Query Execution from Interpretation to Compilation on RDBMS

Driven by Hardware Revolution

Ethan Zhu
jpzhu.nju@smail.nju.edu.cn

Department of Computer Science and Technology
Nanjing University

Oct, 12, 2020

Table of Contents

1. Reviews
2. Query Interpretation
3. Query Compilation
4. Conclusions
5. References
6. Acknowledgements and Questions

Contents

1. Reviews
2. Query Interpretation
3. Query Compilation
4. Conclusions
5. References
6. Acknowledgements and Questions

Table of Contents

1 Reviews

2 Query Interpretation

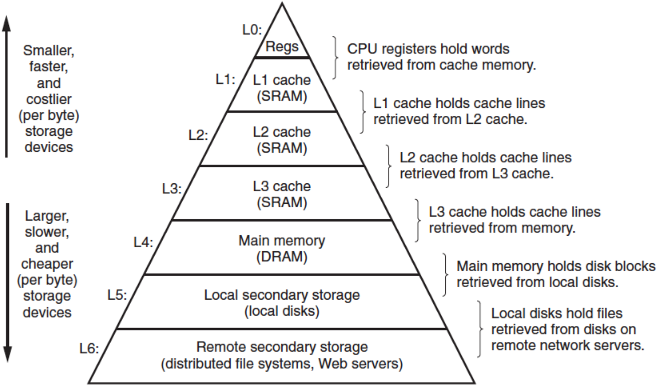
3 Query Compilation

4 Conclusions

5 References

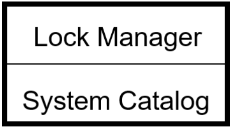
6 Acknowledgements and Questions

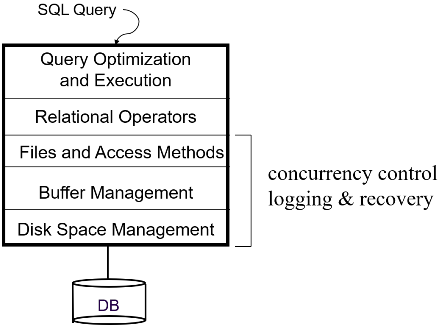
Storage hierarchy



Architecture of Relational Database Management Systems

- Disk space management
- Buffer pool management
- Files and access methods
- Relational operators
- Query optimization and execution





