DB internals. Первая лекция

Надуткин Федор

December 2023

Основы реляционной алгебры

Тип данных — описание множества возможных значений какой-либо величины и допустимых операций.

- Логический тип (`bool`).
- Числовые типы (`int`, `bigint`, `float`).
- Строковые (`char`, `varchar`).
- Дата и время (`date`, `timestamp`, ...).
- Контейнеры (`array`, `map`).
- Специальные типы (`binary`, `json`, `spatial`, ...)

Атрибут — пара <имя, тип>
Пример:

- name: `varchar`
- age: `int`

Значение атрибута — величина, принадлежащая типу атрибута
Пример:

- name: `varchar`: John
- age: `int`: 30

Кортеж(tuple) — несортированное множество атрибутов и их значений
Пример:

- [name:`varchar`:John, age:`int`:30]

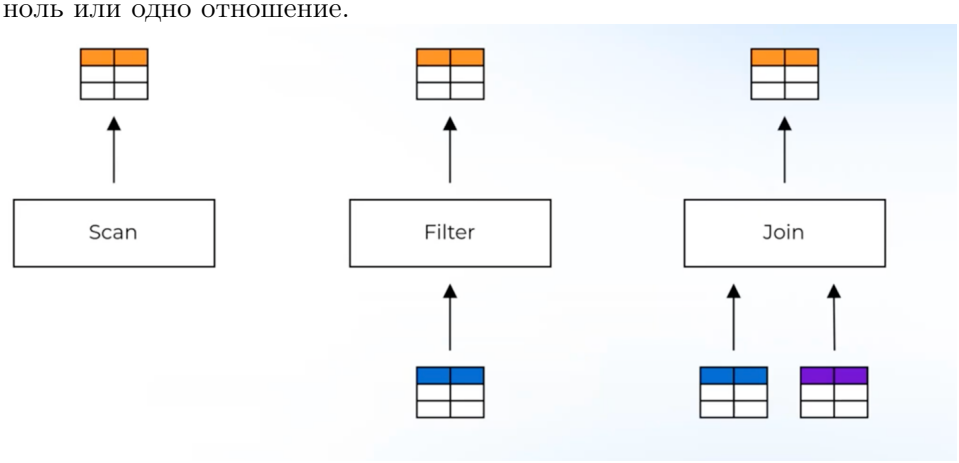
Отношение(relation) - несортированный набор кортежей. Отношение может содержать повторяющиеся кортежи

name:varchar	**age: int**
John	30
Jake	25
John	30

Эквивалентные отношения

- Одинаковый набор кортежей.
- Два кортежа равны, если они содержат идентичный набор атрибутов, и каждая пара одинаковых атрибутов имеет одинаковые значения.

Реляционные оператор — функция, которая принимает ноль/одно/несколько отношений и возвращает ноль или одно отношение.



Синтаксический анализ

```
SELECT dept.name, COUNT
      (* )
FROM emp, dept
WHERE
emp.dept_id = dept.id
GROUP BY dept. name
```

Листинг 1. Пример кода

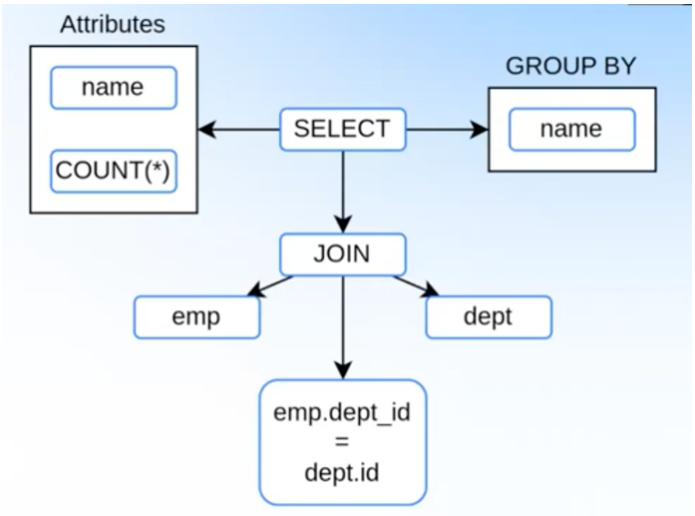


Рис. 1. Дерево разбора

Задача: Проверка, что запрос сформулирован правильно, согласно правилам запроса.
Результат: Синтаксическое дерево, если запрос сформулирован верно. Ошибка (зачастую достаточно точная), если запрос сформулирован неверно.
Примеры:

