КУРСОВАЯ РАБОТА
Программный проект
на тему

Модификаторы DISTINCT и ORDER BY для всех агрегатных функций

Выполнил студент группы 175, 3 курса,
Борзенкова София Сергеевна

Руководитель КР:
Руководитель группы разработки,
Миловидов Алексей Николаевич

ClickHouse

СУБД[1] от компании «Яндекс»

- Написана на языке C++
- Имеет открытый исходный код[2]
- Обрабатывает запросы на языке SQL[3]
- Поддерживает не все возможности языка SQL

[1] СУБД – система управления базами данных
[2] https://github.com/ClickHouse/ClickHouse
[3] SQL – structured query language

Актуальность работы

- ClickHouse не поддерживает запросы, содержащие агрегатные функции с модификаторами DISTINCT и ORDER BY
- Некоторым пользователям СУБД необходимо наличие модификаторов DISTINCT и ORDER BY для выполнения своих задач, вследствие чего они выбирают другие СУБД
- Реализация этой функциональности в других СУБД не применима к ClickHouse из-за особенностей архитектуры

```
SELECT COUNT(UserID)
FROM datasets.visits_v1

┌─COUNT(UserID)─┐
│       1676861 │
└───────────────┘

1 rows in set. Elapsed: 0.029 sec.
```

```
SELECT sumArray(URLCategories)
FROM datasets.visits_v1

┌─sumArray(URLCategories)─┐
│              39509214026 │
└──────────────────────────┘

1 rows in set. Elapsed: 0.028 sec.
```

Цель и задачи

Цель работы — в архитектуре ClickHouse реализовать поддержку запросов, содержащих агрегатные функции с модификатором DISTINCT или ORDER BY.

Задачи:

- Изучить архитектуру ClickHouse, относящуюся к агрегатным функциям
- Реализовать модификаторы DISTINCT и ORDER BY как отдельные комбинаторы агрегатных функций и подключить их к фабрике комбинаторов
- Провести тестирование реализованной функциональности
- Добавить функциональные тесты для написанного кода

Существующие решения

- **Комбинаторы в ClickHouse:**
  - Подходят по архитектуре — являются комбинаторами агрегатных функций
  - Выполняют другую функциональность
- **Модификаторы в PostgreSQL и MySQL:**
  - Выполняются внутри агрегатной функции, если передан флаг о необходимости выполнения — не подходят по архитектуре
- **Модификаторы в Apache Druid:**
  - Схожи по архитектуре
  - Модификатор DISTINCT реализован только для агрегатной функции COUNT и исполняется внутри нее
  - Модификатор ORDER BY отсутствует

