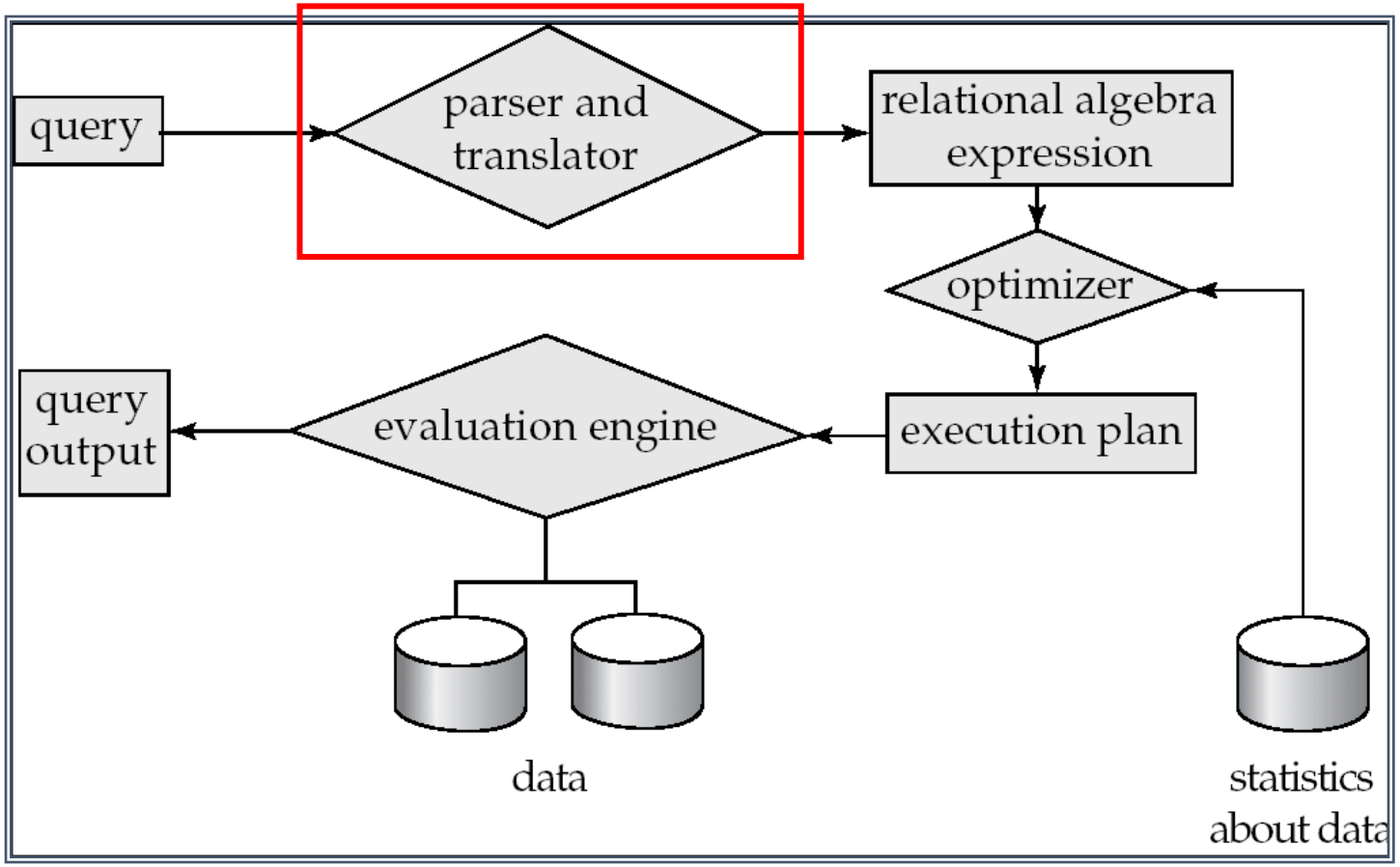
# Planos de execução

# Sumário

- Etapas no processamento de uma consulta SQL
  - Otimização de consultas
- Plano de execução
  - Fluxo de Execução
  - Read all
  - Read some

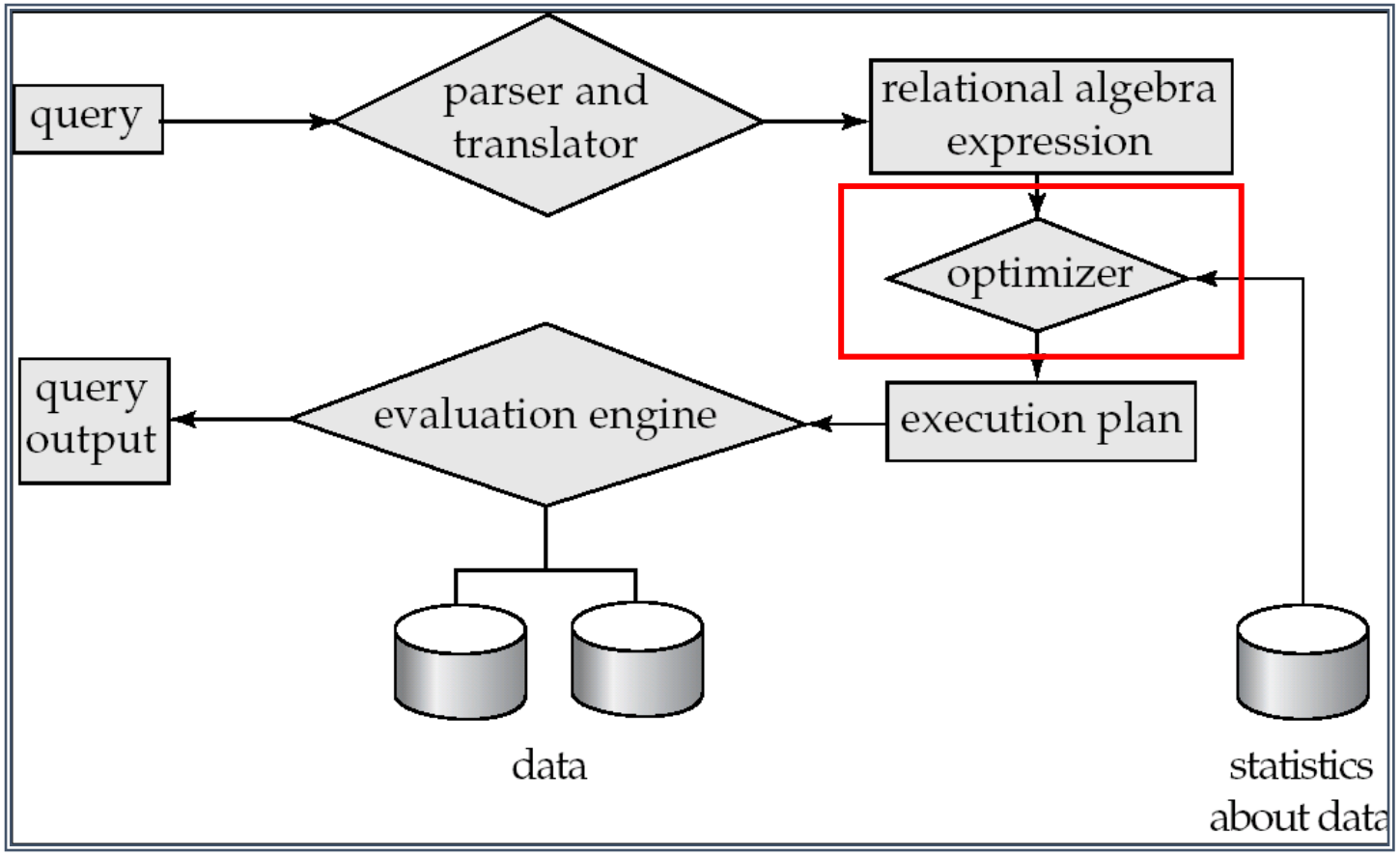
# Etapas de Processamento



# Etapas de Processamento

- Parser
  - verifica a sintaxe e os nomes usados
- Tradutor
  - reescreve a consulta em um formato de representação interno
- Similar ao propósito dos compiladores

