DB internals. Вторая лекция

Надуткин Федор

December 2023

Метаданные операторов

Не всегда ясно, как лучше проводить оптимизацию (например как лучше поменять порядок `Join`), поэтому на помощь приходят метаданные. Чем больше мы знаем о природе данных, возвращаемых оператором, тем больше полезных оптимизаций можно применить.

Метаданные — конкретные значения/структуры, которые мы можем извлечь из конкретного узла.

```
interface PlanNode {
    List<PlanNode> getInputs ( );
    T accept (PlanVisitor<T> visitor);
}

interface PlanVisitor<T> {
    T visitScan (Scan node);
    T visitProject(Project node);
    T visitFilter (Filter node);
    T visitAggregate (Aggregate node);
    T visitJoin(Join node);
}
```

- Расчёт метаданных происходит рекурсивно. Перед тем как получить метаданные оператора, надо получить метаданные дочерних операторов.
- Зачастую метаданные могут повторяться, поэтому иногда их стоит кешировать.

Статистики

Статистики — метаданные, которые приблизительно или точно описывают значения атрибутов.

- Row count — сколько записей возвращает оператор. Важнейший фактор оценки стоймости операторов.
- Min/Max — для оценки селективности предикатов.
- Null count — для достаточно частого запроса `not null`
- NDV (number of distinct values) — помогает оценить стоймость работы таких операторов как Aggregate или Join. Могут быть рассчитаны не только для отдельных атрибутов, но и для их комбинаций.
- Гистограммы — распределение значений атрибутов.

1) Для листьев плана статистики представляет движок.
2) Для остальных операторов, статистики рассчитываются на основе статистик входов.
    - Эвристики - наиболее распространённый подход, однако неточный.
    - Использование статистик уже выполненных планов.
    - ML

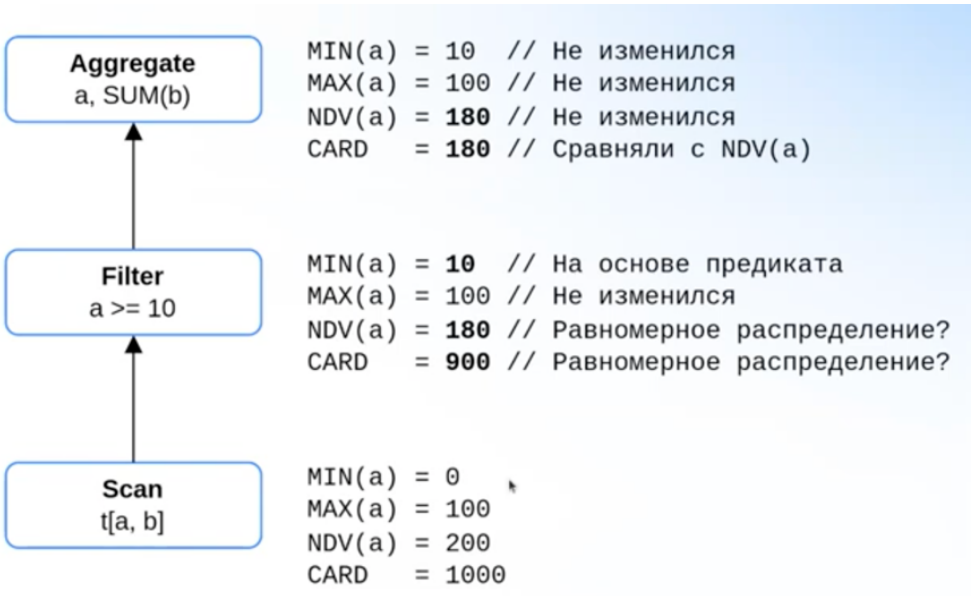

Рис. 1. Пример расчёта статистики. (CARD - координальность оператора)

Constraints

Ограничения, накладываемые оператором.

