DB internals. Третья лекция

Надуткин Федор

February 2024

Ключевые оптимизации

- **Нормализация** — приведение плана к каноническому виду, которы позволяет применить наибольшее количество оптимизаций.
- **Оптимизация** — Уменьшение количество вычислений. Нахождение более дешёвых планов.

На практике, в обычных движках нормализации и оптимизации чередуются подряд. Зачастую они бывают узко специализированными, поэтому «прийти и посмотреть как это сделано» не получится, нужно выделять паттерны.

Нормализация и упрощение выражений

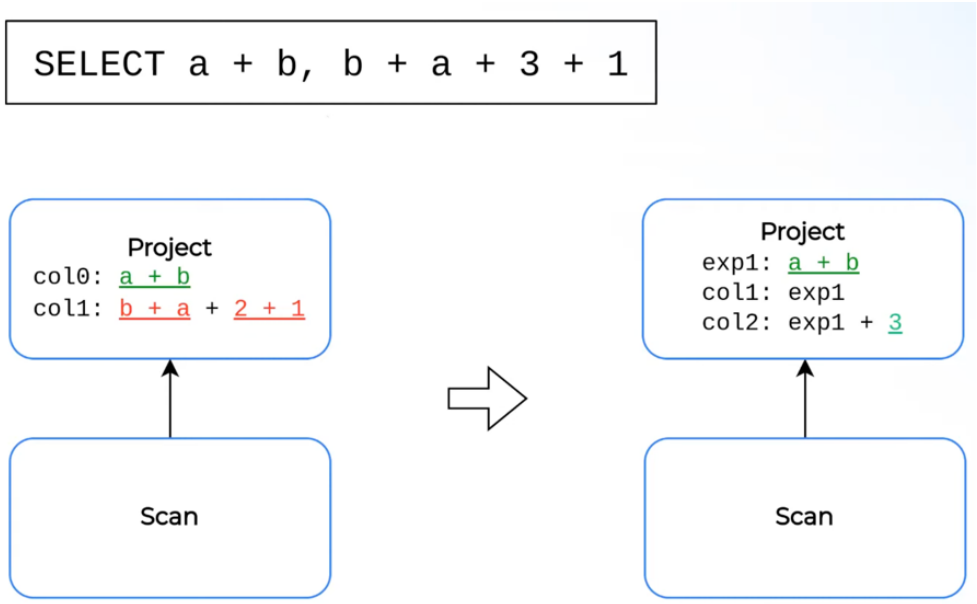


Рис. 1. Нормализация и упрощение выражений

Плюсы:

+ Отсутствие последующего пересчёта на следующих этапах.
+ Упрощение анализа. Например, становится легче искать похожие подпланы. Мы привели выражение в нормальную форму.

Минусы:

- Необходимы подключение интерпретаторов, оптимизаторов, калькуляторов . . . , что сильно усложняет структуру и может сделать проигрыш в некоторых ситуациях.
- Требуют дополнительной логики и алгоритмов.