# Etapas de Processamento

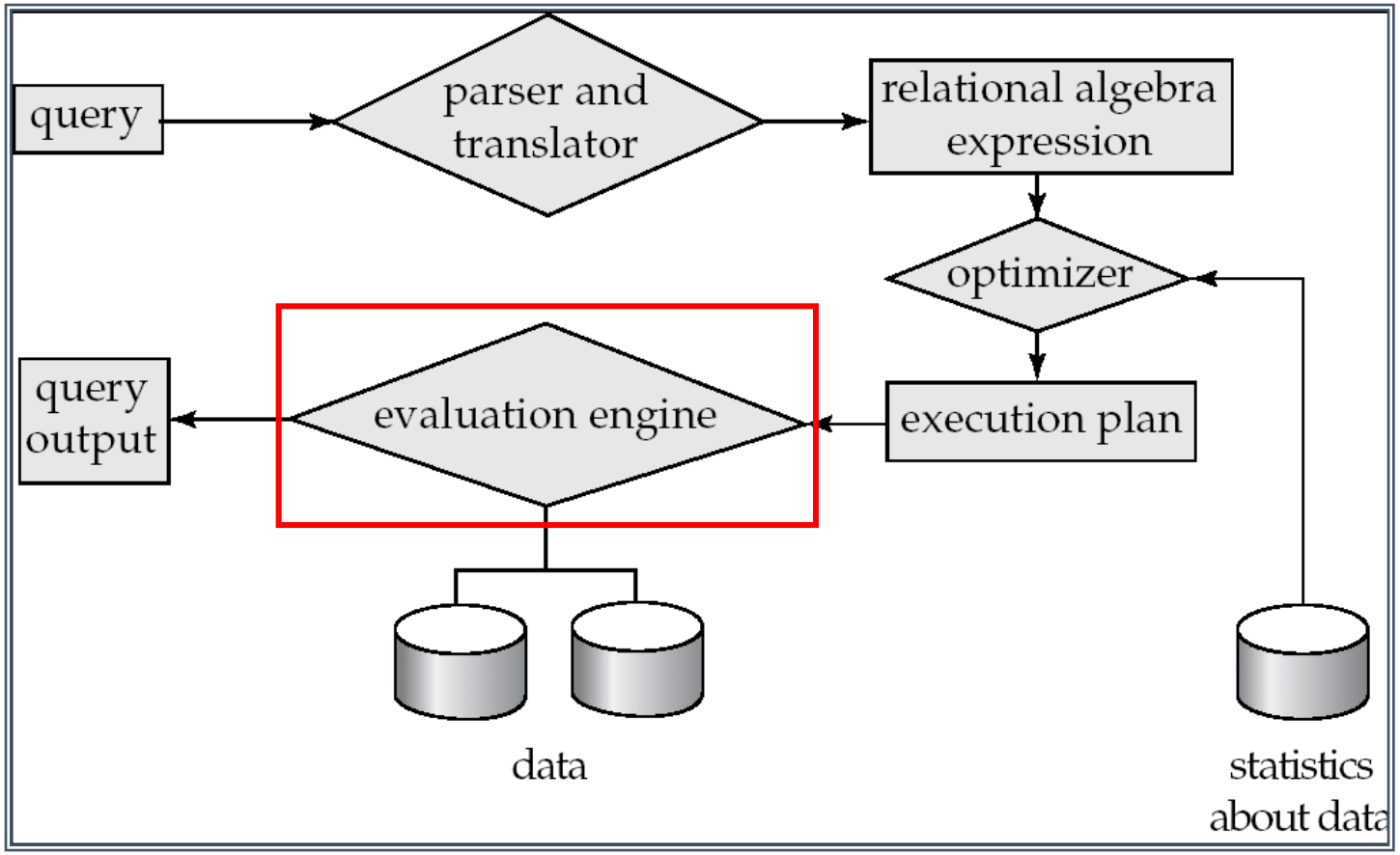


# Etapas de Processamento

- O objetivo do otimizador é gerar um plano de execução para cada consulta

- O plano informa exatamente como a consulta será respondida
  - Ou seja, quais algoritmos / operações / estruturas de dados serão usados

# Etapas de Processamento



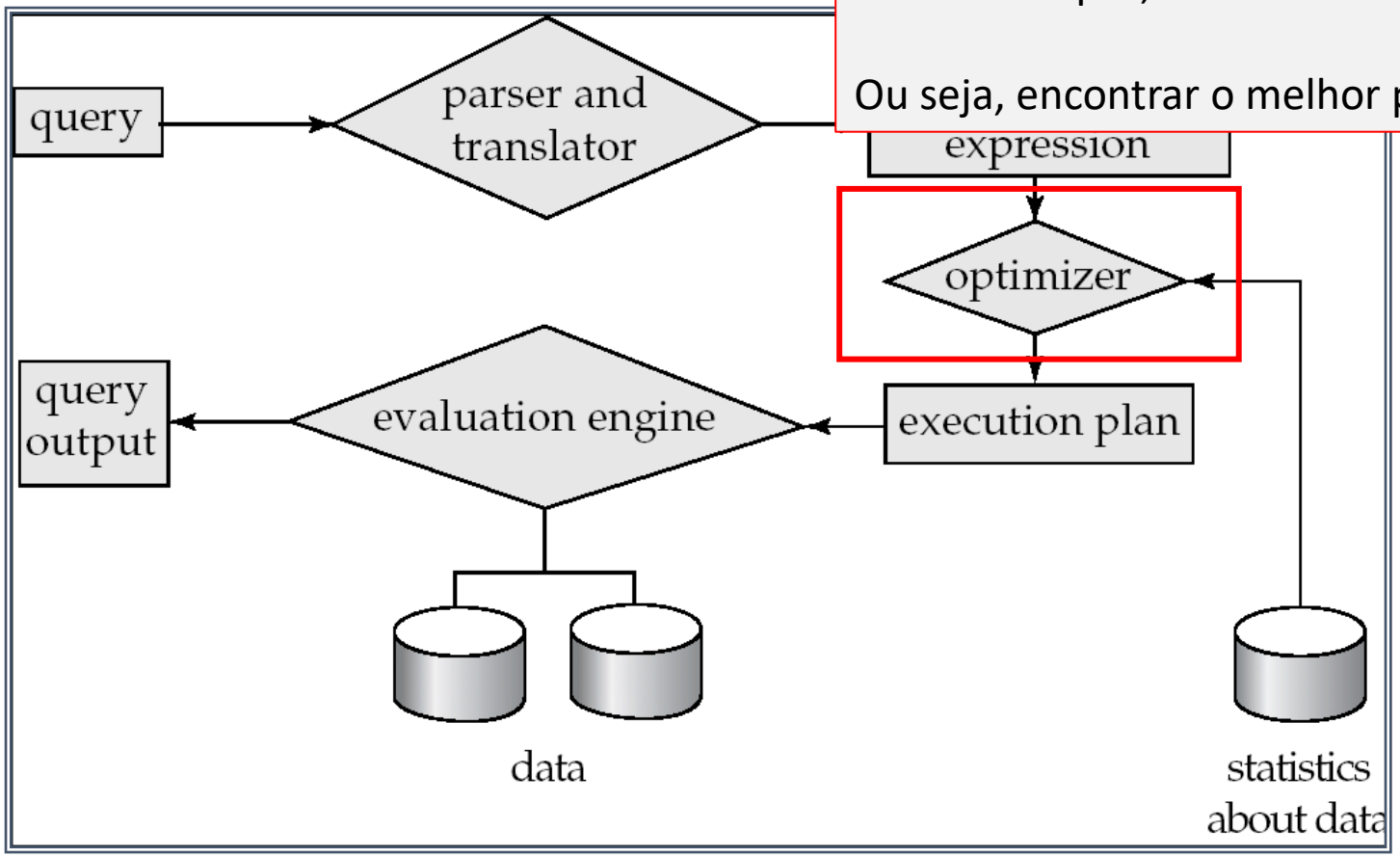
# Etapas de Processamento

- O motor de execução de consultas
  - Executa o plano escolhido
  - Retorna as resposta da consulta.

