Physical Operators and the Volcano Model

> Designed by Zhu Yueming, 2025
> Code Framework Contributor: Zhang Ziyang

In the **basic scenarios** of database systems—such as last week's exercises—users can retrieve all table data using simple commands (e.g., `SELECT * FROM t;`) or extract specific results through filtering conditions (e.g., `SELECT name, age FROM t WHERE gpa > 3.0;`). These operations appear straightforward but face the following implementation challenges:

1. **Generalization of functionality**:
   The database must support **arbitrary complex query compositions** (e.g., multi-layer filtering, joins, aggregation, sorting). A single code logic cannot cover all scenarios.
2. **Execution efficiency and resource management**:
   Hardcoding all functionality results in bloated code, uncontrollable memory usage (e.g., full table loads), and difficulty in optimizing localized logic.
3. **Modularity and extensibility**:
   Adding new command types (such as Count, Delete) should avoid invasive modifications to the core engine.

To address these problems, the Volcano Model proposes the idea of **physical operator abstraction**. We will introduce the concepts of physical operators and the Volcano Model step by step.

I. Physical Operators

In the database execution engine, **composite operations** follow the **single responsibility principle**: each independent operation is abstracted into a **physical operator**, each handling a specific task (such as scanning, filtering, projecting), and returning processed tuples to upper layers via a unified interface.

Example:

```sql
SELECT name, age FROM t WHERE gpa > 3.0;
```

This query can be decomposed into three cooperating physical operators:

1. **ScanOperator**
   - Role: Reads raw data row-by-row from table `t` in the storage engine.
2. **FilterOperator**
   - Role: Applies the predicate `gpa > 3.0` to filter qualified tuples.
3. **ProjectOperator**
   - Role: Trims fields, retaining only `name` and `age` columns.

Standardized Interfaces for Physical Operators

To enable **composability**, all physical operators must follow a unified iterator interface. In Java, the core methods are defined as follows:

```java
public interface PhysicalOperator {
    void Begin() throws DBException;            // Initialize (e.g., open file, initialize child operators)
    boolean hasNext() throws DBException;       // Check if more data exists
    void Next() throws DBException;             // Process next data item
    Tuple Current();                            // Retrieve current tuple
    void Close();                               // Clean up resources (e.g., close file)
    ArrayList<ColumnMeta> outputSchema();       // Define output metadata (column names, types)
}
```

Key data structures to emphasize:

1. **Tuple**:
   The core data unit passed between physical operators. Essentially an abstract representation of a data row and acts as a bridge between binary data and programming language types. DBMSs like MySQL, Vastbase, PostgreSQL, etc., use `Tuple` to pass data upward in their Volcano-style execution.
2. **ColumnMeta**:
   Records the set of columns to return. If projecting the entire table, it returns all columns; if only some, it must return the corresponding subset.

II. Volcano Model

The Volcano Model is a classic architecture in query execution, also known as the **Iterator Model**. It processes data tuple-by-tuple, decomposing SQL statements into physical operators and arranging them into a top-down layered structure (e.g., Projection → Filter → Join → Scan). Upper-level operators invoke lower-level ones via `next()` or `hasNext()` and receive data via `current()`.

Example SQL:

```sql
SELECT t.age, t.name FROM t WHERE t.gpa > 3.0;
```

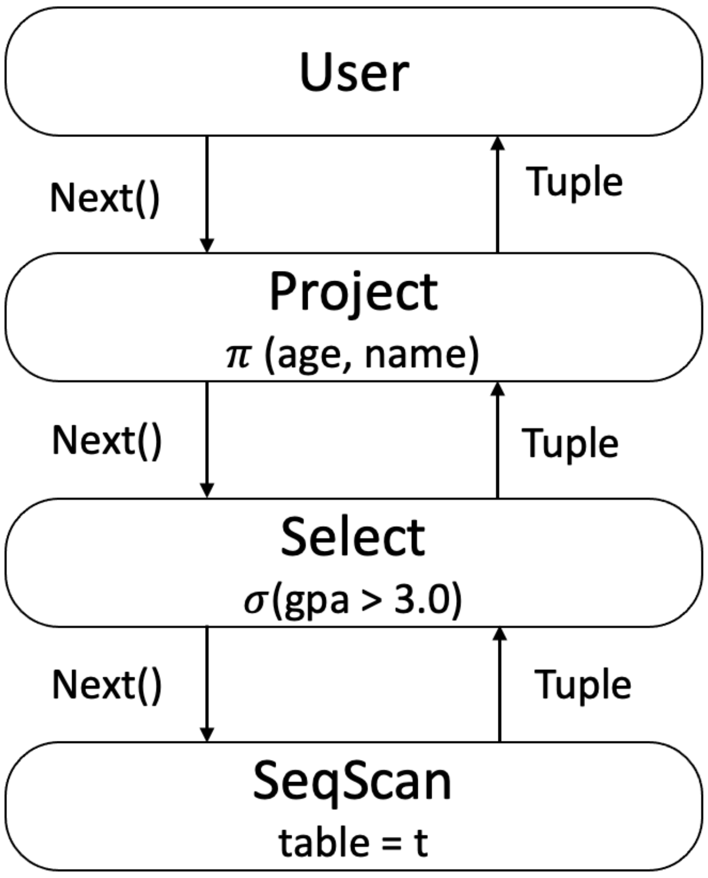
Logical operator tree:

```
ProjectOperator(selectItems=[t.age, t.name])
└── LogicalFilterOperator(condition=t.gpa > 3.0)
        └── TableScanOperator(table=t)
```

Physical operator plan:

```
ProjectOperator(selectItems=[t.age, t.name])
└── FilterOperator(condition=t.gpa > 3.0)
        └── SeqScanOperator(table=t)
```

Execution relationship in the Volcano Model (illustrated below):



Exercise

For the `DELETE` statement, create relevant physical operators and build the Volcano Model accordingly. Implement it for two scenarios:

- `DELETE FROM t WHERE t.id = 3;`
- `DELETE FROM t;`

Design a general Volcano Model that supports both modes and implement it.