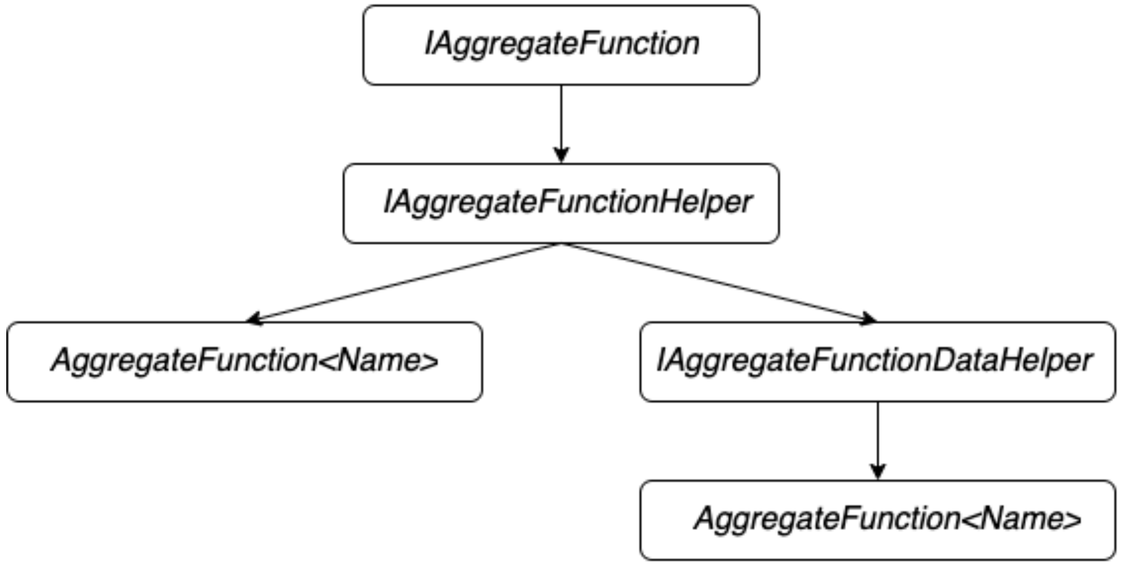
Архитектура ClickHouse

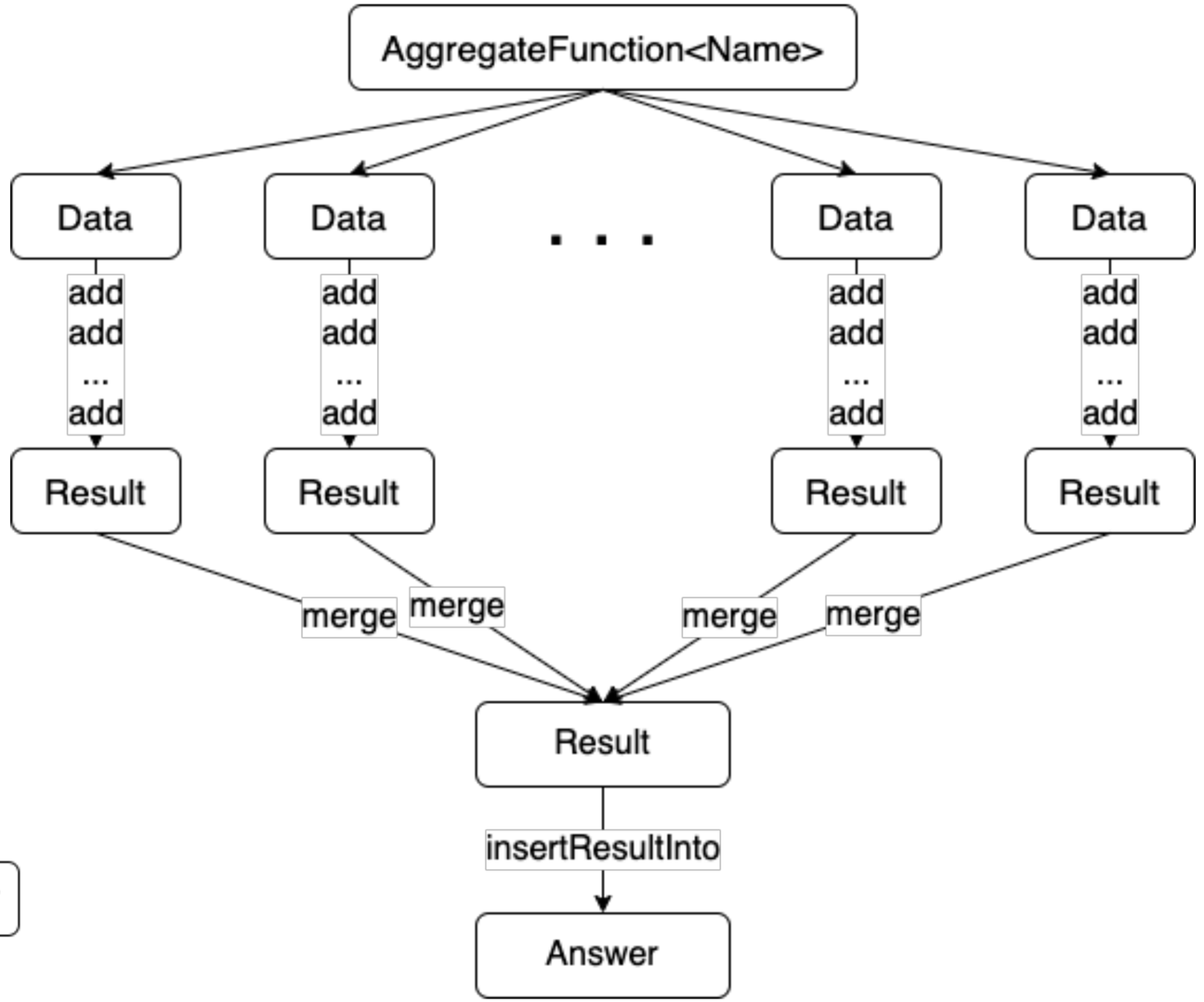
Обработка запросов

1. Распределение на исполняющие элементы
2. Создание экземпляра подходящего класса
3. Распределение данных на несколько потоков исполнения
4. Обработка данных каждым потоком исполнения
5. Совмещение результатов
6. Присоединение к ответу

Агрегатные функции

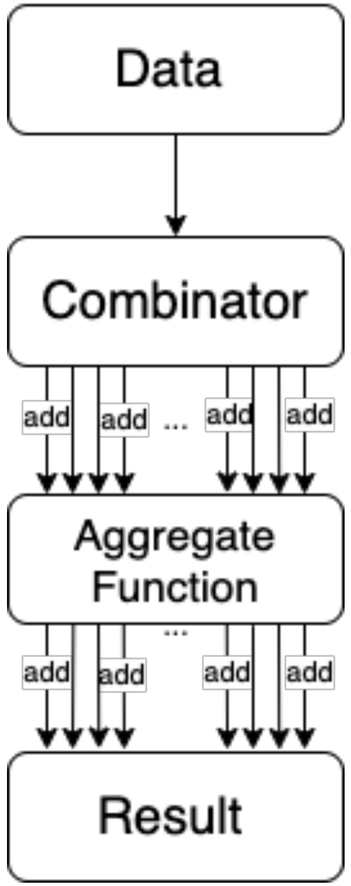
- Собирают статистику данных
- Реализуются наследованием от интерфейса IAggregateFunction
- Работают по идеологии MapReduce в несколько потоков исполнения
- Могут хранить данные в своём состоянии





Комбинаторы агрегатных функций

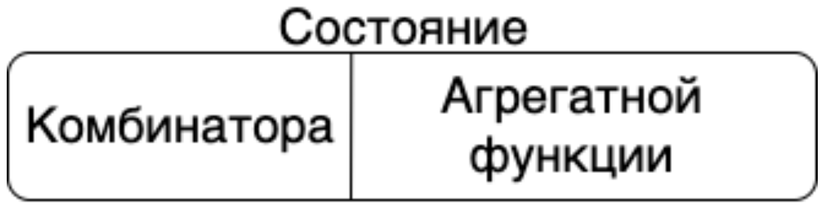
- Реализуют логику модификаторов агрегатных функций
- Являются агрегатными функциями
- Можно комбинировать с другими агрегатными функциями, изменяя их поведение
- Обрабатывают данные перед попаданием во вложенную агрегатную функцию
- Базовый комбинатор в каждом методе вызывает такой же метод с такими же параметрами у вложенной агрегатной функции



Решение

Общая структура

- Реализация в рамках комбинаторов агрегатных функций
- Данные хранятся в состоянии комбинатора
- Состояние комбинатора объединено с состоянием вложенной агрегатной функции
- Состояние вложенной агрегатной функции получается смещением общего состояния на размер структуры данных для комбинатора