# Etapas de Processamento

Das três etapas, o maior desafio está na Otimização

Ou seja, encontrar o melhor plano de execução



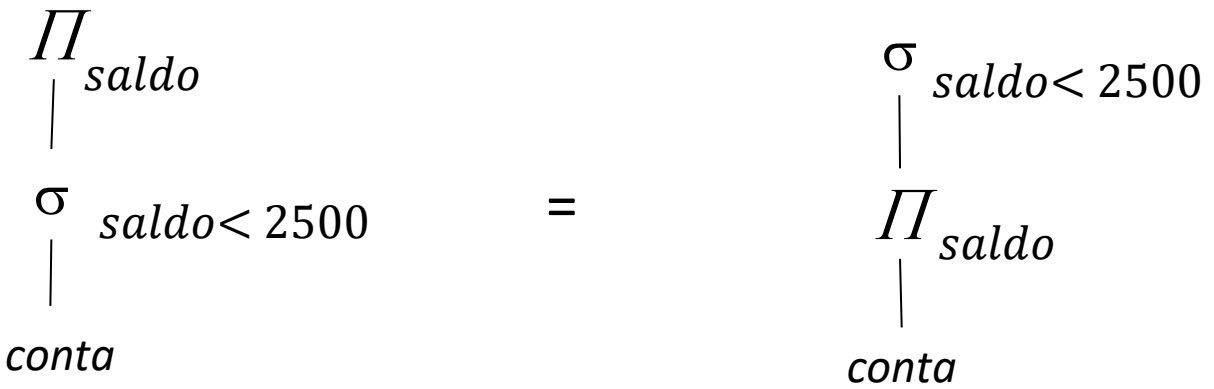
# Sumário

- Etapas no processamento de uma consulta SQL
  - Otimização de consultas
- Plano de execução
  - Fluxo de Execução
  - Read all
  - Read some

# Otimização de consultas

- Existe mais de um caminho até a resposta
  - Por exemplo, uma expressão em álgebra relacional pode possuir várias representações equivalentes

- Ex.



# Otimização de consultas

- Além disso, cada operação da álgebra pode ser executada usando diversos algoritmos diferentes
  - Assim, uma mesma expressão pode ser executada de diversas formas.

- Ex., para a expressão abaixo, pode-se
  - usar um índice sobre saldo para encontrar contas com saldo < 2500
  - Ou realizar uma varredura completa na tabela e descartar contas com saldo ≥ 2500

$$\Pi_{saldo}$$
$$|$$
$$\sigma_{saldo < 2500}$$
$$|$$
$$conta$$

# Otimização de consultas

- Vários planos diferentes podem levar ao mesmo resultado
  - Esses planos são chamados de equivalentes
- Dentre todos os planos equivalentes
  - Cabe ao SGBD escolher aquele que possui o menor custo.
- O custo é estimado usando informações estatísticas do catálogo do banco de dados
  - ex. Número de tuplas de cada relação, tamanho das tuplas, etc.

# Otimização de consultas

- O custo é geralmente medido como o tempo total gasto para responder uma consulta
  - Principais fatores
    - *Acessos a disco*
    - *CPU*

- Iremos ignorar custos de CPU, uma vez que são muito menores
  - Sistemas reais levam esse custo em consideração, mas eles são bem menos significantes

# Otimização de consultas

- O acesso ao disco é relativamente fácil de estimar.

- As estimativas consideram
  - Tempo para realizar uma busca (um seek)
  - Tempo para realizar uma transferência

# Otimização de consultas

- O custo do seek (busca)
  - é usualmente muito maior do que a transferência de página
    - No entanto, é um pouco mais difícil estimá-lo (depende da distância do cabeçote de leitura até o setor desejado
    - Além disso, esse custo não se aplica a SSDs, que realizam a busca sem recorrer à uma movimentação mecânica

- Por esses motivos, durante as aulas focaremos no custo de transferência de páginas
  - Quanto menos páginas transferidas, melhor é o plano

# Sumário

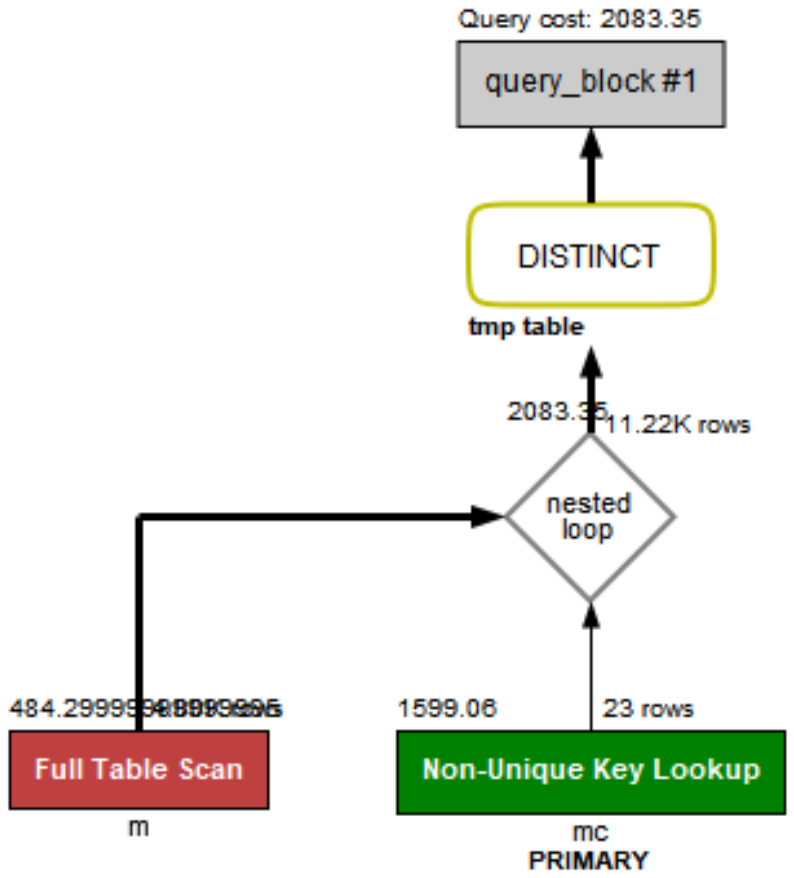
- Etapas no processamento de uma consulta SQL
  - Otimização de consultas
- Plano de execução
  - Fluxo de Execução
  - Read all
  - Read some

# Plano de Execução

- Exemplo de plano de execução em MySQL

```
select distinct title, c_name
from movie m natural join movie_cast mc
where m.release_year = 2000
```

**Plano MySQL**

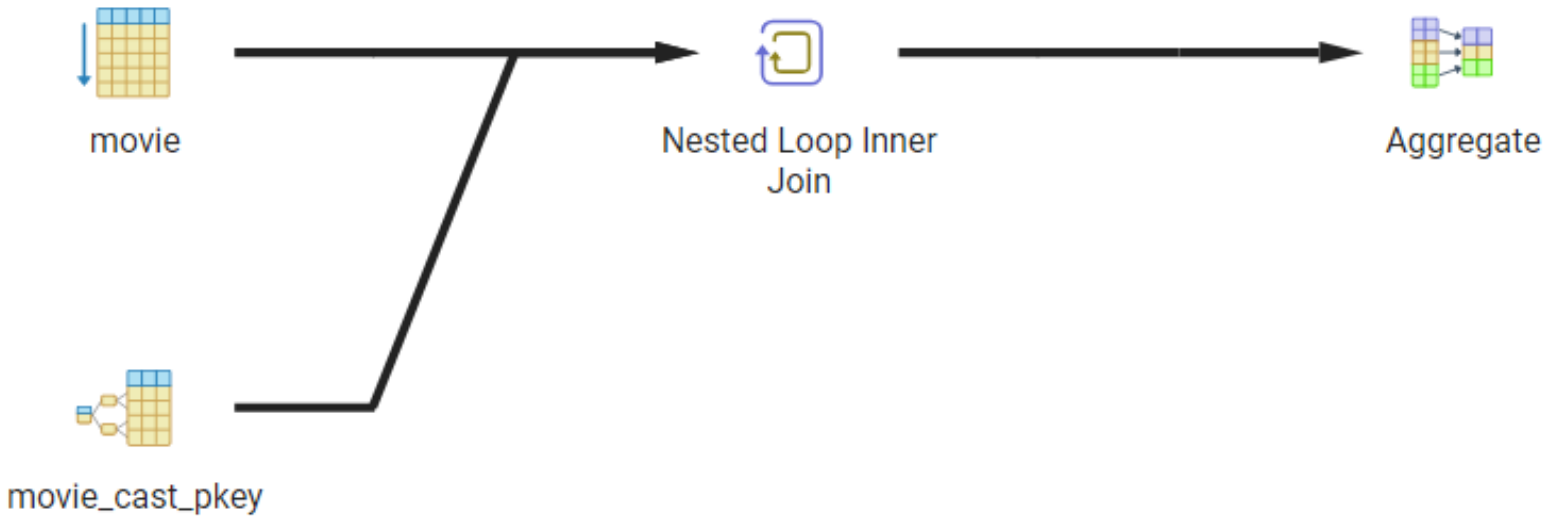


# Plano de Execução

- Exemplo de plano de execução em PostgreSQL

```
select distinct title, c_name
from movie m natural join movie_cast mc
where m.release_year = 2000
```

**Plano PostgreSQL**



# Plano de Execução

- Como vimos, cada SGBD possui uma terminologia e um formato de visualização próprios

- Neste minicurso, adotaremos o formato e terminologia usada no DBest
  - Apesar das diferenças, dá para fazer um paralelo com os planos gerados pelo MySQL e PostgreSQL
    - Todos seguem a arquitetura de execução de consultas proposta pelo modelo Volcano.

