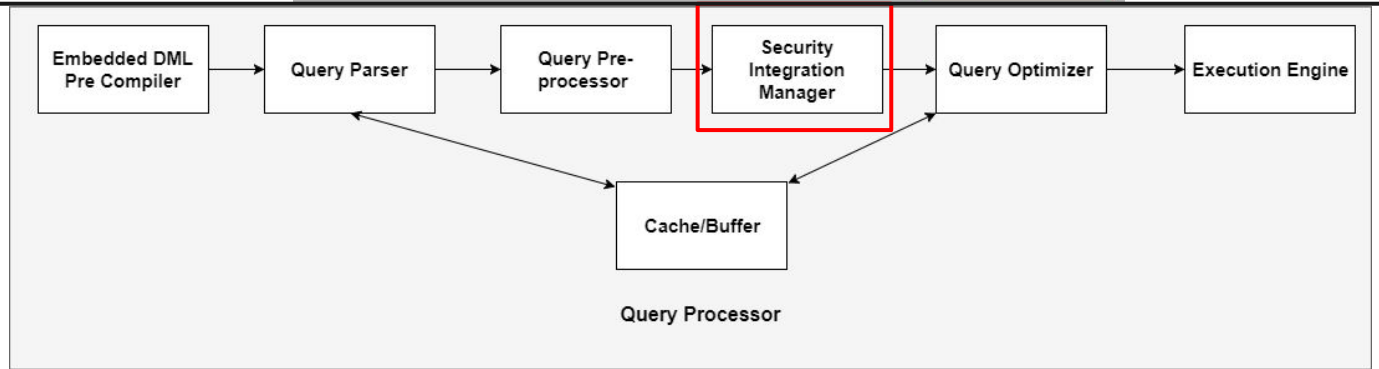
Conceptual

Security Integration Manager

Other Systems Depending on Security

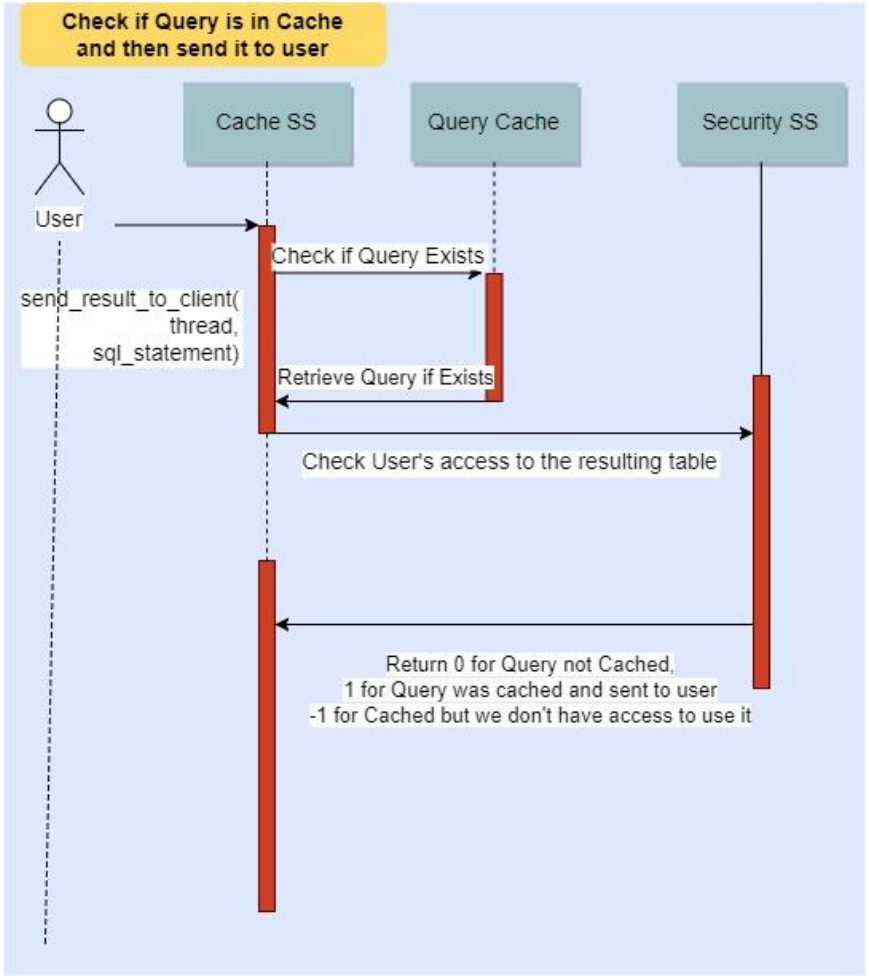


Concrete