Architecture of Query Planning

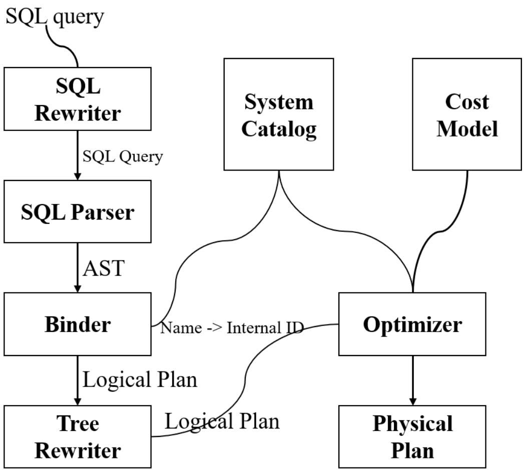


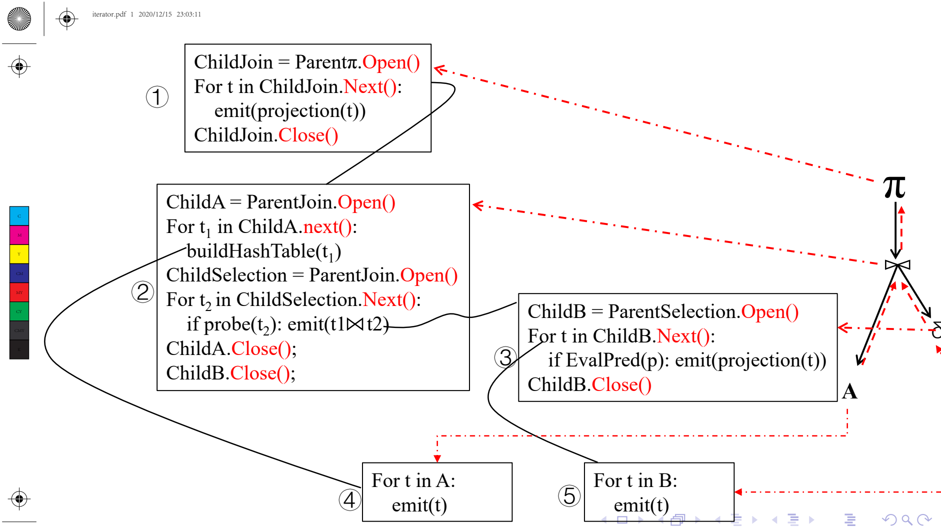
Table of Contents

1. Reviews
2. Query Interpretation
3. Query Compilation
4. Conclusions
5. References
6. Acknowledgements and Questions

SQL Query Example

```
SELECT A.id, B.value
FROM A, B
WHERE A.id = B.id AND B.value >= 100 * 3
```

Iterator/Volcano Processing Model



Characteristics of Iterator/Volcano Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.

Characteristics of Iterator/Volcano Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.
- Parallel - Each Operator can run on a different thread, and next is no longer a function call but inter-process communication, enabling pipelined parallelism between operators.

Characteristics of Iterator/Volcano Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.
- Parallel - Each Operator can run on a different thread, and next is no longer a function call but inter-process communication, enabling pipelined parallelism between operators.
- Pipeline - Every call to the next interface only needs to return one data, so it is not necessary to return all the data after calculation, so as to avoid the overhead of too much data materialization.

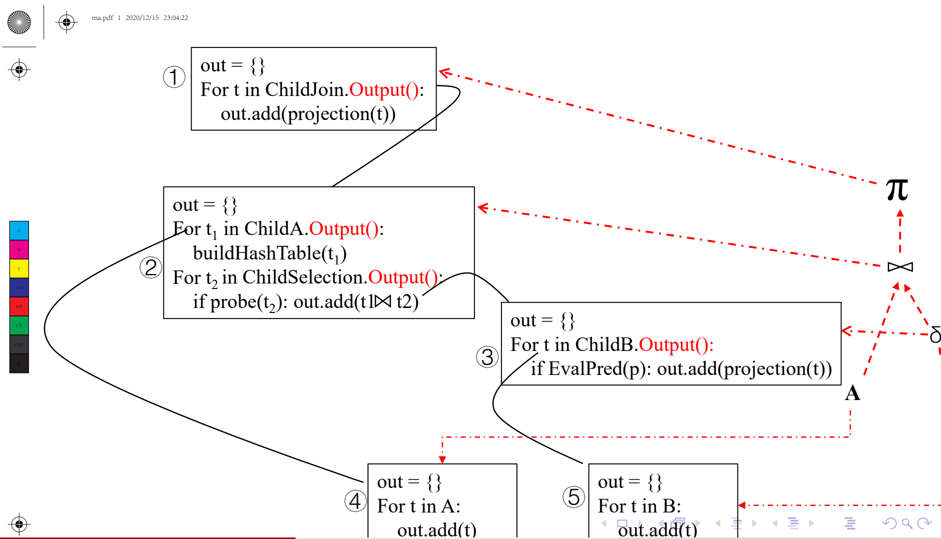
Characteristics of Iterator/Volcano Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.
- Parallel - Each Operator can run on a different thread, and next is no longer a function call but inter-process communication, enabling pipelined parallelism between operators.
- Pipeline - Every call to the next interface only needs to return one data, so it is not necessary to return all the data after calculation, so as to avoid the overhead of too much data materialization.
- Pull and Push - Although called a volcano, the data is not erupting from the bottom of the Plan Tree, but is pulled up from the top through an iterator.

Characteristics of Iterator/Volcano Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.
- Parallel - Each Operator can run on a different thread, and next is no longer a function call but inter-process communication, enabling pipelined parallelism between operators.
- Pipeline - Every call to the next interface only needs to return one data, so it is not necessary to return all the data after calculation, so as to avoid the overhead of too much data materialization.
- Pull and Push - Although called a volcano, the data is not erupting from the bottom of the Plan Tree, but is pulled up from the top through an iterator.
- Generous Purpose - It is suitable for OLAP/OLTP on disk-oriented RDBMS.