# Plano de Execução

- No DBest, o plano de execução é uma árvore composta por nós de transformação de registros
  - Esses nós são chamados de operadores

- Três tipos de nós
  - Nós unários
  - Nós binários
  - Nós fonte

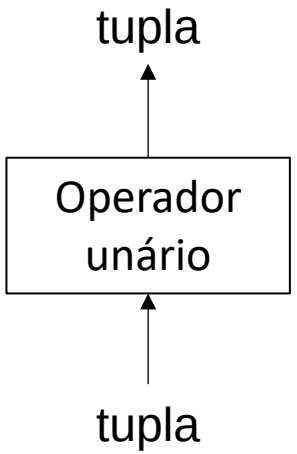
- Os nós usam tuplas como informações de entrada/saída

- Tupla: uma sequência ordenada de valores, onde cada valor corresponde a um atributo (coluna) de uma relação (tabela).

- Pode-se fazer um paralelo entre tupla e registro

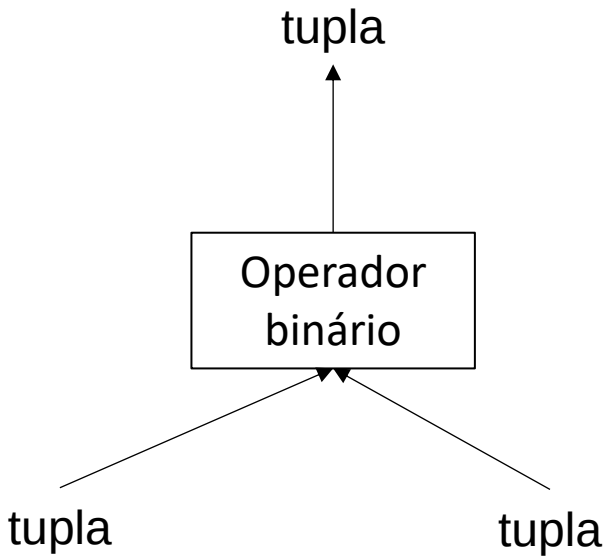
# Plano de Execução

- Operadores unários
  - Recebem uma tupla na entrada e devolvem uma tupla na saída



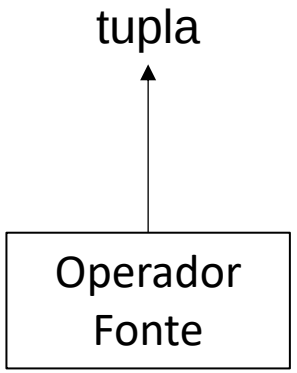
# Plano de Execução

- Operadores binários
  - Recebem duas tuplas na entrada e devolvem uma tupla na saída



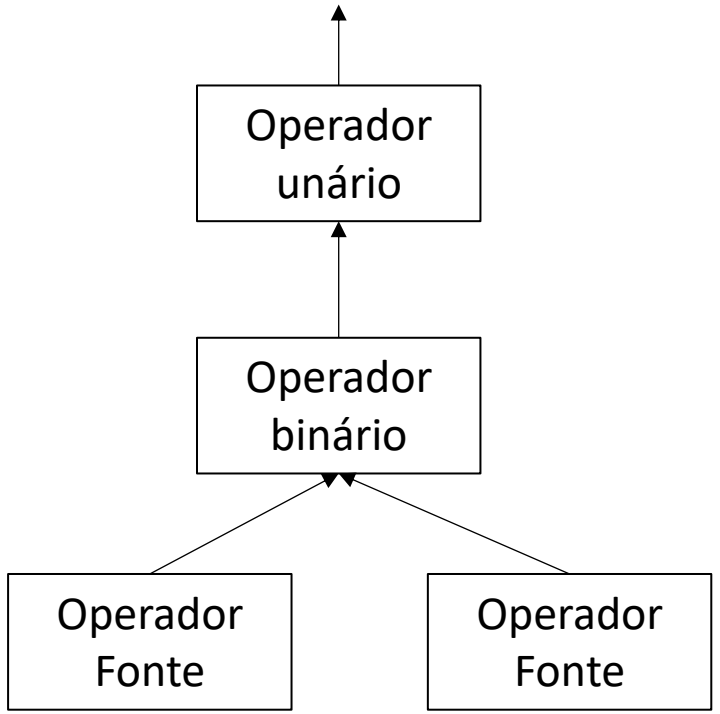
# Plano de Execução

- Operadores fonte
  - Extraem tuplas a partir de uma fonte de dados
    - Ex. arquivo heap, arquivo B+ tree



# Plano de Execução

- Nós podem ser combinados para formar fluxos de transformação de tuplas

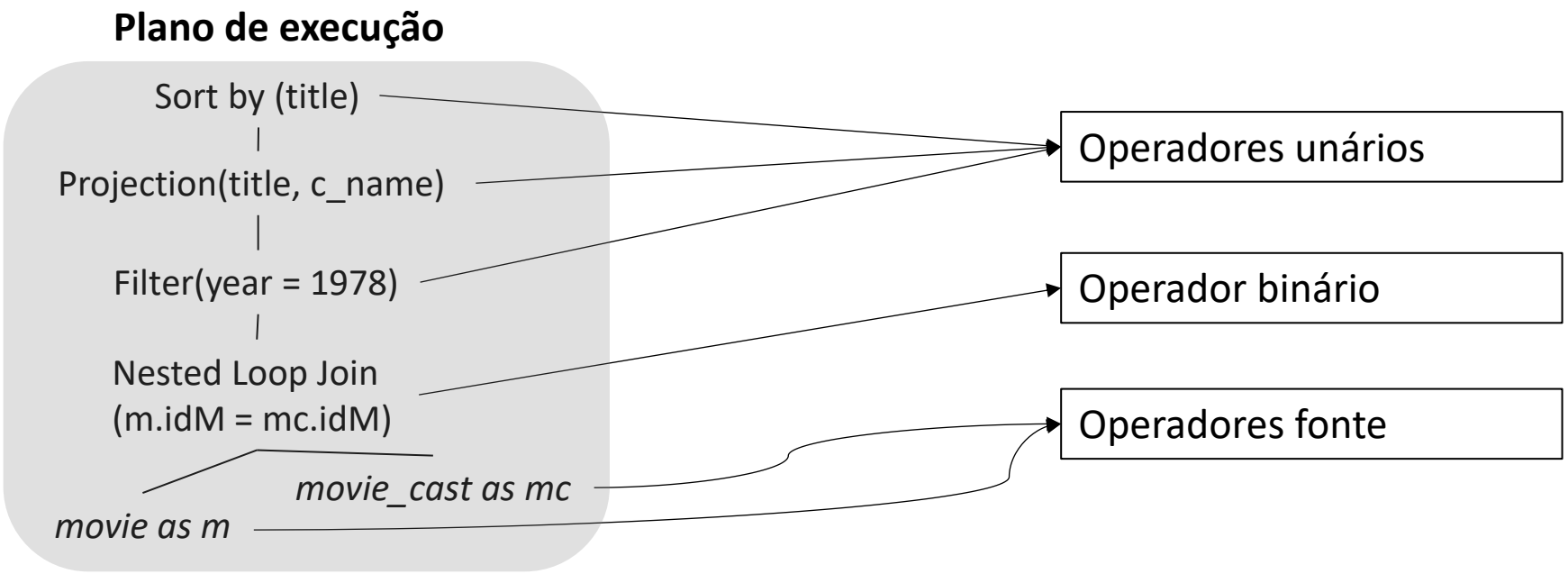


# Plano de Execução

- Operadores típicos em um plano de execução
  - Projection: indica quais colunas devem ser retornadas
  - Filter: indica quais tuplas devem ser retornadas
  - Join: indica que tuplas relacionadas de duas tabelas devem ser combinadas
  - Aggregation: indica que tuplas devem ser agrupadas e valores sintetizados devem ser calculados
  - …