- Postgres/Bison
- Trino/ANTLR

Семантический анализ

Рис. 2. Пример семантического анализа

Проверяет логическую корректность запроса:

- Доступность объектов
- Семантика операторов

Обычно реализован в виде монолитного компонента, специфичного для конкретного движка.

Оптимизация

Оптимизация на основе AST

Некоторые движки (как например Postgres) реализуют планирование запросов на основе синтаксического дерева (или схожего представления).

\+ Быстро рекализуют некоторые оптимизации.

\- Ограничивает потенциал оптимизатора из-за сложно структуры синтаксического дерева.

Оптимизация на основе реляционного представления

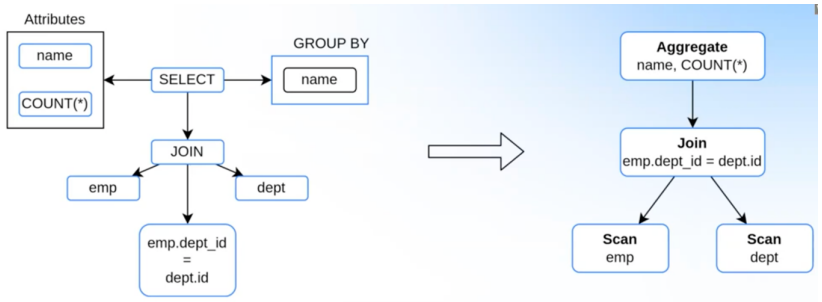


Рис. 3. Трансформация в реляционное дерево

В реляционном представлении операторы имеют простую семантику (`Scan`, `Project`, `Filter`, `Join`, ...), что позволяет реализовывать более широкий спектр трансформаций. Зачастую этот процесс происходит в ходе семантического анализа.

`Note:` Postgres трансформируется в реляционное дерево уже после оптимизации.

Row expressions

```
dept = 'HR'
AND salary > 100
```

Листинг 2. Пример кода

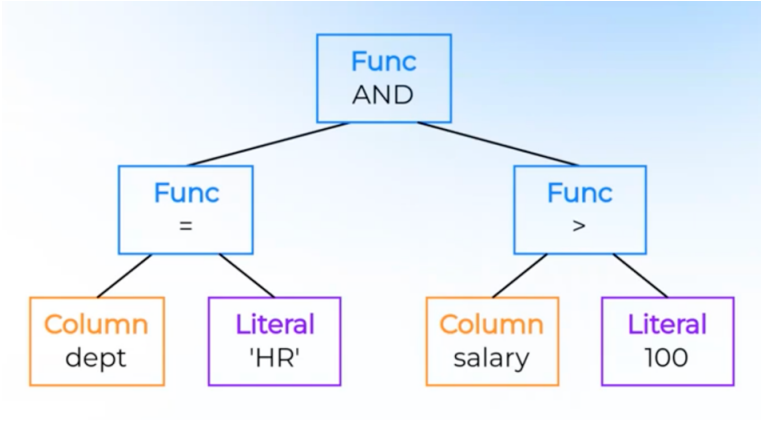


Рис. 4. Пример разбора

Row expression — дерево выражений, которое возвращает одно значение. Вход и выходы – конкретные значения.

```
interface RowExpression {
    Type getType ();
    T accept (Visitor<T> visitor);
    boolean equals ();
    int hashCode ();
}
```

Листинг 3. Интерфейс Row expression

Константа — терминальный узел, который содержит типизированное значение.

```
class Literal implements RowExpression {
    Object value;
    Type type;
}
```

Листинг 4. Класс константы

Атрибут — терминальный узел, который ссылается на значение в атрибуте текущего кортежа. Есть 2 способа представления атрибута.