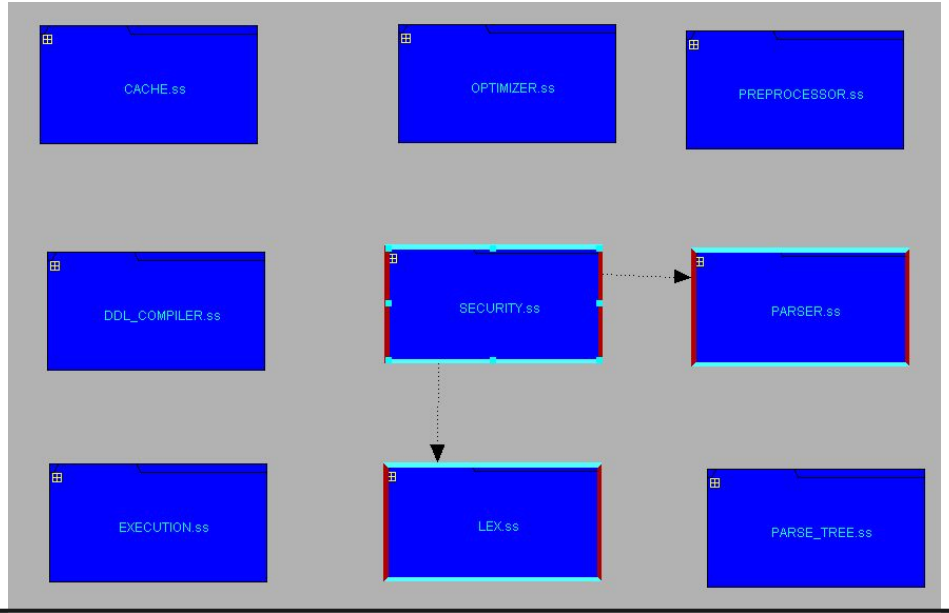
Conceptual

Use Case

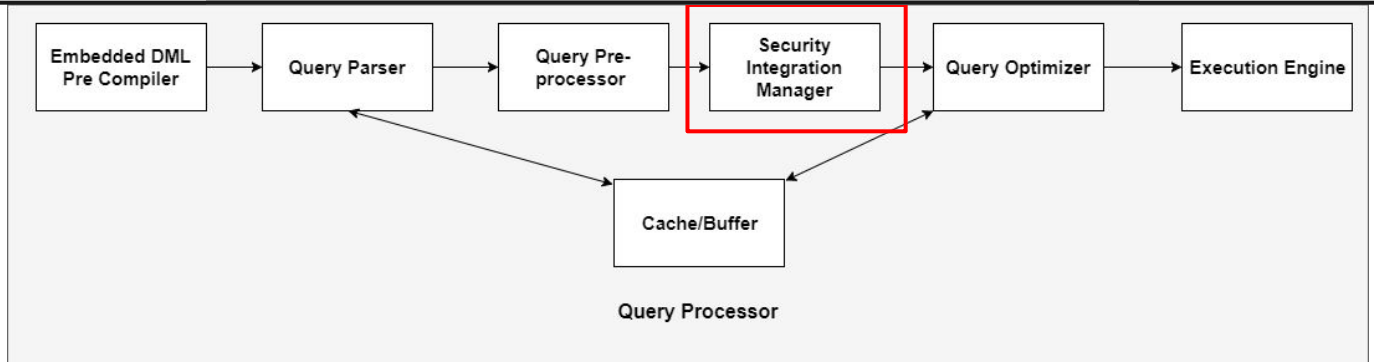


Security Integration Manager

Security depending on other systems