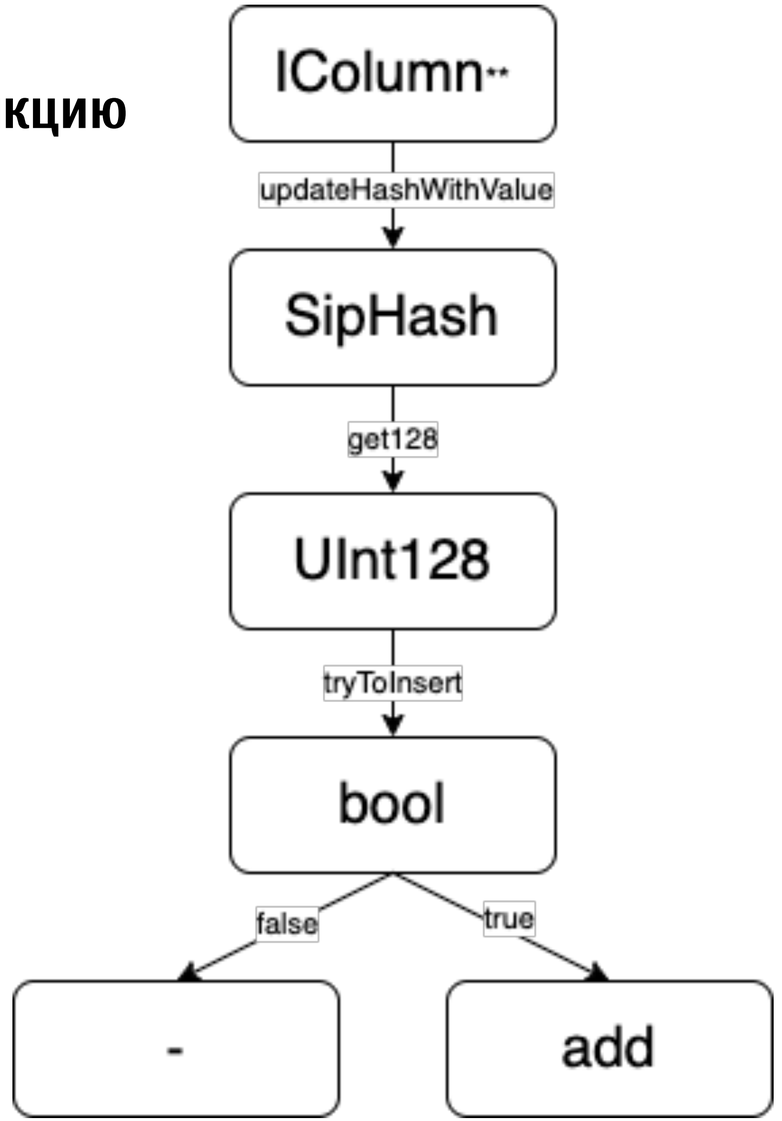


Модификатор DISTINCT

Уникализирует данные перед попаданием в агрегатную функцию

- Аргументы - данные для уникализации
- Не изменяет количество столбцов данных
- Данные хранятся в состоянии комбинатора
- Для хранения данных используется структура AggregateFunctionDistinctData
- Метод add проверяет строку данных на уникальность и передает данные вложенной агрегатной функции в случае уникальности

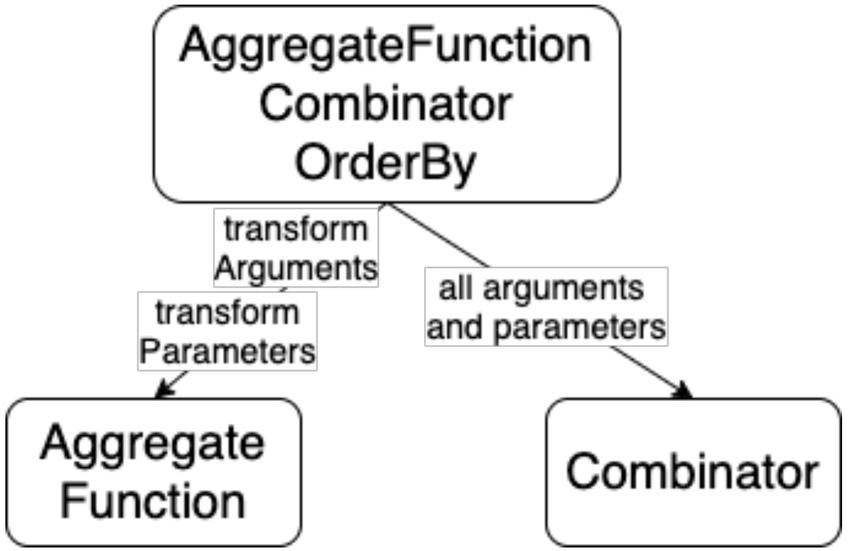
```
SELECT sumDistinct(Sign)
FROM datasets.visits_v1

┌─sumDistinct(Sign)─┐
│                 0 │
└───────────────────┘

1 rows in set. Elapsed: 0.048 sec.
```