Filter into Join

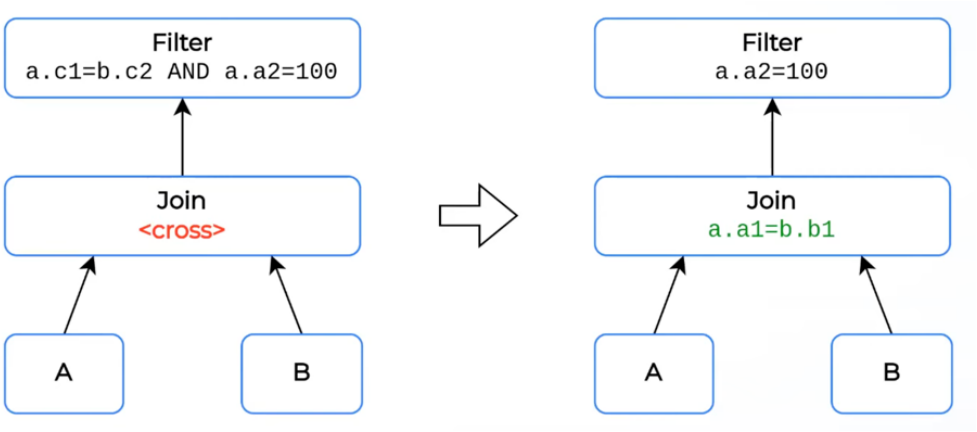


Рис. 2. Filter into Join

OuterJoin to InnerJoin

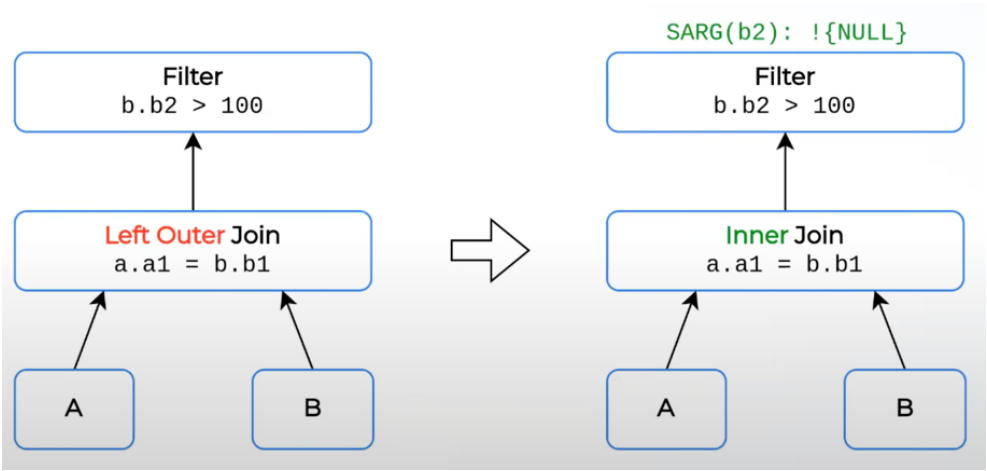


Рис. 3. OuterJoin to InnerJoin

В случае, если для левого ключа нет колонки справа (и наоборот) `OuterJoin` заменяет правый ключ на `NULL`. Если в будущем у нас будут операторы, которые отбрасывают такие ключи, то можно заменить `OuterJoin` на `InnerJoin`.

Упрощение агрегатов

Иногда данные в таблице у нас разрозненные, а как следствие агрегаторы по значению будут лишь возвращать значение по ключу. Такие операторы можно (и нужно) заменить или выбросить.

Одни операторы, через другие

Некоторые движки могут не поддерживать непосредственно какие-то операции, но хорошо поддерживают другие. Мы можем попробовать выразить один оператор через другие. Например, `Distinct Aggregate` через `Join`.

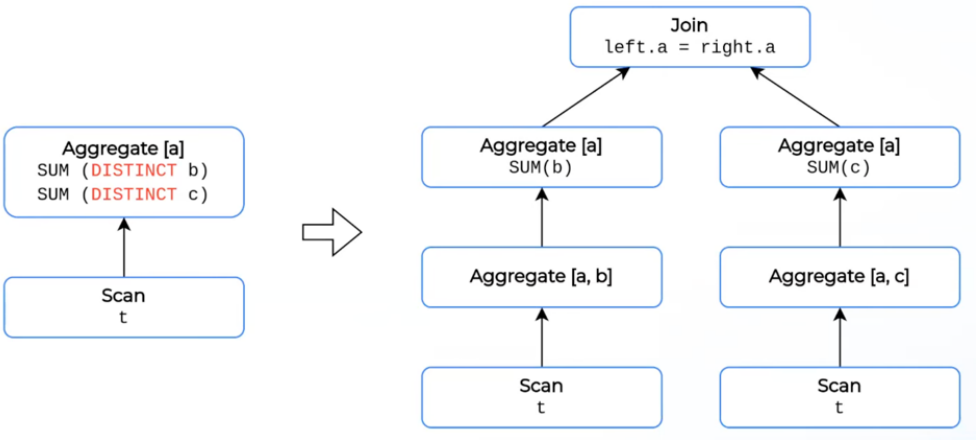


Рис. 4. Distinct Aggregate to Join

Или например `Intersect`, `Minus`, . . . выразить через `UnionAll`. Или например `Intersect`, `Minus`, . . . выразить через `UnionAll`.