- Адресация происходит по имени, порядок атрибутов в отношении не имеет значения.

```
class Column implements RowExpression {
    String name;
    Type type;
}
```

Листинг 5. Пример адресации по имени

- Адресация происходит по индексу, порядок атрибутов в отношении имеет значение.

```
class Column implements RowExpression {
    int index;
    Type type;
}
```

Листинг 6. Пример адресации по индексу

Функция — промежуточный узел, который содержит вызываемую функцию и аргументы.

```
class Call implements RowExpression {
    Function descriptor;
    Type type;
    List<RowExpression> arg;
}
```

Листинг 7. Функция

Реляционные операторы

Scan

```
SELECT city, amount
FROM sales
WHERE city = 'SPB'
```

Листинг 8. SQL приводящий в Scan

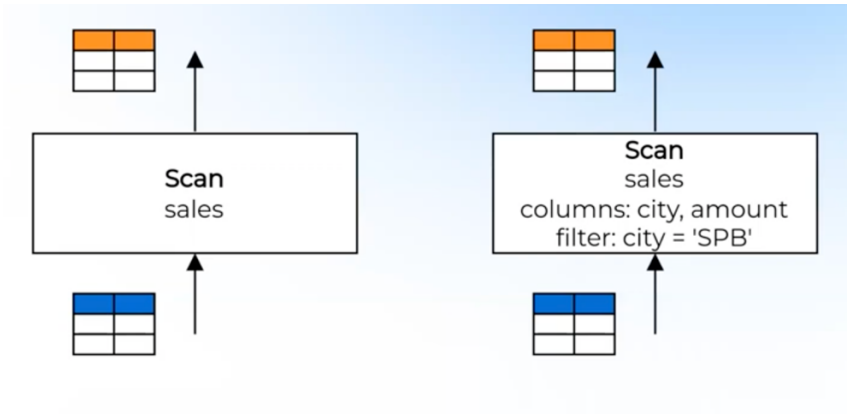


Рис. 5. Работа Scan

Scan — листовой оператор, который возвращает данные из какого-либо источника.

- Всегда содержит ссылку на объект сканирования.
- Может опционально содержать стратегию доступа к объекту.
- Может содержать дополнительную инормацию для оптимизации процедуры сканирования. (Колонки которые надо сканировать, дополнительные фильтры)

Project

```
SELECT
city ,
amount * comission as 'agent
    pay'
FROM sales
```

Листинг 9. SQL приводящий в Project

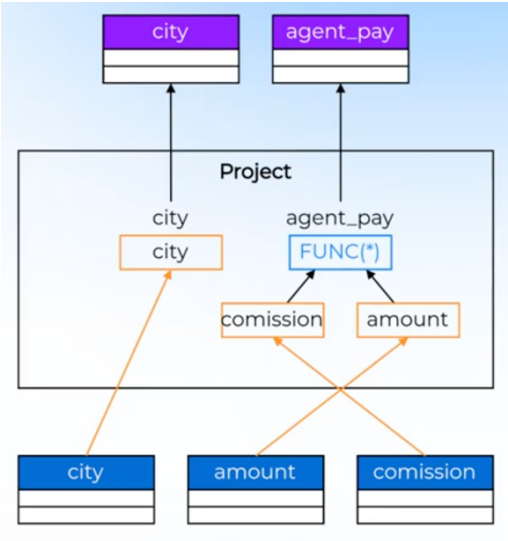


Рис. 6. Работа Project

Project — промежуточный оператор, формирующий из входного кортежа, новый кортеж с неким другим набором атрибутов при помощи коллекции **row expression**.