Модификатор ORDER BY

Упорядочивает данные перед попаданием в агрегатную функцию

- Аргументы - столбцы, которые и по которым нужно упорядочить
- Параметр - количество столбцов, по которым нужно упорядочить данные
- Возвращает в агрегатную функцию только столбцы, которые нужно упорядочить

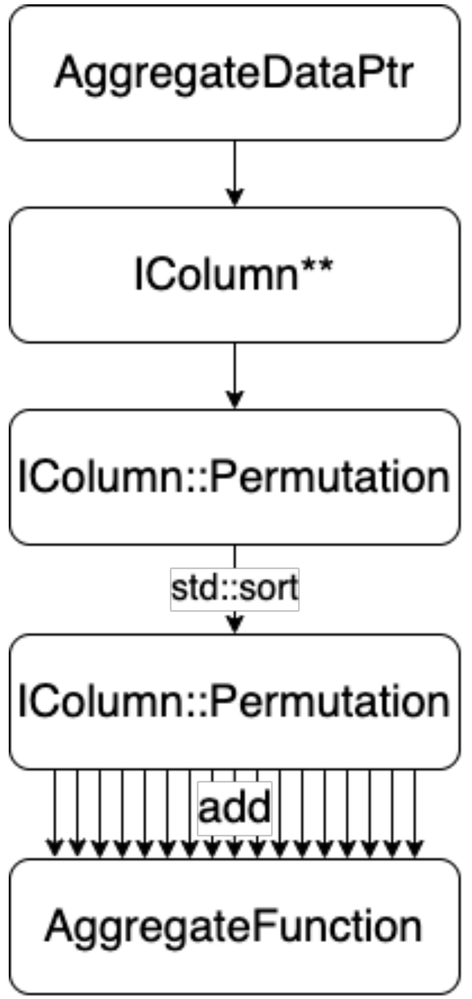



```
SELECT groupArrayOrderBy(2)(x, y, z)
FROM
(
    SELECT
        number AS x,
        number % 3 AS y,
        number % 5 AS z
    FROM system.numbers
    LIMIT 10
)

┌─groupArrayOrderBy(2)(x, y, z)─┐
│ [0,6,3,9,1,7,4,5,2,8]         │
└───────────────────────────────┘

1 rows in set. Elapsed: 0.005 sec.
```

Модификатор ORDER BY

Упорядочивает данные перед попаданием в агрегатную функцию

- Данные хранятся в состоянии комбинатора
- Для хранения данных используется структура AggregateFunctionOrderByData
- Метод add добавляет получаемые данные в состояние комбинатора
- Метод insertResultInto упорядочивает данные и передает их вложенной агрегатной функции



Проверка результатов

Тестирование

- Сборка проекта с внесенными изменениями
- Запуск различных запросов из интерактивной консоли ClickHouse
- Использование различных данных:
  - Примеры данных ClickHouse
  - Системные таблицы
- Сравнение времени исполнения с другими комбинаторами

```
SELECT sumDistinct(Sign)
FROM datasets.visits_v1

┌─sumDistinct(Sign)─┐
│                 0 │
└───────────────────┘

1 rows in set. Elapsed: 0.048 sec.
```

```
SELECT sumDistinct(x)
FROM
(
    SELECT number % 13 AS x
    FROM system.numbers
    LIMIT 100
)

┌─sumDistinct(x)─┐
│             78 │
└────────────────┘

1 rows in set. Elapsed: 0.004 sec.
```

```
SELECT groupArrayOrderBy(1)(x, y)
FROM
(
    SELECT
        number AS x,
        number % 3 AS y
    FROM system.numbers
    LIMIT 10
)

┌─groupArrayOrderBy(1)(x, y)─┐
│ [0,3,6,9,1,4,7,2,5,8]      │
└────────────────────────────┘

1 rows in set. Elapsed: 0.006 sec.
```

