Design Decisions:

It uses the join predicate and filter predicate to compare the tuples that meet the requirement for filter and nested loop join. For hashEqui Join, I added the gethash function to all of the fields as a virtual function.

1. Aggregate.:
    a. Design Choices:
        i. Aggregator Selection: Depending on the type of the aggregation field, it creates either an `IntegerAggregator` or `StringAggregator`.
        ii. Grouping Logic: Checks if grouping is required (based on `gfield`) and processes tuples accordingly.
        iii. Tuple Processing: Tuples are processed through the `Aggregator::mergeTupleIntoGroup` method, integrating each tuple into the aggregate computation.
        iv. Result Formatting: The result is formatted as a tuple, either as a pair (groupValue, aggregateValue) or a single value depending on grouping.
    b. Challenges & Solutions: Handling different data types (integer and string) and operations while maintaining a consistent interface for aggregation.
2. IntegerAggregator.cpp:
    a. Design Choices:
        i. Aggregate Functions: Implements different aggregation functions like MIN, MAX, SUM, AVG, and COUNT using a function pointer (`aggFunc`).
        ii. Data Storage: Uses a map to store the aggregated results and a count map for AVG calculation.
        iii. Iterator Implementation: An inner class `IntegerAggregatorIterator` is used for iterating over the results.
    b. Challenges & Solutions: Devising a flexible method to handle various aggregation operations, especially for AVG which requires tracking the count.
3. Insert.cpp:
    a. Design Choices:
        i. BufferPool Interaction: Uses `BufferPool::insertTuple` for inserting tuples.
        ii. Result Reporting: Returns a count of the number of tuples inserted.
    b. Challenges & Solutions: Ensuring that the insert operation is reflected in the database and accurately reporting the number of inserted tuples.
4. Delete.cpp:

a. Design Choices:
    i. BufferPool Interaction: Utilizes `BufferPool::deleteTuple` for deleting tuples.
    ii. Result Reporting: Similar to Insert, it returns a count of the number of tuples deleted.
b. Challenges & Solutions: Ensuring that the delete operation is effective and that the count of deleted tuples is accurate.

Missing or incomplete elements:
For now, we have finished everything.

how long you spent on the assignment:
We spent about 15 hours on this assignment.

Collaboration:
Sijin Chen & Song Xu collaborated on this assignment. Sijin Chen did the implementation for Filter, Join, JoinPredicate, and HashEquiJoin. Song implemented Aggregate, Insert, and Deletion.