Filter

```
SELECT city, amount
FROM sales
WHERE city = 'SPB'
```

Листинг 10. SQL приводящий в Filter

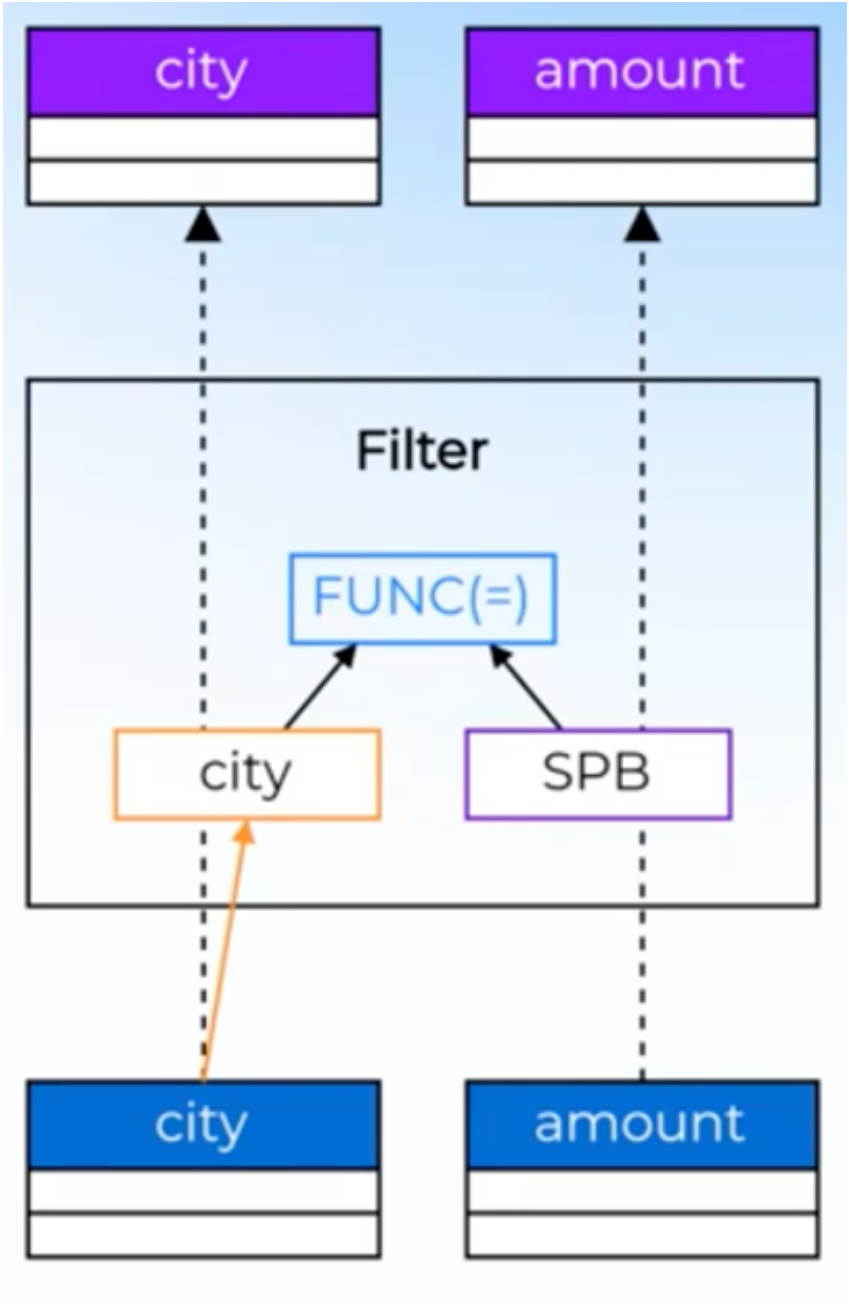


Рис. 7. Работа Filter

Filter — промежуточный оператор, который отфильтровывает определённые кортежи, не удовлетворяющие предикату (некоему **row expression**).