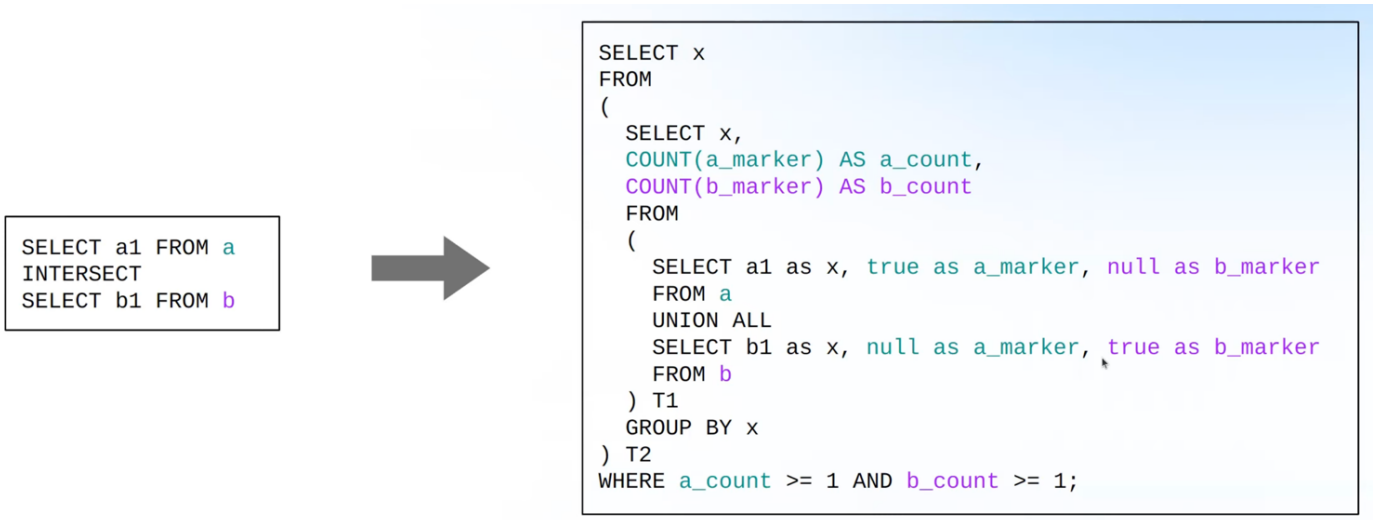


Рис. 5. Intersect из UnionAll

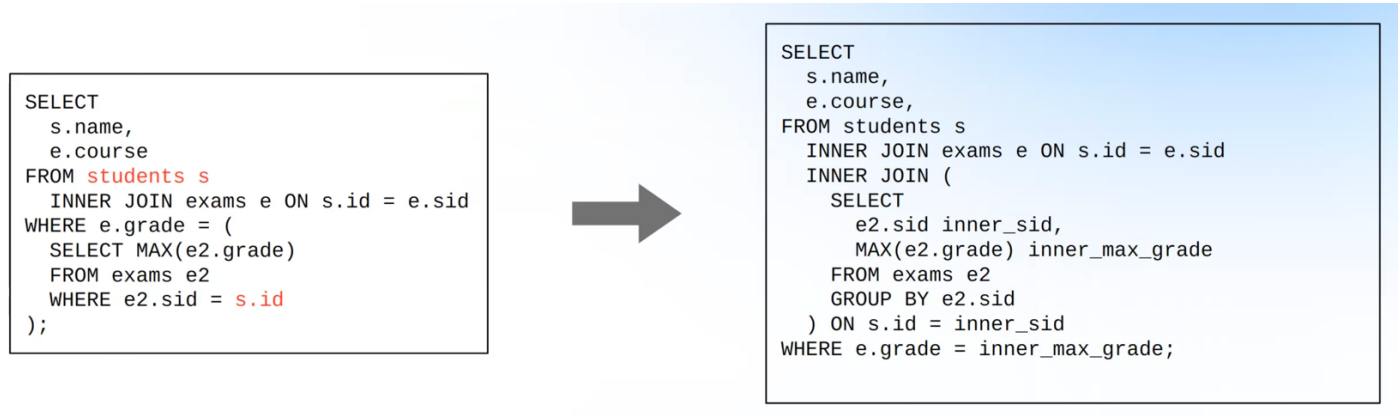
Переписывание подзапросов

Поддерживание коррелированных подзапросов достаточно трудная задача, более того в распределённых системах она только вредит, как следствие можно переделать запрос на запрос без корреляций.

Доказательство, что из любого коррелированного запроса можно сделать некоррелированный.

Хороший пример как делают в Алибабе (читать под ВПН).

