Design Decisions:

In my design, I implement a join operation between two database iterators using a join predicate. My constructor initializes with two child iterators and a join predicate, computing tuple sizes and creating a merged tuple descriptor. I provide methods to access join predicate details and manage the join operation's state, including opening, closing, and rewinding iterators. The core of my design is the fetchNext method, where I implement a nested loop join algorithm to iterate over tuples, apply the join predicate, and merge matching tuples. I maintain state for efficient processing and adhere to database operation standards.

In my design, I develop an integer aggregation framework for a database system. It involves managing aggregate data and iterating over aggregates with specific operations like count, min, max, average, and sum.

IntAggregateData: This class tracks aggregate statistics (sum, min, max, count) for integers, with methods to access and modify these statistics.

IntegerAggregatorIterator: As a subclass of DbIterator, this class iterates over aggregated data. It uses a hash map (aggMap) to store aggregate data keyed by integers, and processes aggregates based on the specified operation (like count, min, max, average, sum).

IntegerAggregator: This class aggregates integer fields from tuples. It supports grouping by a field and various aggregation operations. The mergeTupleIntoGroup method updates aggregate statistics for each group or the entire dataset (if no grouping is specified).

Iterator: The iterator method creates an IntegerAggregatorIterator instance to iterate over the aggregated data.

My approach ensures efficient aggregation and iteration over integer data in a database, supporting essential aggregation operations and optional grouping.

API Modifications:

Upon thorough evaluation of the provided Application Programming Interface (API), we determined that no modifications or adjustments were essential for our implementation's objectives. Thus, the API remains unaltered.

Code Completeness:

As per the stipulated requirements for PA3, we have successfully realized and executed all essential components, ensuring a holistic solution.

Implementation Duration and Challenges:

The entire duration of PA3's implementation spanned approximately one full week. Throughout this period, we diligently committed changes to our GitHub repository over 30 times, ensuring version control best practices and tracking our progress.

The primary challenge encountered involves the process of debugging and addressing unexpected null pointer occurrences. We spent a lot of time figuring out how to do equijoin and pass the test given.

Collaboration Details:

Our team employed a collaborative approach to tackle the project. Specifically:

Yanpeng Zhao spearheaded the tasks encompassed in EX3.1, 3.2.
Pengfei Li undertook the challenges presented in EX3.2, 3.3
While each member primarily focused on their designated segments, our collaboration wasn't confined strictly to these demarcations. We embraced collective brainstorming, debugging, and testing, ensuring that the final output was a product of combined expertise and team synergy.