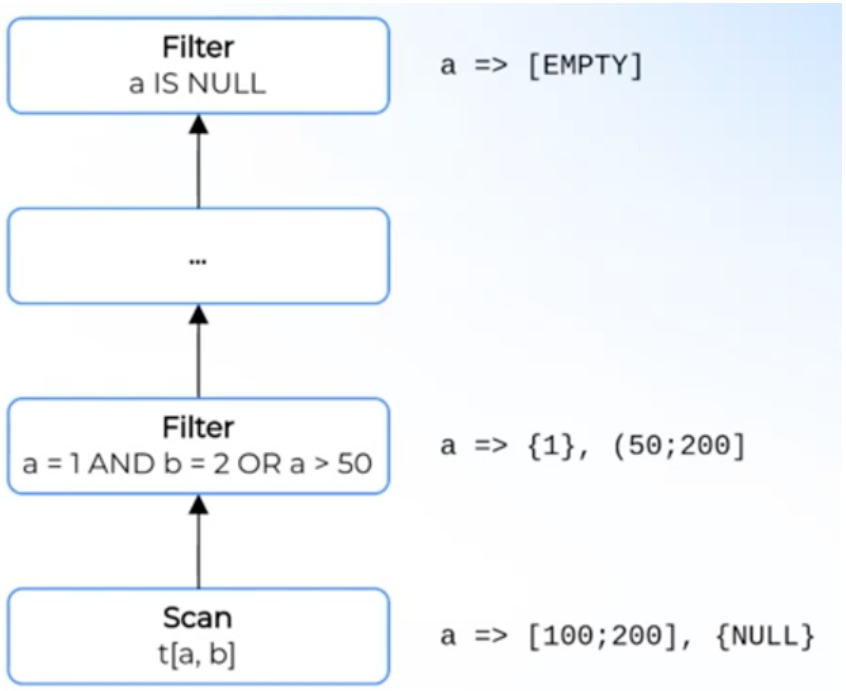

Рис. 2. Пример оптимизации запроса, исходя из ограничений

Уникальность

Исходя из работы операторов можно понять, что некий атрибут уникален, а как следствие упрощение дерево разбора.

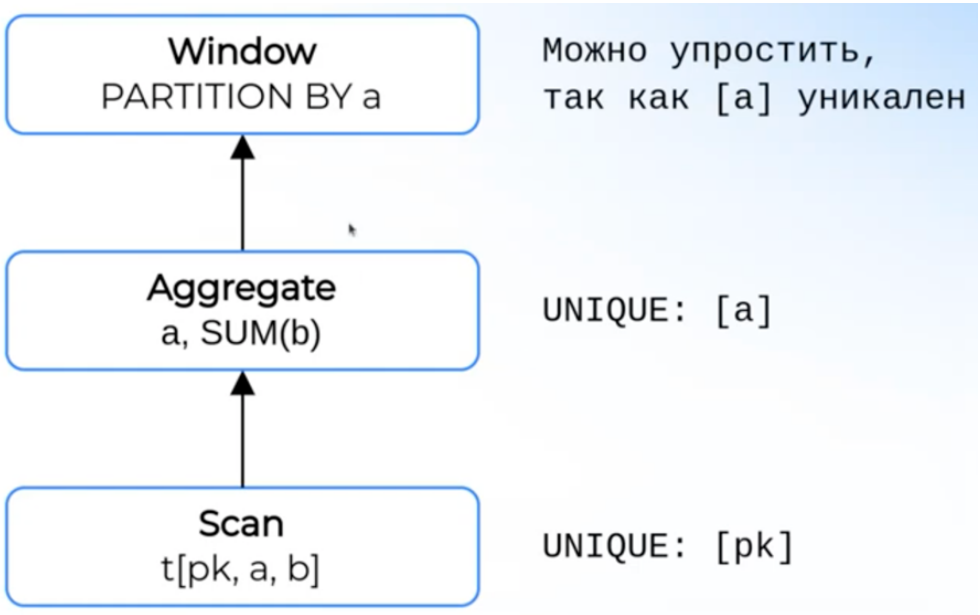

Рис. 3. Пример оптимизации запроса, исходя из уникальности

Свойства

Свойство — характеристика данных, которую можно изменить путём добавления к плану специального оператора к плану. Примеры:

- **Sortedness** - Отсортированность кортежей в отношении. Используется, например, в merge sort.
- **Distribution** - Распределение данных по вычислительным элементам.