На практике от коррелированных подзапросов не избавляются полностью, но стараются.



```
SELECT
  s.name,
  e.course
FROM students s
  INNER JOIN exams e ON s.id = e.sid
WHERE e.grade = (
  SELECT MAX(e2.grade)
  FROM exams e2
  WHERE e2.sid = s.id
);
```

```
SELECT
  s.name,
  e.course,
FROM students s
  INNER JOIN exams e ON s.id = e.sid
  INNER JOIN (
    SELECT
      e2.sid inner_sid,
      MAX(e2.grade) inner_max_grade
    FROM exams e2
    GROUP BY e2.sid
  ) ON s.id = inner_sid
WHERE e.grade = inner_max_grade;
```

Рис. 6. Переписывание запроса с корреляцией на запрос без неё

Filter pushdown

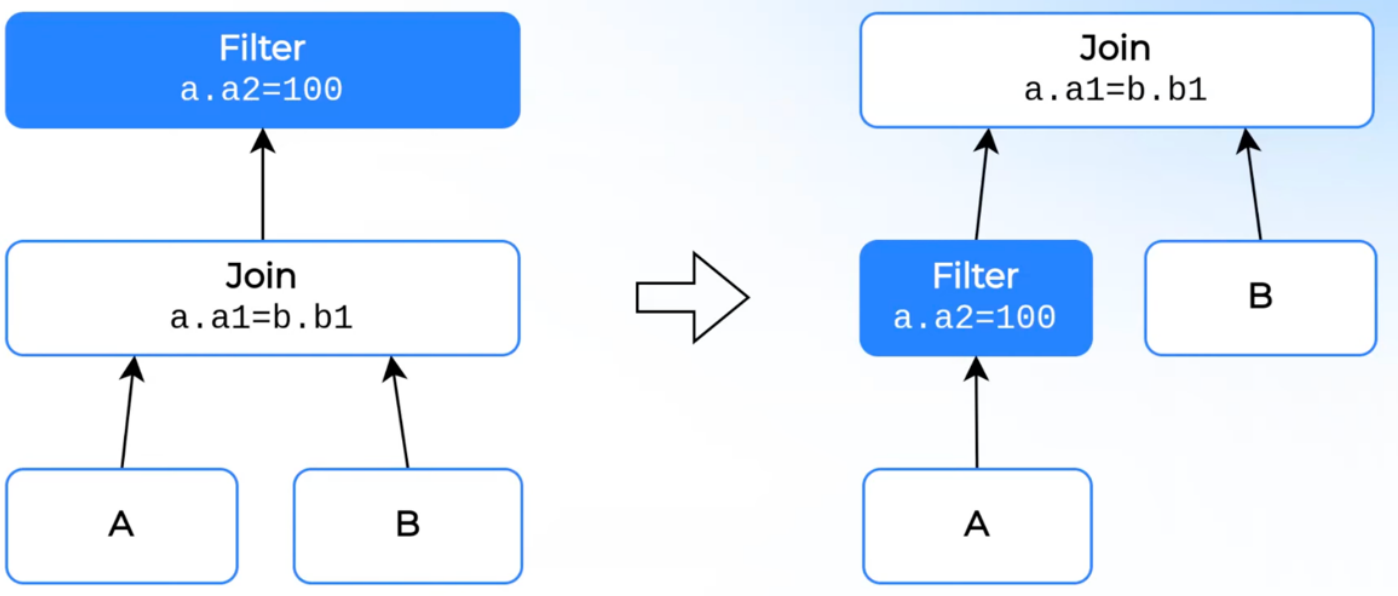


Рис. 7. Filter pushdown

`Filter` можно прокинуть практически через любой оператор, а значит опустить его всегда можно. Однако, если `Filter` не всегда хорошо фильтрует, то pushdown может и навредить.