# Plano de Execução

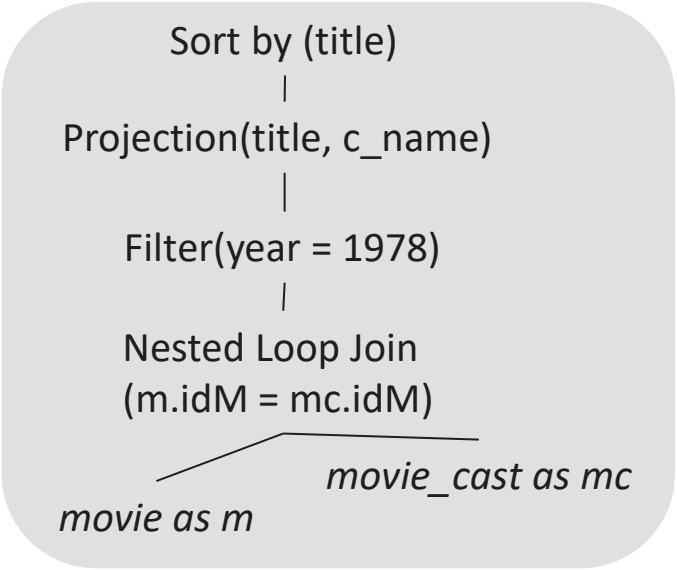
- O exemplo abaixo apresenta um plano de execução concreto usando operadores



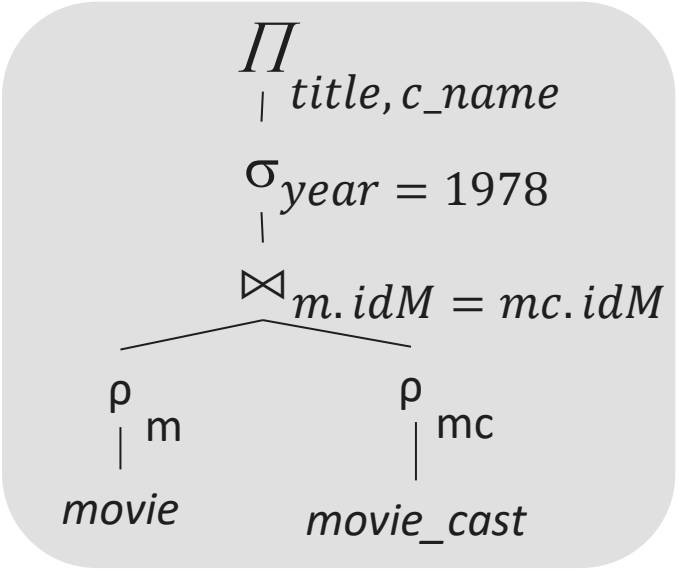
# Plano de Execução

- Um plano de execução possui associações com a álgebra relacional
  - Mas não são a mesma coisa

**Plano de execução**



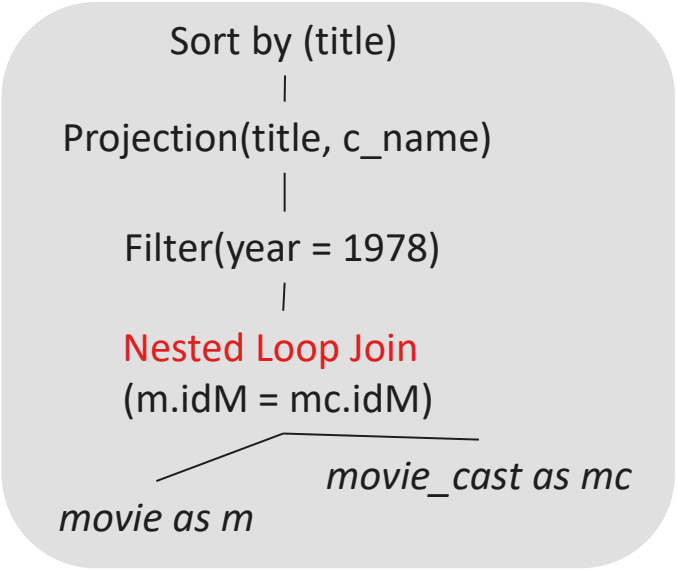

**Expressão em álgebra relacional**



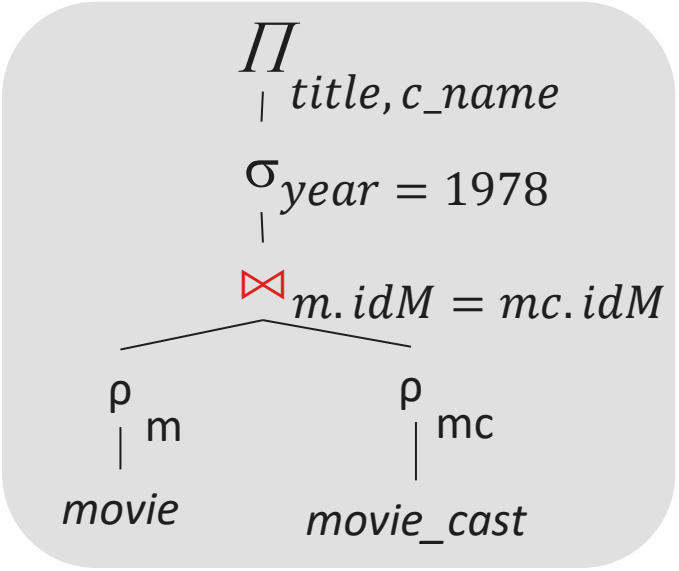
# Plano de Execução

- Diferença 1
  - A álgebra relacional define apenas as operações
  - Um plano de execução define operadores (algoritmos para cada operação)
    - Ex. Nested Loop Join

**Plano de execução**



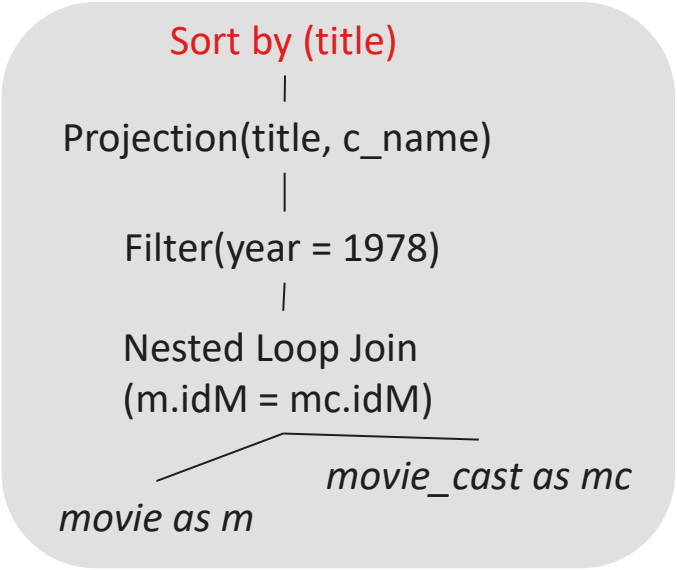

**Expressão em álgebra relacional**



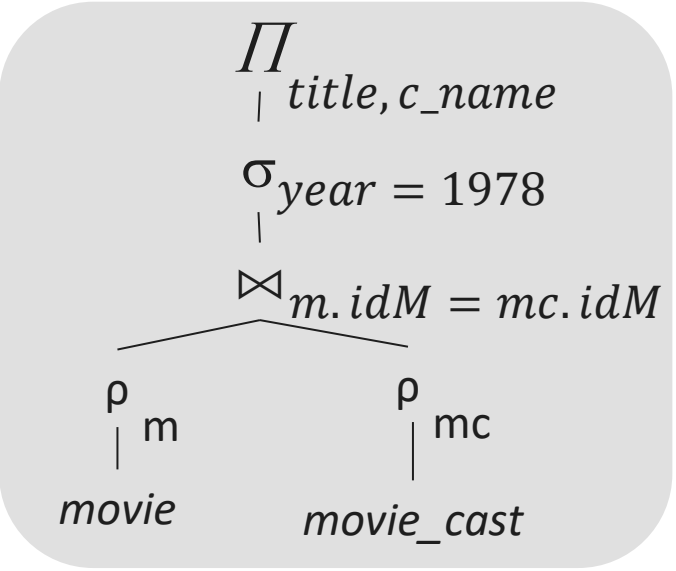
# Plano de Execução

- Diferença 2
  - A álgebra relacional não possui o conceito de ordenação
  - Um plano de execução possui operadores para ordenação