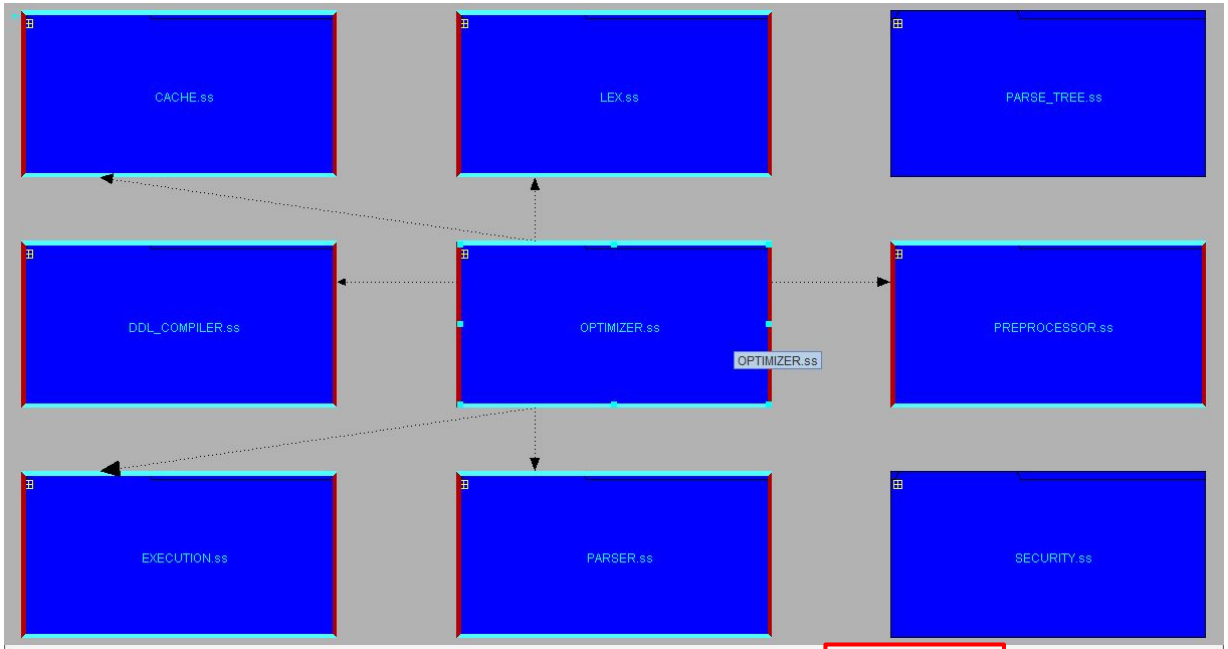
Concrete



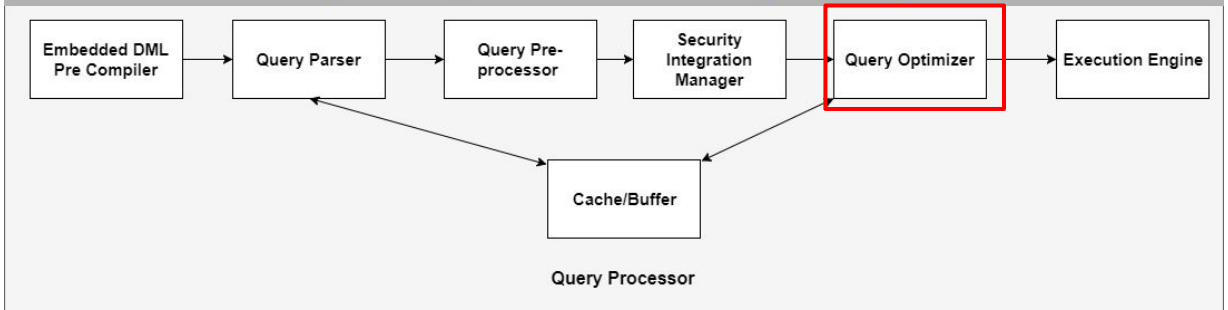
Conceptual

Optimizer to other systems dependencies

Concrete

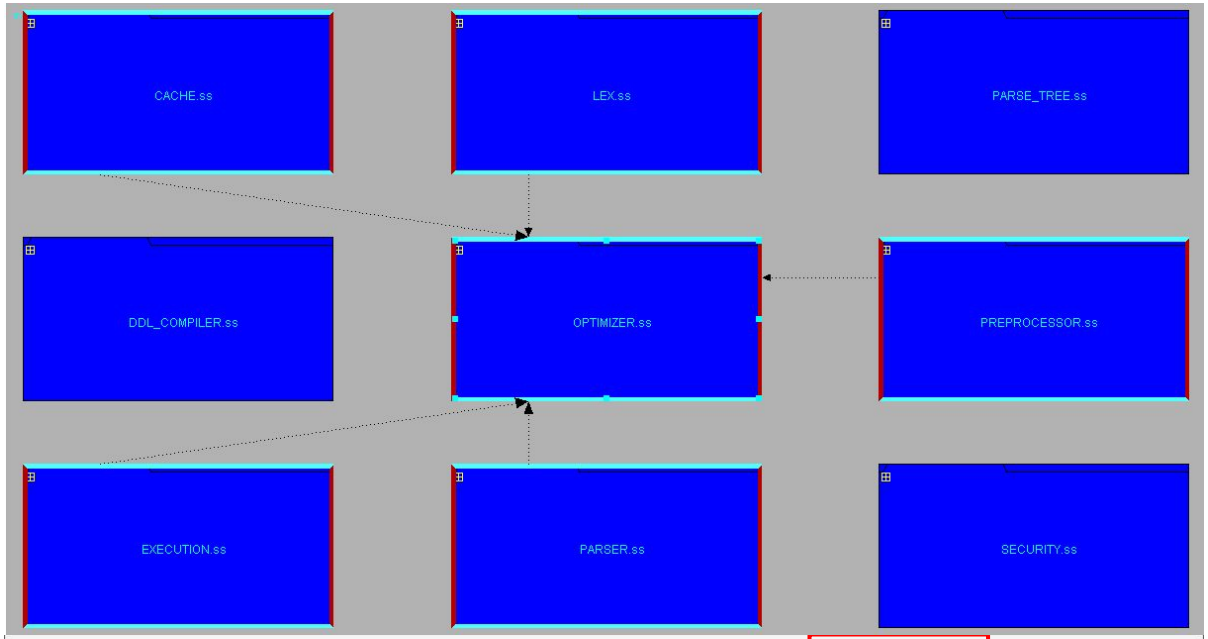