Filter pull up

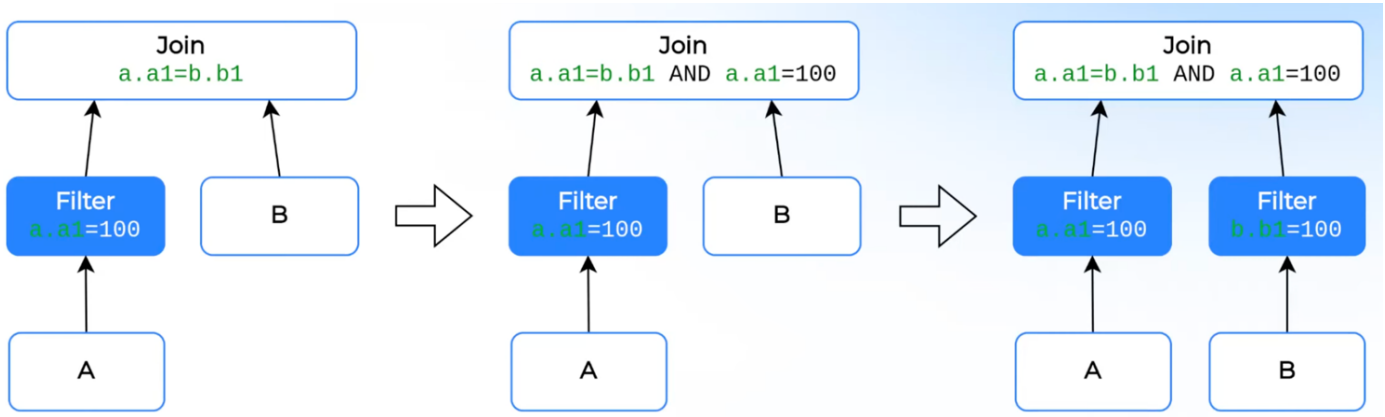


Рис. 8. Filter pull up

Перемещение фильтра наверх иногда позволяет создать дополнительные транзитивные предикаты.

Dynamic/Runtime Filters

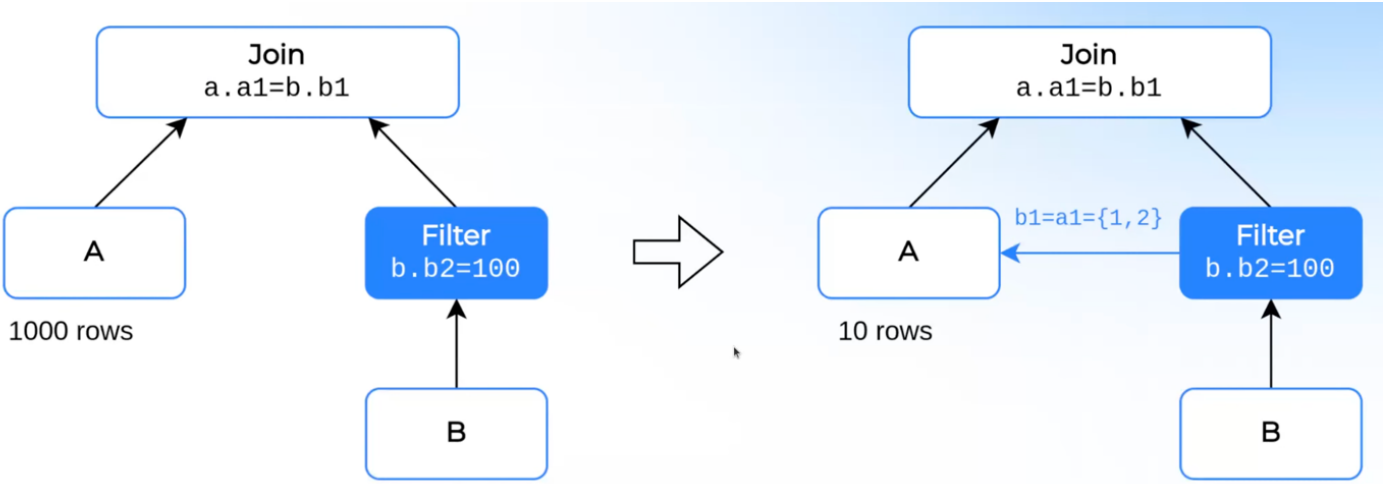


Рис. 9. Dynamic Filters

Можно посмотреть какие значения b_1 соответствуют $b_2 = 100$, если там будет мало различных значений, то мы получим дополнительный фильтр.

Pushdown других операторов

- **Project** — передача меньшего количества атрибутов между операторами.
- **Aggregate** – уменьшение количества кортежей как можно раньше.
- **Limit** — ограничить набор данных, возвращаемых нижестоящим оператором.

Spool

Поиск повторяющихся операторов.