**Plano de execução**

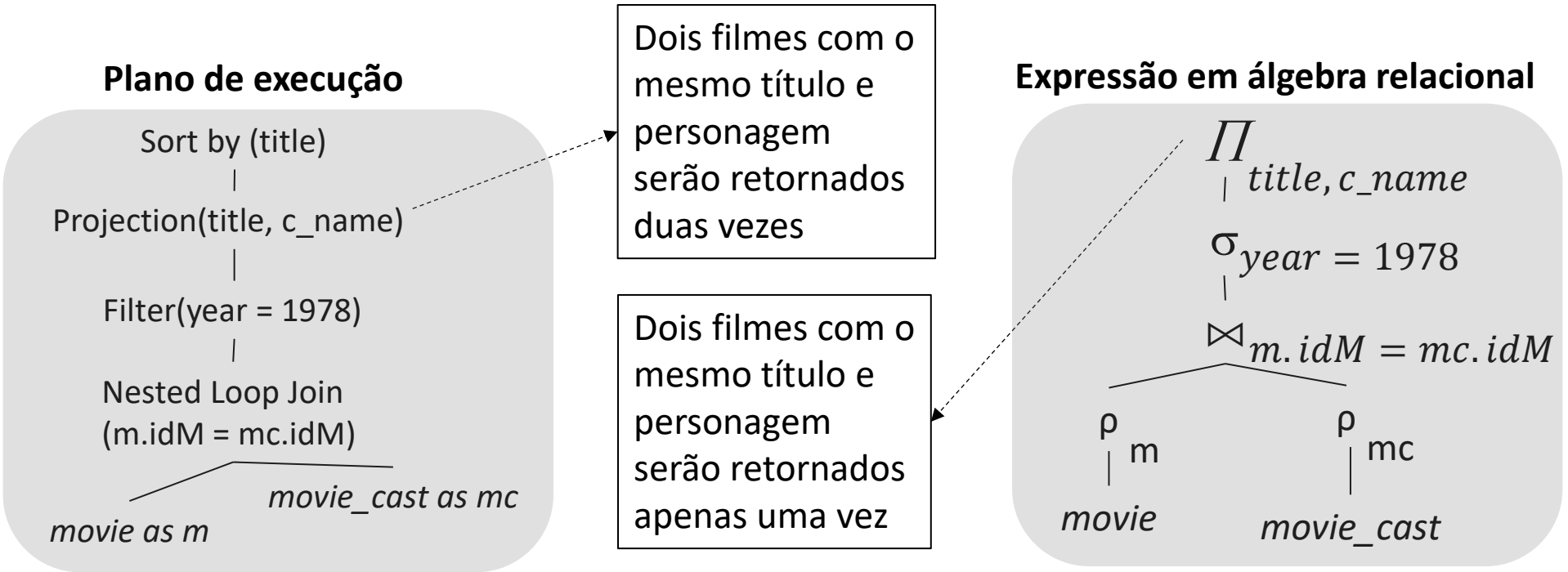



**Expressão em álgebra relacional**



# Plano de Execução

- Diferença 3
  - A álgebra relacional não possui o conceito de registros duplicados
  - Um plano de execução permite que registros duplicados sejam retornados



# Sumário

- Etapas no processamento de uma consulta SQL
  - Otimização de consultas
- Plano de execução
  - Fluxo de Execução
  - Read all
  - Read some

# Fluxo de Execução

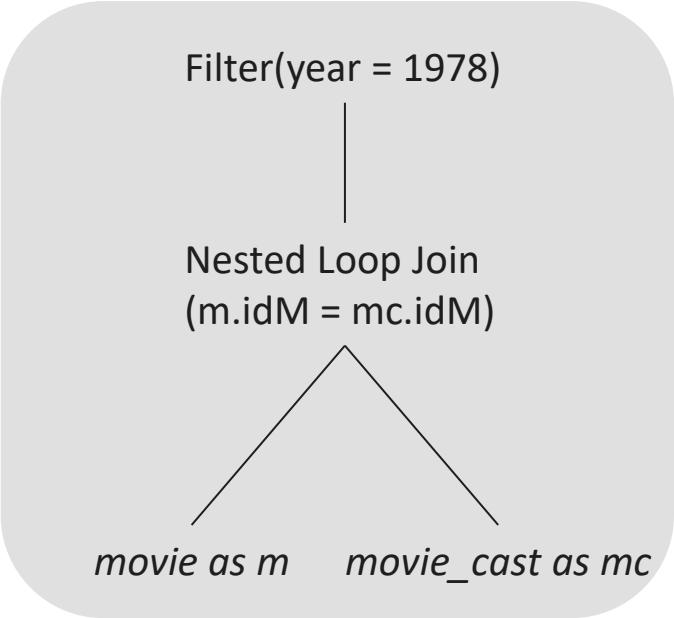
- O fluxo de dados segue o modelo Volcano

- Nesse modelo, um operador se conecta a outro por meio de um iterador
  - open(): a conexão é aberta
  - next(): o próximo registro é solicitado
  - close(): a conexão é fechada

- O modelo também é chamado de iterator-based, ou pull-based
  - Porque os registros são puxados (pulled) um de cada vez, conforme forem requisitados

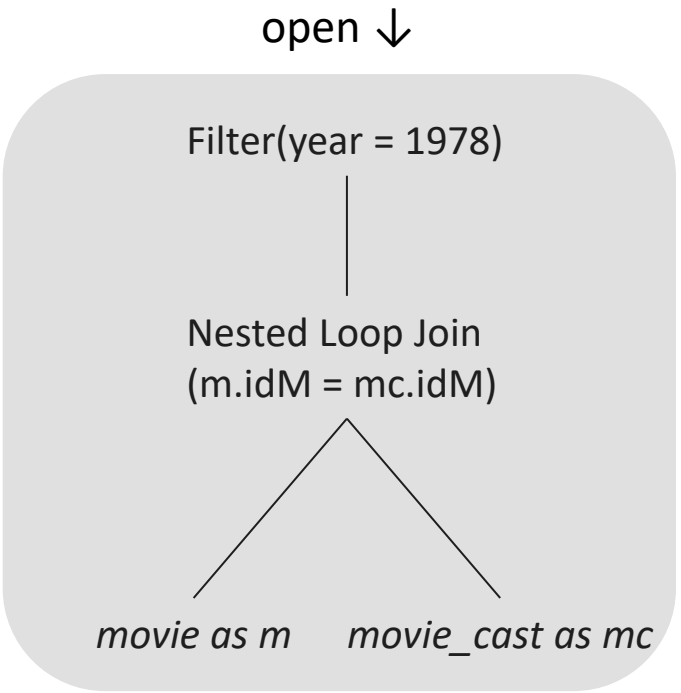
# Fluxo de Execução

- open/close: a abertura/fechamento de um operador é propagada para todos os nós a ele conectados



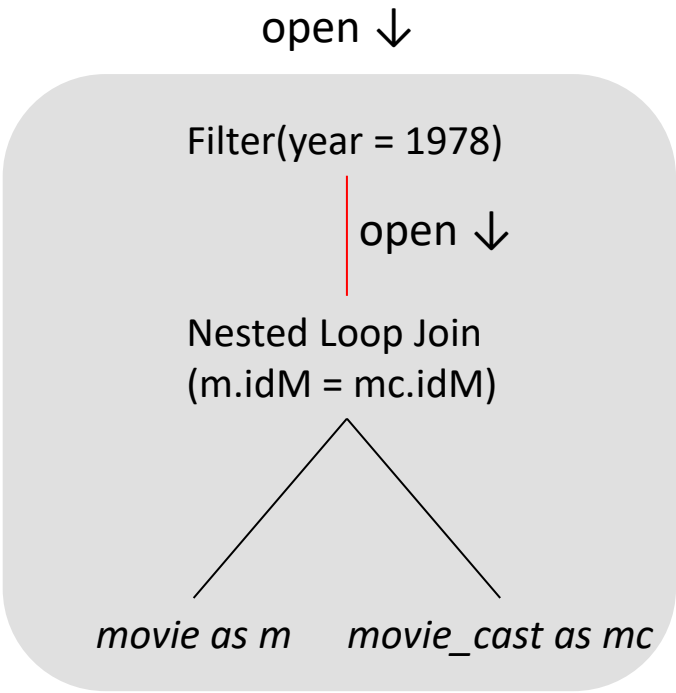
# Fluxo de Execução

- open/close: a abertura/fechamento de um operador é propagada para todos os nós a ele conectados