Materialization Processing Model



Characteristics of Materialization Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.

Characteristics of Materialization Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.
- Parallel - Each Operator can run on a different thread, which using producer-consumer model.

Characteristics of Materialization Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.
- Parallel - Each Operator can run on a different thread, which using producer-consumer model.
- Pipeline - Every call to the output interface needs to return one buffer data.

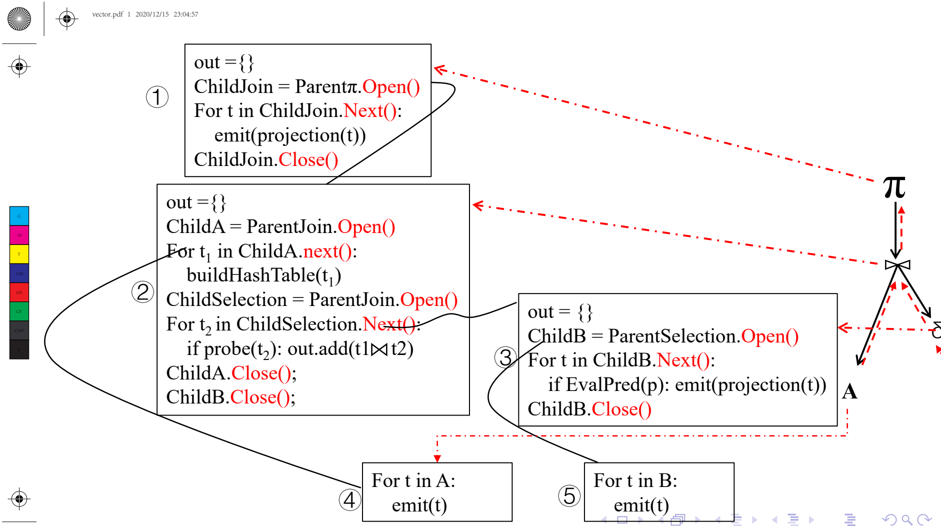
Characteristics of Materialization Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.
- Parallel - Each Operator can run on a different thread, which using producer-consumer model.
- Pipeline - Every call to the output interface needs to return one buffer data.
- Push - The data is pushed from the bottom of the Plan Tree.

Characteristics of Materialization Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.
- Parallel - Each Operator can run on a different thread, which using producer-consumer model.
- Pipeline - Every call to the output interface needs to return one buffer data.
- Push - The data is pushed from the bottom of the Plan Tree.
- OLTP - It is suitable for OLTP on disk-oriented RDBMS.

Vectorization Processing Model



Characteristics of Vectorization Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.

Characteristics of Vectorization Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.
- Parallel - Each Operator can run on a different thread, which using producer-consumer model.

Characteristics of Vectorization Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.
- Parallel - Each Operator can run on a different thread, which using producer-consumer model.
- Pipeline - Every call to the next interface needs to return one buffer data.

Characteristics of Vectorization Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.
- Parallel - Each Operator can run on a different thread, which using producer-consumer model.
- Pipeline - Every call to the next interface needs to return one buffer data.
- Pull and Push - The data is pushed from the bottom of the Plan Tree by control flow from top to bottom.

Characteristics of Vectorization Processing Model

- Extensible - Operators that relies on interfaces rather than concrete implementations, it's easy to extend a new Operator to replace an existing one.
- Parallel - Each Operator can run on a different thread, which using producer-consumer model.
- Pipeline - Every call to the next interface needs to return one buffer data.
- Pull and Push - The data is pushed from the bottom of the Plan Tree by control flow from top to bottom.
- OLAP - It is suitable for OLTP on disk-oriented RDBMS.

What's the problem?

- Iterator Overhead - Iterator is generally implemented using a virtual function, which requires multiple function calls per line of data read, incurring some overhead.

What's the problem?

- Iterator Overhead - Iterator is generally implemented using a virtual function, which requires multiple function calls per line of data read, incurring some overhead.
- Cache Inefficiency - Use a pull and push model, after the data is loaded into memory and cannot stay in the cache and the register, need more memory access.

What's the problem?