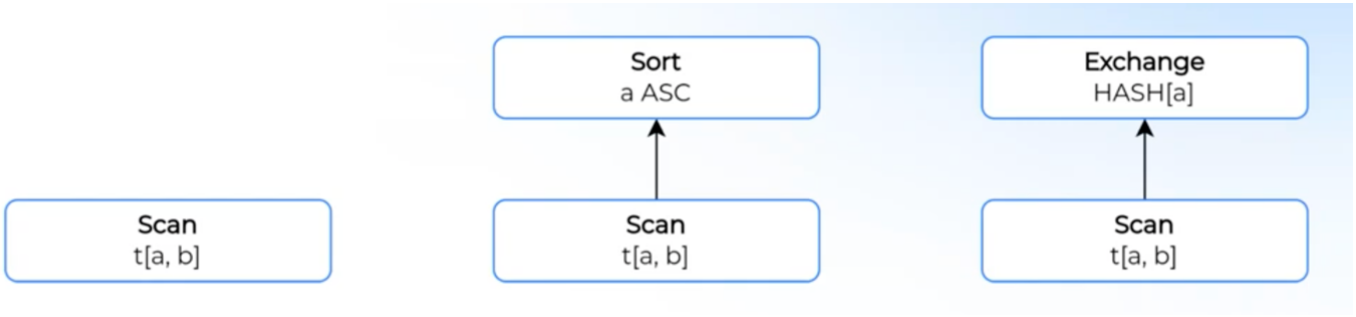

Рис. 4. Пример добавления операторов для изменения свойств

Оптимизация запросов

Эквивалентные планы — планы, производящие эквивалентные отношения для любых входных данных.

Стоймость плана — величина, описывающая трудоёмкость его выполнения (Какой-то скаляр/вектор/...)

Трансформация плана — замена текущего плана на эквивалентный. Производится с целью снижения стоймости плана.

Оптимизация — последовательность трансформаций плана.

Visitor

Многие системы используют паттерн `Visitor`. Пример кода указан выше.

- Позволяет обойти узлы плана в заданном порядке.
- Позволяет реализовать любую логику.
- Крайне громоздкий, поэтому используется для больших и сложных оптимизаций.

Rule-based оптимизация

```
class Rule {
    Pattern getPattern();
    PlanNode apply(Capture context);
}
```

- Изолированная трансформация, которую оптимизатор применяет к отдельной части плана в соответствии паттерном.
- Паттерн - логика сопоставления паттерна с определённым участком кода.
- Поиск паттерна может быть осуществлён разными алгоритмами.

Рис. 5. Пример матчинга паттерна и последующей оптимизации

Итеративный драйвер

Простейший паттерн, который проходит по дереву операторов, метча паттерны и выполняя изменения.

- Не учитывает специфику данных, из-за чего применённые оптимизации могут не давать выигрыша
- Возможна проблема зацикливания.

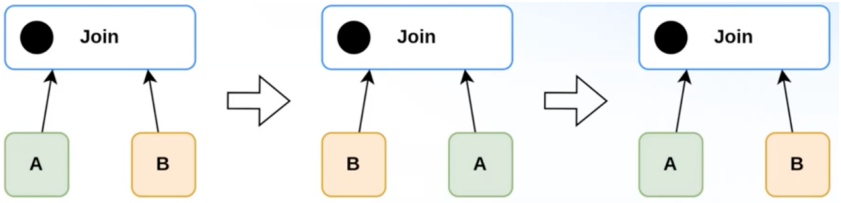

Рис. 6. Пример зацикливания

- Может зависеть от порядка обхода и вследствие чего может оказаться недооптимизированно.