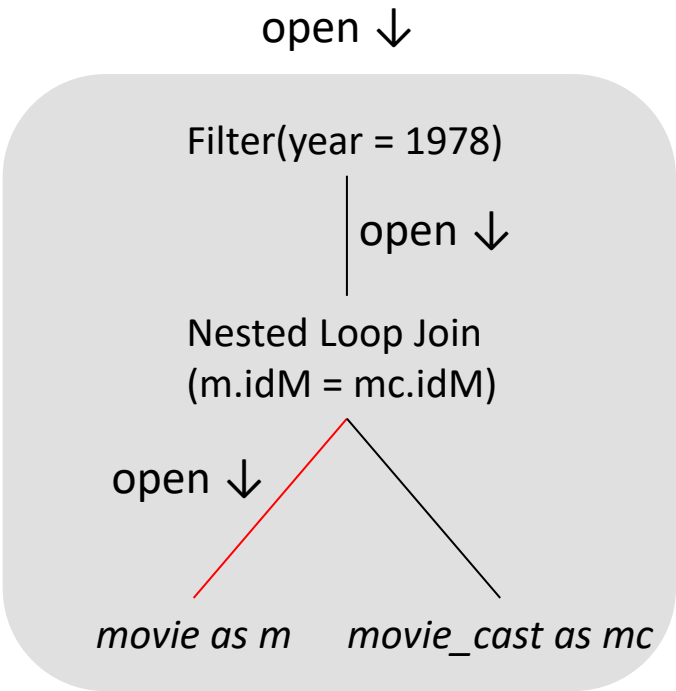
# Fluxo de Execução

- open/close: a abertura/fechamento de um operador é propagada para todos os nós a ele conectados



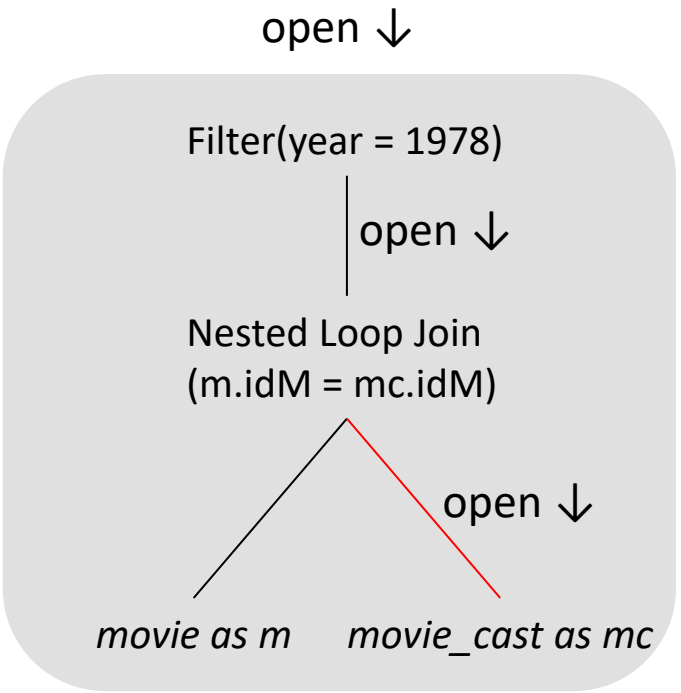
# Fluxo de Execução

- open/close: a abertura/fechamento de um operador é propagada para todos os nós a ele conectados



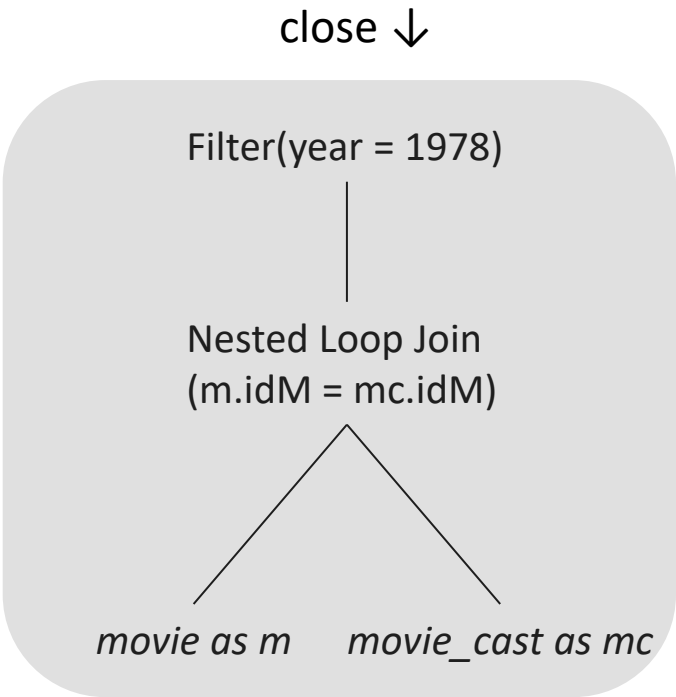
# Fluxo de Execução

- open/close: a abertura/fechamento de um operador é propagada para todos os nós a ele conectados



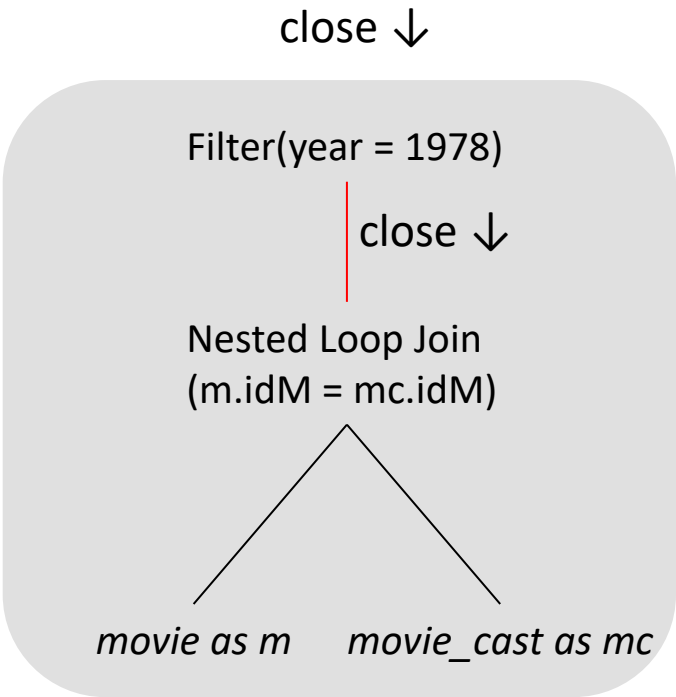
# Fluxo de Execução

- open/close: a abertura/fechamento de um operador é propagada para todos os nós a ele conectados



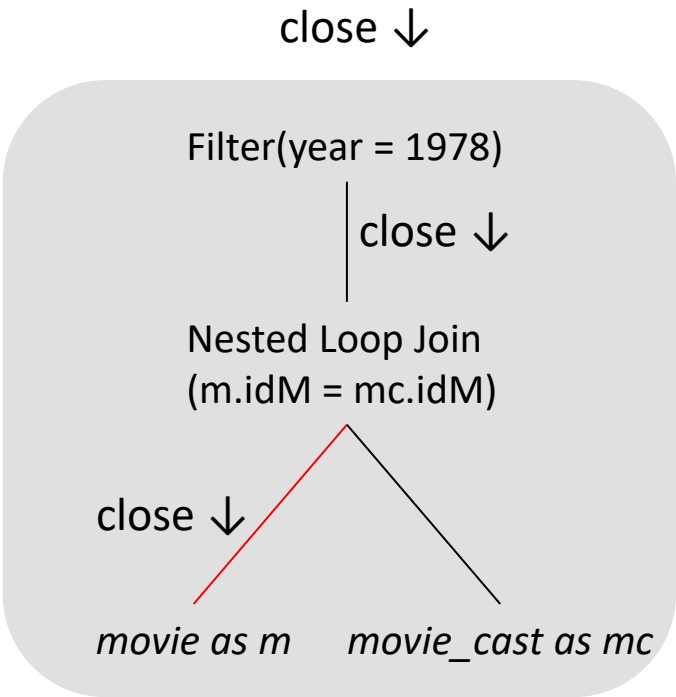
# Fluxo de Execução

- open/close: a abertura/fechamento de um operador é propagada para todos os nós a ele conectados