Conceptual

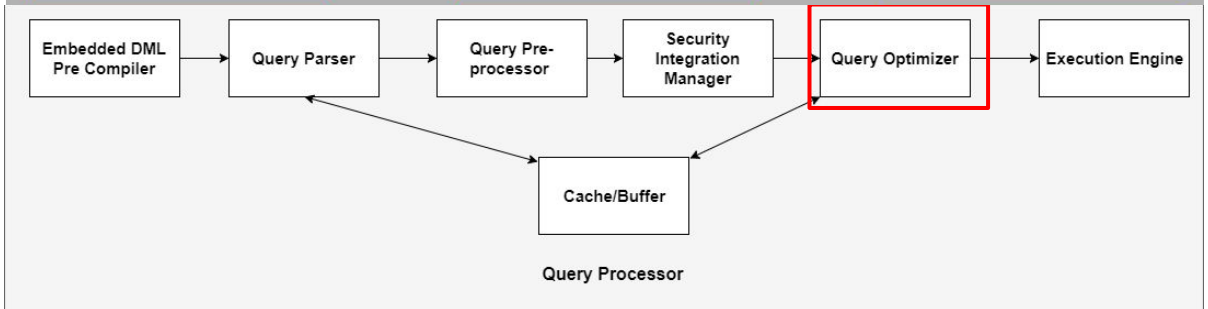


Other systems to Optimizer dependencies

Concrete



Conceptual



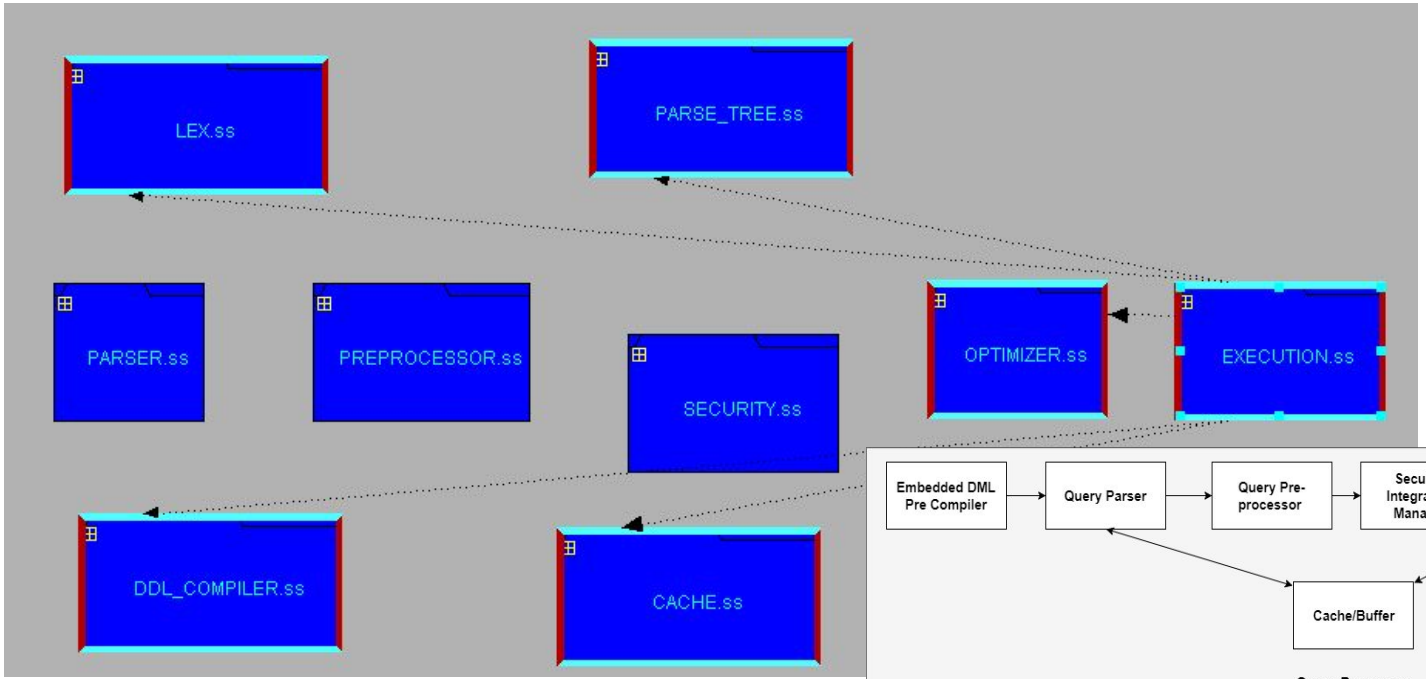
Execution Engine

1) Execution engine to other subsystems dependencies.

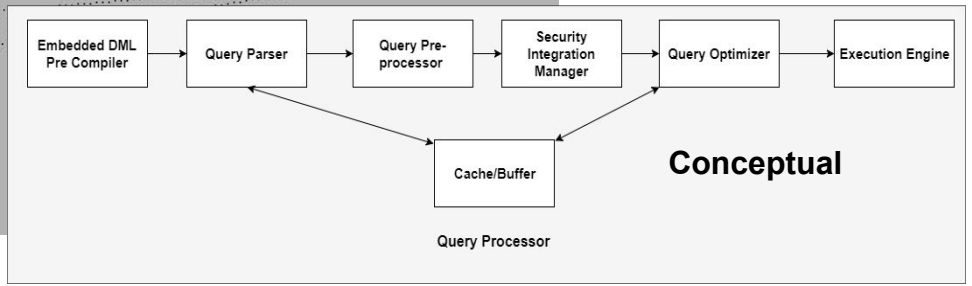
2) Other subsystems to execution engine dependencies.

3) New subsystems.