- Iterator Overhead - Iterator is generally implemented using a virtual function, which requires multiple function calls per line of data read, incurring some overhead.
- Cache Inefficiency - Use a pull and push model, after the data is loaded into memory and cannot stay in the cache and the register, need more memory access.
- CPU Inefficiency - Iterator model of a large number of branch statements, memory access is not friendly to the CPU.

Attempt

Based on these issues, industry began to explore the path of query compilation, but it was not until recent years that it began to get more practical application.

Industrial products such as Impala, SparkSQL, PostgreSQL also began to adopt the scheme of compilation execution.

However, there is still a lot of imagination space for query compilation. And there is still a gap between the product landing in the industry and the academic research.

Table of Contents

1. Reviews
2. Query Interpretation
3. Query Compilation
4. Conclusions
5. References
6. Acknowledgements and Questions

Questions

- What can be compiled in a query?

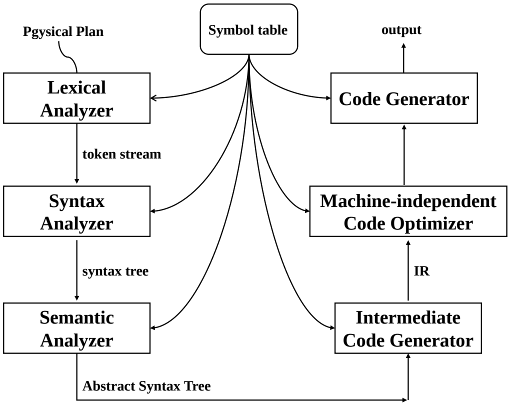
Questions

- What can be compiled in a query?
- How to compile?

Questions

- What can be compiled in a query?
- How to compile?
- What is the result of compiling?

Compiler Overview



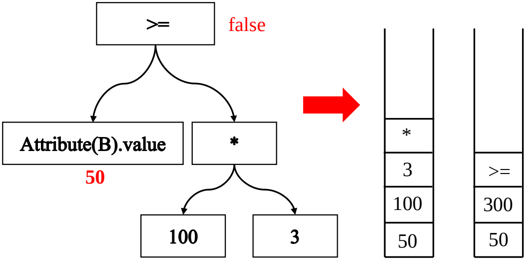
Examples of Expression Interpretation

Execution Context

SELECT * from B
WHERE B.value >= 100 * 3

current tuple
(123,50)

schema
(id, value)



Disadvantages of Expression Interpretation

- The implementation of interpreting execution often recursively interprets each Operator, each Operand, resulting in multiple function calls.

Disadvantages of Expression Interpretation

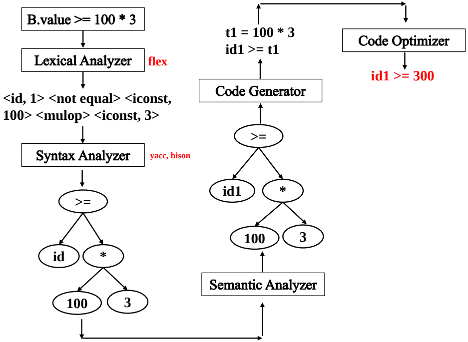
- The implementation of interpreting execution often recursively interprets each Operator, each Operand, resulting in multiple function calls.
- Query interpreter needs to interpret once for each tuple, which is inefficient.

Disadvantages of Expression Interpretation

- The implementation of interpreting execution often recursively interprets each operator, each operand, resulting in multiple function calls.
- Query interpreter needs to interpret once for each tuple, which is inefficient.

How to solve these issues?

Examples of Expression Compilation



Example of Attributes of Tuples Compilation

```
void MaterializeTuple(char* tuple) {
  for (int i = 0; i < num_slots_; ++i)
{
    char* slot = tuple + offsets_[i];
    switch(types_[i]) {
      case BOOLEAN:
        *slot = ParseBoolean();
        break;
      case INT:
        *slot = ParseInt();
        break;
      case FLOAT: …
      case STRING: …
      // etc.
    }
  }
}
```

interpreted

```
void MaterializeTuple(char* tuple) {
  *(tuple + 0) = ParseInt();    // i = 0
  *(tuple + 4) = ParseBoolean();// i = 1
  *(tuple + 5) = ParseInt();    // i = 2
}
```

codegen'd

Example of Operators Compilation

Interpretation

```
for t in range(table.num_tuples):
    tuple = get_tuple(table, t)
    if eval(predicate, tuple, parameters):
        emit(tuple)
```