Драйвер с мемоизацией

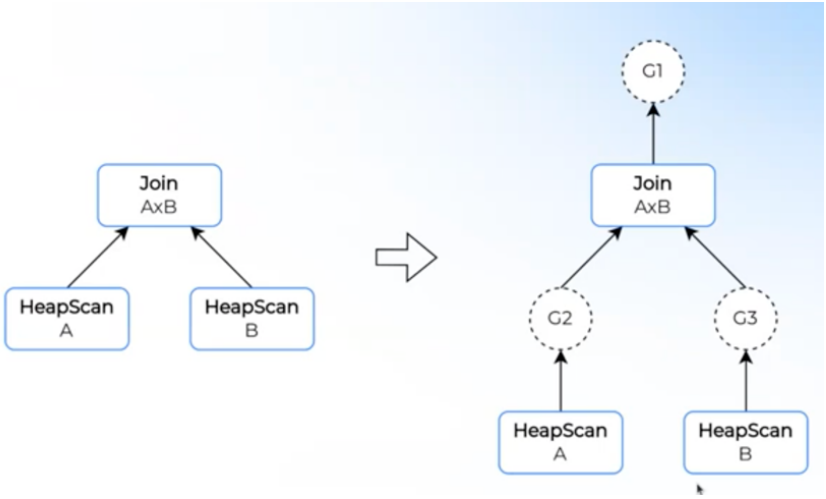

Рис. 7. Пример перевода из обычного графа в MEMO

MEMO — структура данных, которая позволяет компактно хранить большое количество планов. Представляет собой чередование тех операторов, что у нас были в старых графах и групп эквивалентности, содержащих один или несколько реальных узлов.

Memo крайне удобно добавление новых узлов, изменение в узле = ещё один AND оператор на входе в OR. Более того изменения порядка узлов также могут выглядеть не сложно и не опираться на реализацию драйвера.

Ещё одним плюсом Memo является простота поиска наиболее дешёвого плана. Если дать вершинам стоймость, то можно найти оптимальное дерево, выбирая на каждом OR оптимальную вершину.

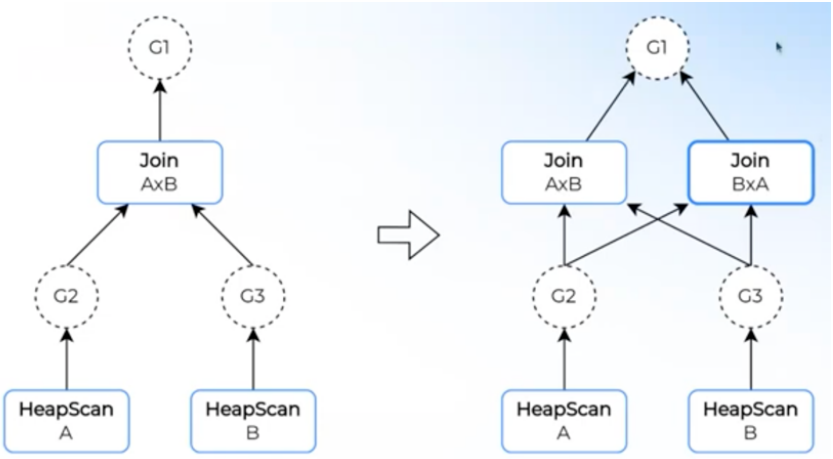

Рис. 8. Добавление узла в Memo

```
class FilterJoinTranspose {
    Pattern pattern = filter().join();

    void apply(Context context) {
        Operator filter = context.get(0);
        Operator join = context.get(1);

        Operator newJoin = new Join(filter.replace(join.left()), join.right());
        return newJoin;
    }
}
```

Листинг 1. Пример правила перестановки

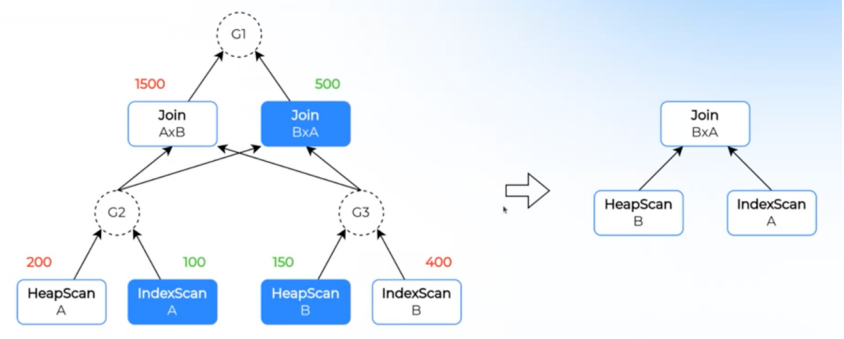

Рис. 9. Выбор оптимального дерева.