Execution to other subsystems dependenc

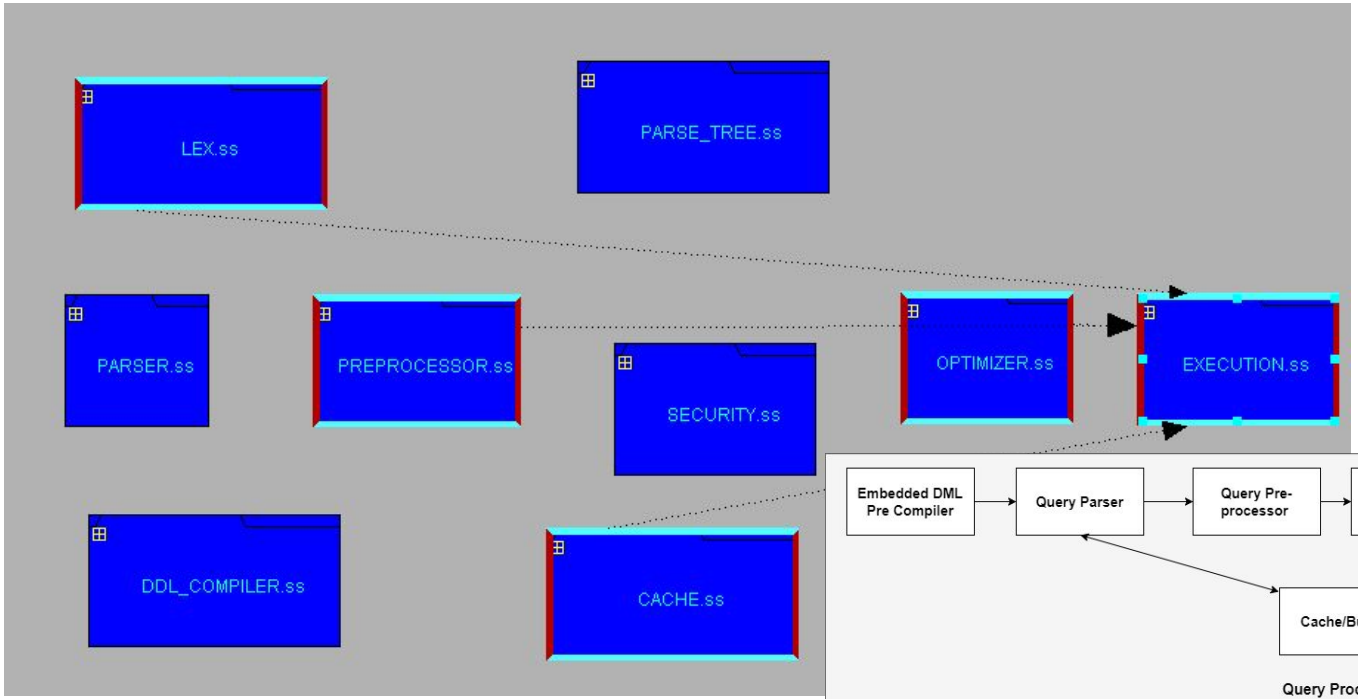


Concrete

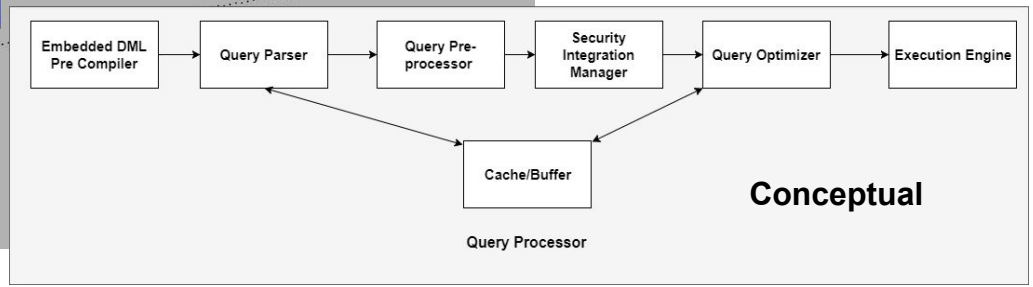


Conceptual

Other subsystems to execution dependenc



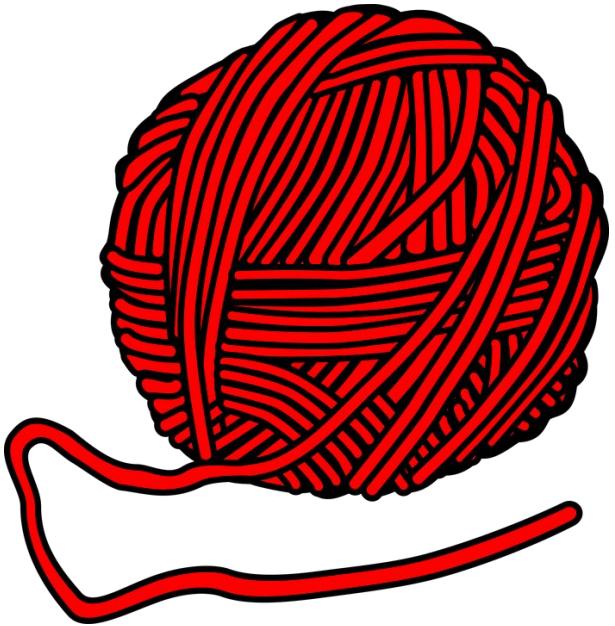
Concrete



Conceptual

Conclusions

-



Lessons Learned

- Lots of Work
- Iterative Process
- Automation
  - Scripts
  - Regex

It is difficult to derive concrete subsystems from looking at the source code and directory structure. Comments and documentation provide little help in understanding the concrete architecture.

Security Integration Manager is called many times throughout the query processing task by different subsystems, as opposed to just once.

Question Period