Функциональные тесты

- Тесты на языке SQL
- Имеют заранее известные ответы
- Используют только встроенные, системные или временные таблицы
- Запускаются в GitHub при добавлении изменений в проект

```sql
SELECT sum(DISTINCT x) FROM
(SELECT number AS x FROM system.numbers LIMIT 1000);

SELECT sum(DISTINCT x) FROM
(SELECT number % 13 AS x FROM system.numbers LIMIT 1000);

SELECT groupArray(DISTINCT x) FROM
(SELECT number % 13 AS x FROM system.numbers LIMIT 1000);

SELECT groupArray(DISTINCT x) FROM
(SELECT number % 13 AS x FROM system.numbers_mt LIMIT 1000);

SELECT corrStableDistinct(DISTINCT x, y) FROM
(SELECT number % 11 AS x, number % 13 AS y FROM system.numbers LIMIT 1000);
```

```sql
SELECT groupArrayOrderBy(2)(x, y, z) FROM
(SELECT number AS x, number % 3 AS y, number % 5 AS z
FROM system.numbers LIMIT 10);

SELECT groupArrayOrderBy(1)(x, y) FROM
(SELECT number AS x, number % 3 AS y FROM system.numbers_mt LIMIT 10);

SELECT groupArrayOrderBy(1)(x, y) FROM
(SELECT number AS x, number AS y FROM system.numbers_mt LIMIT 10);
```

Заключение

Результаты

- Изучена архитектура агрегатных функций и комбинаторов, а также некоторых других уже реализованных в ClickHouse структур
- Продумана архитектура комбинаторов DISTINCT и ORDER BY
- Внесены правки в общую архитектуру комбинаторов
- Реализованы и подключены к фабрике комбинаторы DISTINCT[1] и ORDER BY[2]
- Протестированы новые комбинаторы
- Написаны автоматические (функциональные) тесты для новых комбинаторов
- На данный момент код находится в состоянии code-review

[1] https://github.com/ClickHouse/ClickHouse/pull/10930
[2] https://github.com/ClickHouse/ClickHouse/pull/11235

Развитие

- Внести изменения в код, если это потребуется по итогам review
- Внести изменения двух pull requests в основную кодовую базу ClickHouse
- Внедрить поддержку не только внутреннего SQL (aggFuncOrderBy(1)(x, y)), но и обычного (aggFunc(x ORDER BY y))
- Выяснить потребность пользователей в уникализации данных не по возвращаемому столбцу (aggFunc(x DISTINCT y)) и реализовать в случае наличия