Aggregation

```
SELECT year, city,
    SUM(amount)
FROM sales
GROUP BY ROLLUP year, city
```

Листинг 11. SQL приводящий в Aggregation

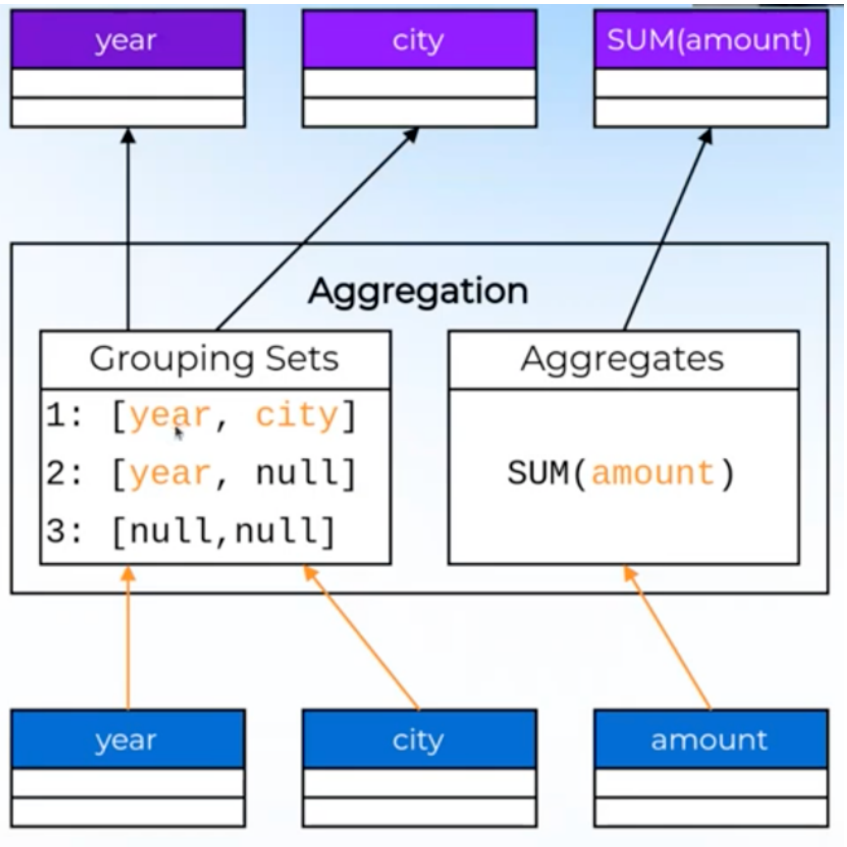


Рис. 8. Работа Aggregation

Aggregation — промежуточный оператор, который считает **Aggregates** по **Grouping Sets**.