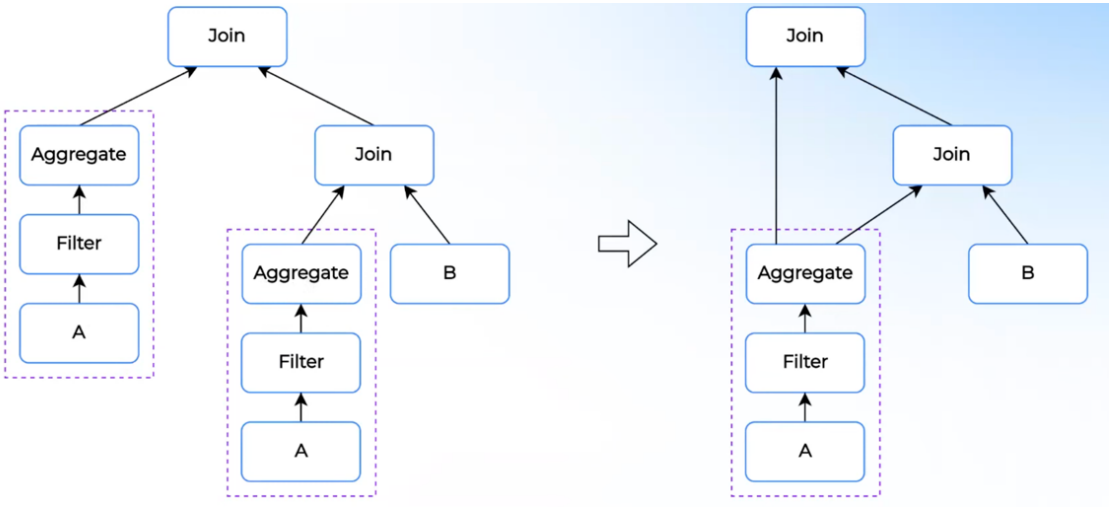


Рис. 10. Spool

Приводим дерево к ациклическому графу **DAG**, таким образом мы можем избежать выполнения идентичных подпланов.

- Может ухудшиться из-за снижения параллелизма.
- Риски `Deadlock`, так как оба оператора смотрят в одну и ту же область и могут её забивать и блокировать друг друга.

Materialized views

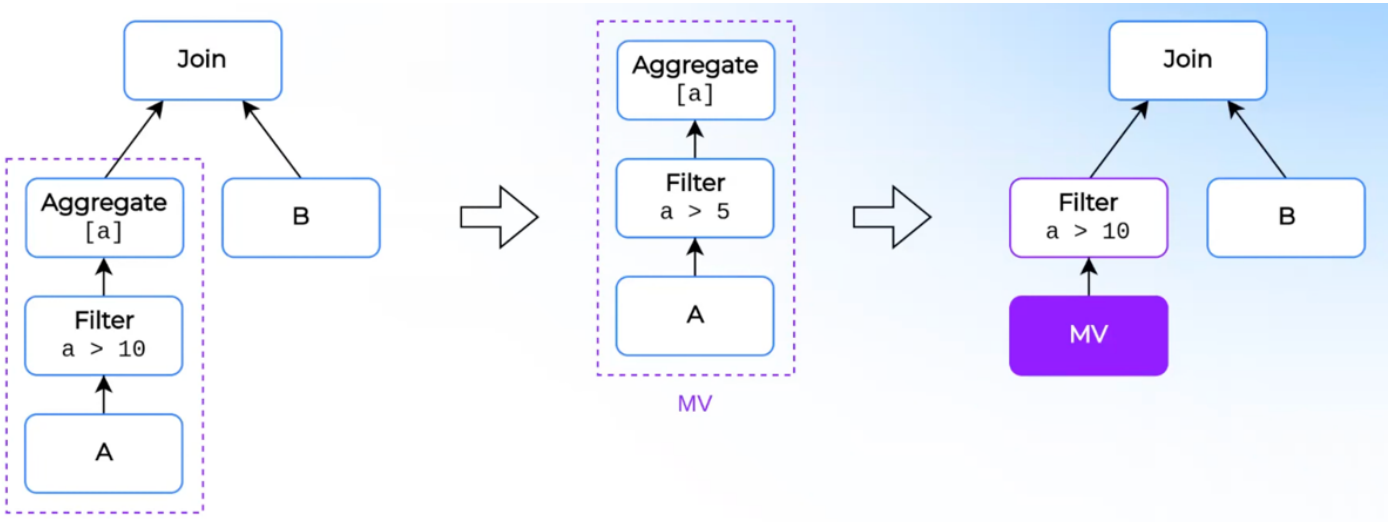


Рис. 11. Materialized views

Имея набор `Materialized views` мы можем сократить нагрузку на вычисления, однако это потребляет дополнительную память.

- Как подобрать оптимальный набор для заданной нагрузки?
- Как поддерево в плане переписать на `materialized view`?

Для решения этого есть:

- Базовый алгоритм
- Пример алгоритма.