Compilation

```
tuple_size = xxx
predicate.offset = xxx
parameters_value = xxx
for t in range(table.num_tuples):
    tuple = table.data  + t * table.size
    val = (tuple+predicate_offset) + 1
    if (val == parameter_value):
        emit(tuple)
```

- Get schema in the system catalog for table.
- Calculate the offset based on the tuple size.
- Return the pointer to the tuple.

Table of Contents

1. Reviews
2. Query Interpretation
3. Query Compilation
4. Conclusions
5. References
6. Acknowledgements and Questions

Conclusions

- As hardware (disk, memory, etc) speeds up, so does software change.

Conclusions

- As hardware (disk, memory, etc) speeds up, so does software change.
- Compilers generate machine code, whereas interpreters interpret intermediate code.

Conclusions

- As hardware (disk, memory, etc) speeds up, so does software change.
- Compilers generate machine code, whereas interpreters interpret intermediate code.
- Interpreters are easier to write and can provide better error messages (symbol table is still available).

Conclusions

- As hardware (disk, memory, etc) speeds up, so does software change.
- Compilers generate machine code, whereas interpreters interpret intermediate code.
- Interpreters are easier to write and can provide better error messages (symbol table is still available).
- Interpreters are at least 5 times slower than machine code generated by compilers.

Conclusions

- As hardware (disk, memory, etc) speeds up, so does software change.
- Compilers generate machine code, whereas interpreters interpret intermediate code.
- Interpreters are easier to write and can provide better error messages (symbol table is still available).
- Interpreters are at least 5 times slower than machine code generated by compilers.
- Interpreters also require much more memory than machine code generated by compilers.

Table of Contents

1 Reviews

2 Query Interpretation

3 Query Compilation

4 Conclusions

5 References

6 Acknowledgements and Questions

References I

- Tahboub, R.Y. and Rompf, T., 2020, June. Architecting a Query Compiler for Spatial Workloads. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (pp. 2103-2118).
- Kersten, T., Leis, V., Kemper, A., Neumann, T., Pavlo, A. and Boncz, P., 2018. Everything you always wanted to know about compiled and vectorized queries but were afraid to ask. Proceedings of the VLDB Endowment, 11(13), pp.2209-2222.
- Tahboub, R.Y., Essertel, G.M. and Rompf, T., 2018, May. How to architect a query compiler, revisited. In Proceedings of the 2018 International Conference on Management of Data (pp. 307-322).

References II

- Menon, P., Mowry, T.C. and Pavlo, A., 2017. Relaxed operator fusion for in-memory databases: Making compilation, vectorization, and prefetching work together at last. Proceedings of the VLDB Endowment, 11(1), pp.1-13.
- Shaikhha, A., Klonatos, Y., Parreaux, L., Brown, L., Dashti, M. and Koch, C., 2016, June. How to architect a query compiler. In Proceedings of the 2016 International Conference on Management of Data (pp. 1907-1922).
- Lang, H., Mühlbauer, T., Funke, F., Boncz, P.A., Neumann, T. and Kemper, A., 2016, June. Data blocks: Hybrid OLTP and OLAP on compressed storage using both vectorization and compilation. In Proceedings of the 2016 International Conference on Management of Data (pp. 311-326).

References III

- Nagel, F., Bierman, G. and Viglas, S.D., 2014. Code generation for efficient query processing in managed runtimes. Proceedings of the VLDB Endowment, 7(12), pp.1095-1106.
- Wanderman-Milne, S. and Li, N., 2014. Runtime Code Generation in Cloudera Impala. IEEE Data Eng. Bull., 37(1), pp.31-37.

Table of Contents

1. Reviews
2. Query Interpretation
3. Query Compilation
4. Conclusions
5. References
6. Acknowledgements and Questions

Thank you!
Welcome for any questions!

Ethan Zhu
Department of Computer Science and Technology
Nanjing University