Sort

```
SELECT year, city
FROM sales
ORDER BY year ASC, city
    DESC
```

Листинг 12. SQL приводящий в Sort

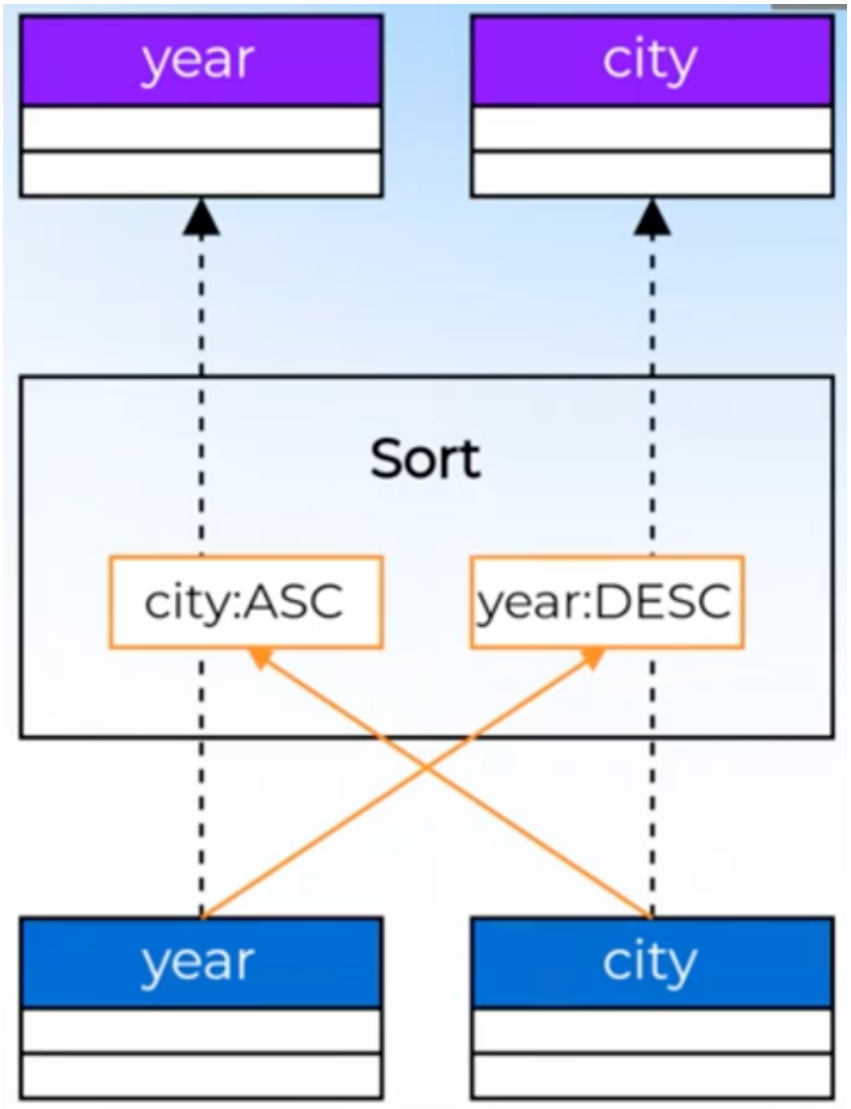


Рис. 9. Работа Sort

Sort — промежуточный оператор, меняющий порядок кортежей. С точки зрения SQL, имеет смысл для отображения результата, только конечному пользователю, так как обычно порядок не важен в отношениях. Может быть добавлен в обход пользователя (например для merge join) для будущих оптимизаций.

Join

```
SELECT city_id, amount, id,
    name
FROM sales JOIN city
ON sales.city_id = city.id
```

Листинг 13. SQL приводящий в Join

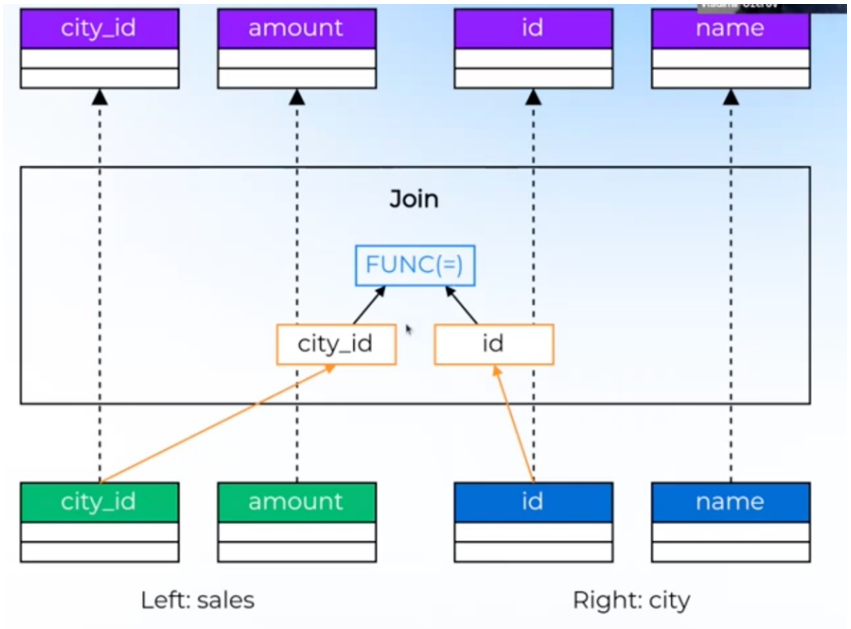


Рис. 10. Работа Join

Join — промежуточный оператор, который соединяет данные с нескольких входов по условию. **On** – по факту является неким предикатом, соединяющим нужные записи и отфильтровывающий не удовлетворяющие условию кортежи. Зачастую бинарный из-за простоты имплементации, но могут быть множественными (а также может переводить в множественный и оптимизировать начиная от него).