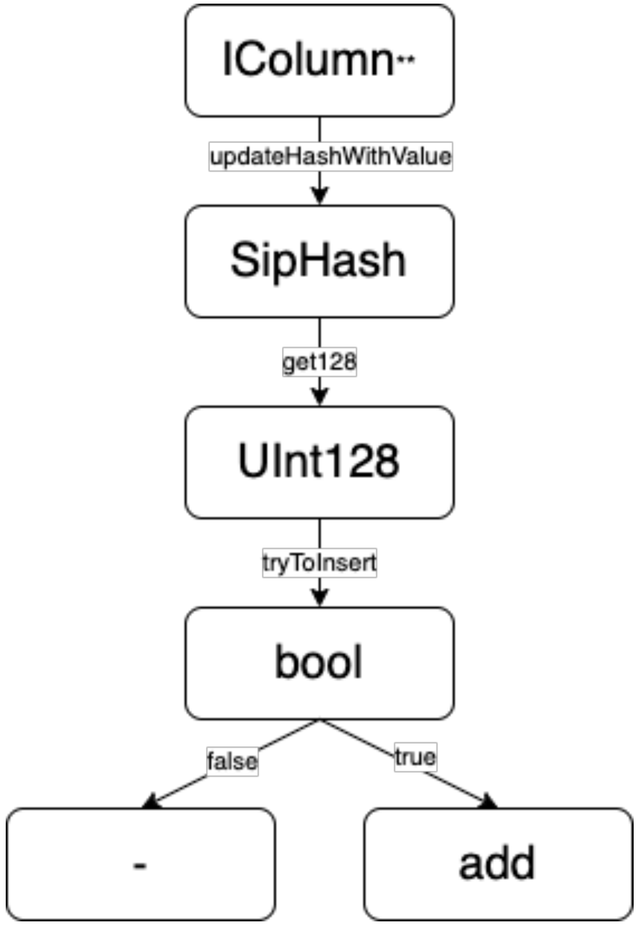
Спасибо за внимание

DISTINCT: AggregateFunctionDistinctData

- Хранилище данных:
  1. HashSet<UInt128, ...>
  2. Хранятся преобразованные к UInt128 хэши данных
- Метод для вставки данных:
  1. bool tryToInsert(UInt128 key)
  2. Пытается вставить элемент в HashSet по ключу key
  3. Возвращает true в случае успеха, false — если значение не уникально

DISTINCT: Метод add

- Создает экземпляр класса SipHash
- Обновляет SipHash для всех столбцов данных с помощью метода updateHashWithValue структуры передаваемых данных
- Преобразует SipHash к UInt128
- Пытается вставить данные в состояние комбинатора
- В случае успеха (true) вызывает метод add вложенной агрегатной функции

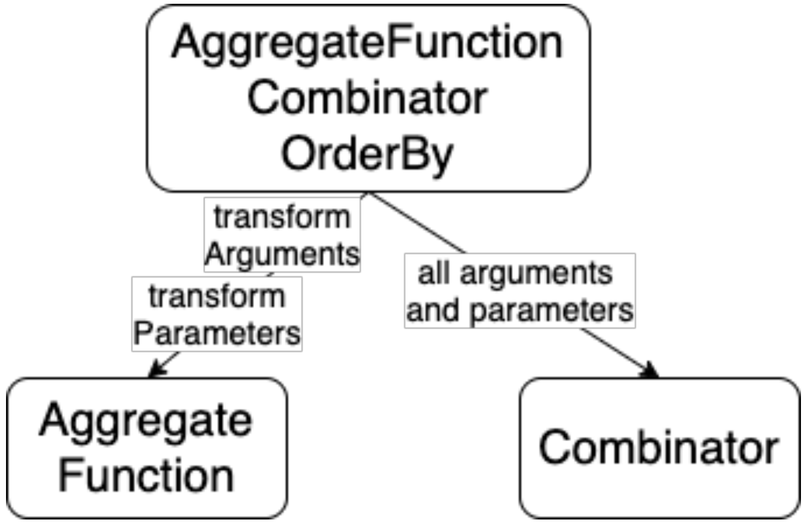


ORDER BY: AggregateFunctionOrderByData

- Хранилище данных:
  1. std::vector<IColumn*> — для хранения получаемой строки
  2. std::once_flag — для определения инициализации
  3. Arena* — для выполнения операций

ORDER BY: Аргументы и параметры

- Передаваемый параметр обозначает количество столбцов, по которым необходимо провести упорядочивание
- В функции transformArguments происходит отделение столбцов, которые должны быть переданы в агрегатную функцию, с помощью переданного параметра (для этого изменили интерфейс функции, чтобы она принимала не только аргументы, но и параметры)
- В функции transformParameters удаляется параметр комбинатора перед передачей параметров в агрегатную функцию
- В комбинатор попадают все аргументы и все параметры



ORDER BY: Метод add

- Один раз для всех потоков инициализирует структуру хранения данных типом получаемых данных и количеством столбцов, используя вызов call_once и флаг из состояния комбинатора
- Инициализирует арену в состоянии комбинатора
- Добавляет строку данных в состояние комбинатора

ORDER BY: Метод insertResultInto

- Создает перестановку из данных, лежащих в состоянии комбинатора
- Сортирует данные по заданным столбцам
- Проходится по отсортированным данным и вызывает для каждого элемента данных метод add вложенной агрегатной функции, используя арену из состояния комбинатора
- Вызывает метод insertResultInto вложенной агрегатной функции