Properties and enforcers

Некоторые операторы задают требования к свойствам своего входа, поэтому можно либо добавить узлы, чтобы операторы имели такие свойства (например отсортировать) или проверить, что оно выполняется.

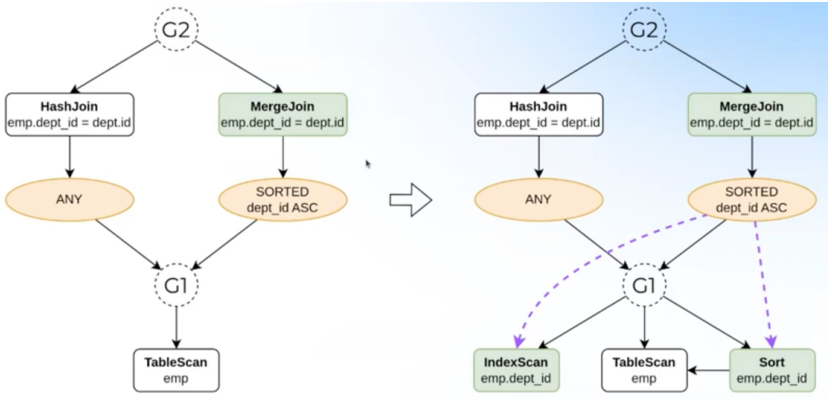

Рис. 10. Оптимизация по properties

Cascades

Cascades - алгоритм, который использует MEMO, rule-based оптимизацию и управление свойствами для поиска оптимальных планов. Мы можем совмещать оптимизации, спускаясь и поднимаясь по операторам и изменяя, если это принесёт оптимизацию.

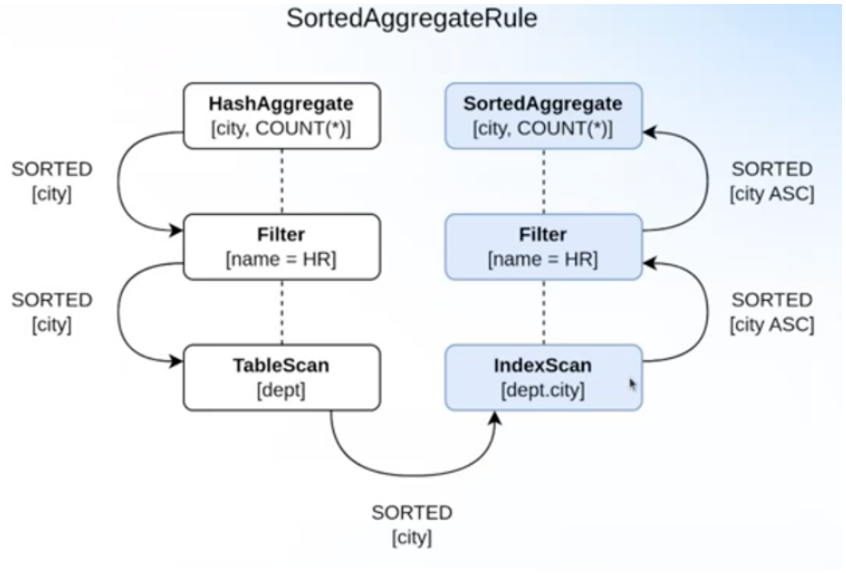

Рис. 11. Спуск в Cascades до TableScan

Однако такая технология может быть не во всех базах данных, так как несмотря на плюсы, сам алгоритм достаточно сложный и может привести к неожиданным эффектам.

Итоги

Оптимизаторы зачастую используют комбинации алгоритмов. Так они могут использовать iterative, потом cascades... Делается это чтобы количество планов не разрасталось из-за больших оптимизаторов.