Set-оператор

```
SELECT s_city_id, s_amount
FROM stores sales
UNION ALL
SELECT c_city_id, c_amount
FROM catalog_sales
```

Листинг 14. SQL приводящий в Set

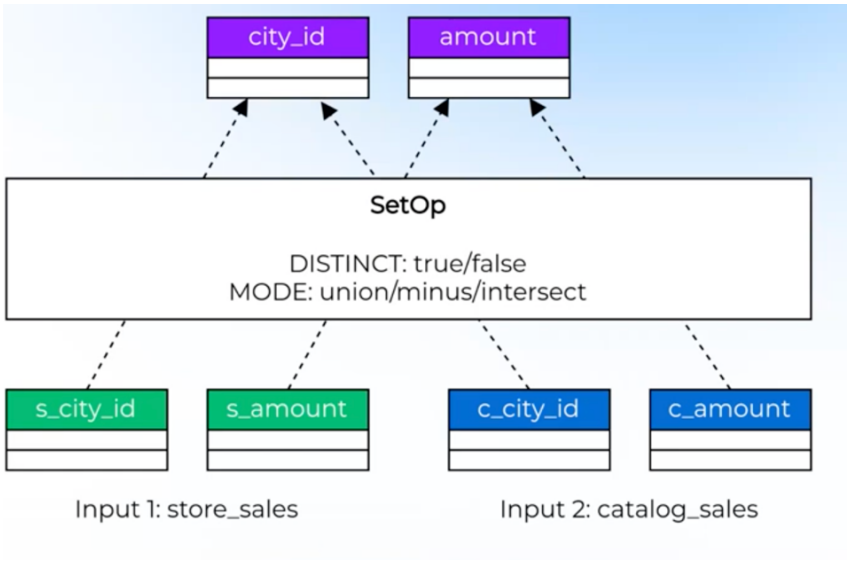


Рис. 11. Работа Set

Set-оператор (Union, Minus, Intersect) — промежуточный оператор, производящий объединение/вычитание/пересечени данных из нескольких источников.

Пример перевода запроса в реляционные операторы

```
SELECT
    dept,
    SUM(salary) as sum_salary
FROM employee
WHERE city = MSK
GROUP BY dept
HAVING SUM(salary) > 1000
ORDER BY SUBSTR (dept, 3)
    DESC
```

Листинг 15. Сложный запрос

```
1. Scan [employee]
2. Filter [city = MSK]
3. Aggregate [groupingSet=[dept], SUM
     (salary)]
4. Filter [sum_salary > 1000]
5. Project [dept, sum_salary,
     $sort_col=SUBSTR(dept, 3)]
6. Sort [$sort_col DESC]
7. Project [dept, sum_salary]
```

Листинг 16. Реляционные операторы

Декларативная и императивная программы

```
SELECT SUM(sales.amount )
FROM sales JOIN city
ON sales.city_id = city.id
WHERE city.name = 'SPB'
```

- **Декларативная прорамма**
  - Сделать Join таблиц
  - Применить фильтр
  - Сделать аггрегацию
- **Итеративная программа**
  - Отсканировать таблицу city
  - Вычислить city.id для подходящих под предикат строк, передать в sales
  - Построить hash-таблицу для подходящих записей city
  - Отсканировать sales, используя индекс по city_id
  - Многопоточно осуществить hash join и аггрегации
  - Соединение результатов потоков

Задача: оптимизация от декларативного исполнение к итеративному.