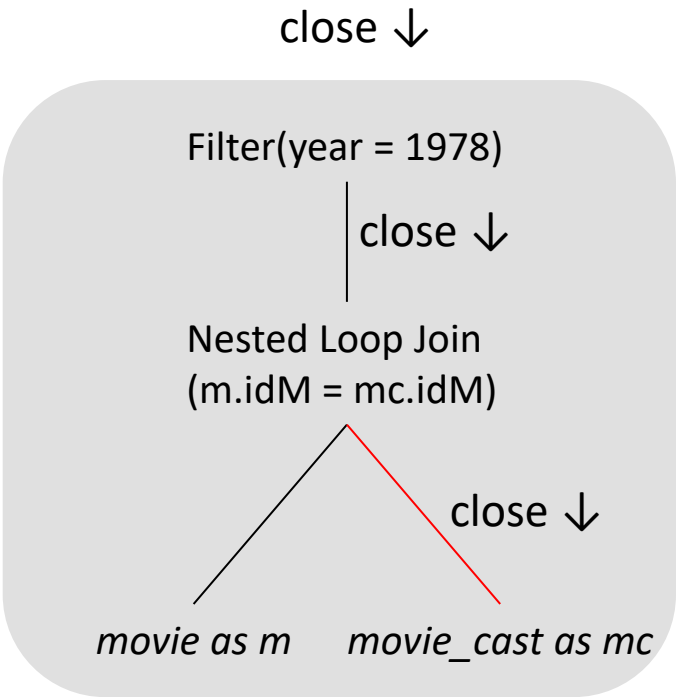
# Fluxo de Execução

- open/close: a abertura/fechamento de um operador é propagada para todos os nós a ele conectados



# Fluxo de Execução

- open/close: a abertura/fechamento de um operador é propagada para todos os nós a ele conectados

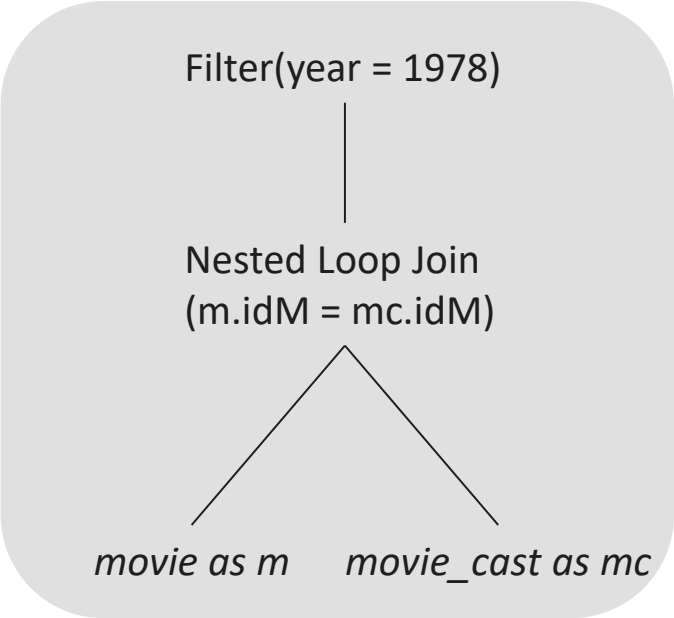


# Fluxo de Execução

- next: inicia em um nó e é propagado aos demais nós conectados

- O next retorna uma tupla
  - em alguns casos, o ponteiro do operador é avançado para que a próxima tupla seja retornada depois

- Essa propagação de tuplas pelos operadores, geradas uma de cada vez, é chamada de execução por pipeline

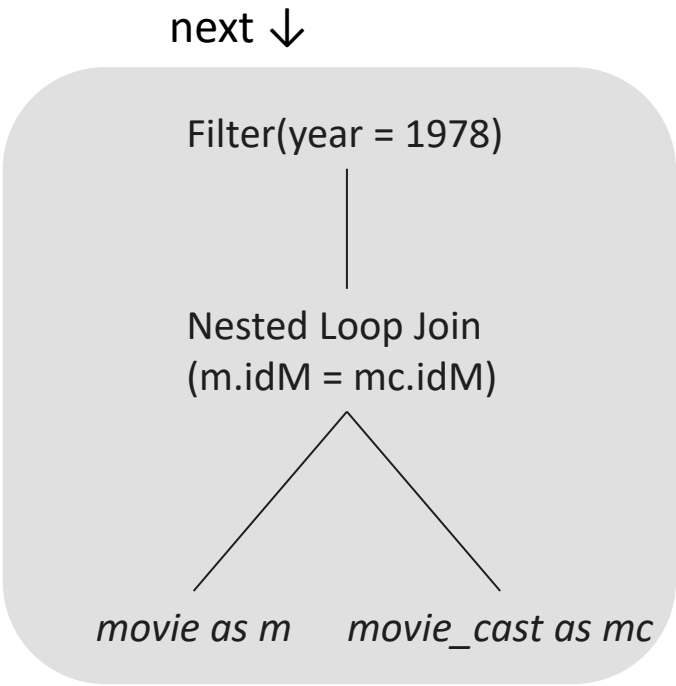


# Fluxo de Execução

- next: inicia em um nó e é propagado aos demais nós conectados

- O next retorna uma tupla
  - em alguns casos, o ponteiro do operador é avançado para que a próxima tupla seja retornada depois

- Essa propagação de tuplas pelos operadores, geradas uma de cada vez, é chamada de execução por pipeline

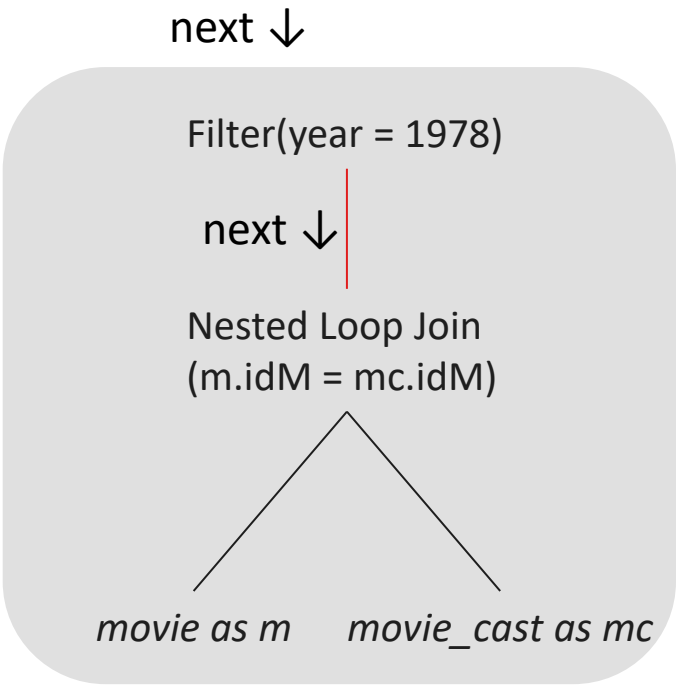


# Fluxo de Execução

- next: inicia em um nó e é propagado aos demais nós conectados

- O next retorna uma tupla
  - em alguns casos, o ponteiro do operador é avançado para que a próxima tupla seja retornada depois

- Essa propagação de tuplas pelos operadores, geradas uma de cada vez, é chamada de execução por pipeline

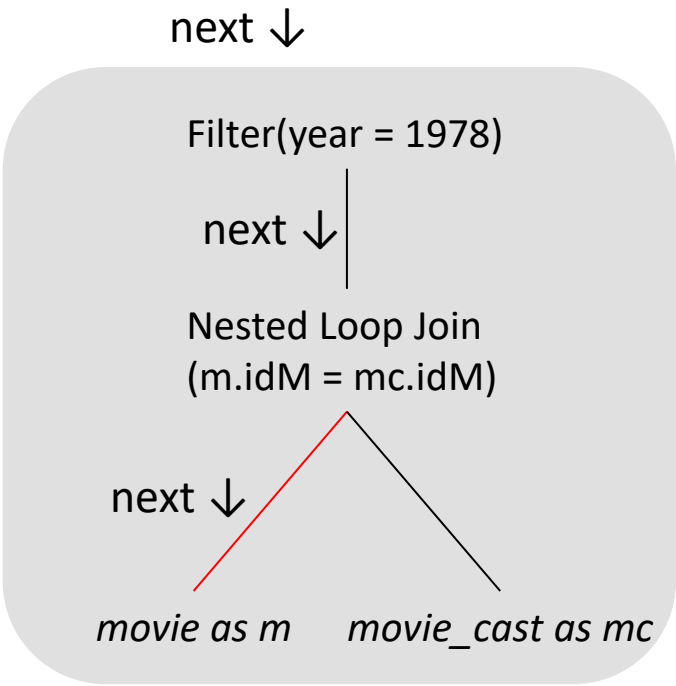


# Fluxo de Execução

- next: inicia em um nó e é propagado aos demais nós conectados

- O next retorna uma tupla
  - em alguns casos, o ponteiro do operador é avançado para que a próxima tupla seja retornada depois

- Essa propagação de tuplas pelos operadores, geradas uma de cada vez, é chamada de execução por pipeline

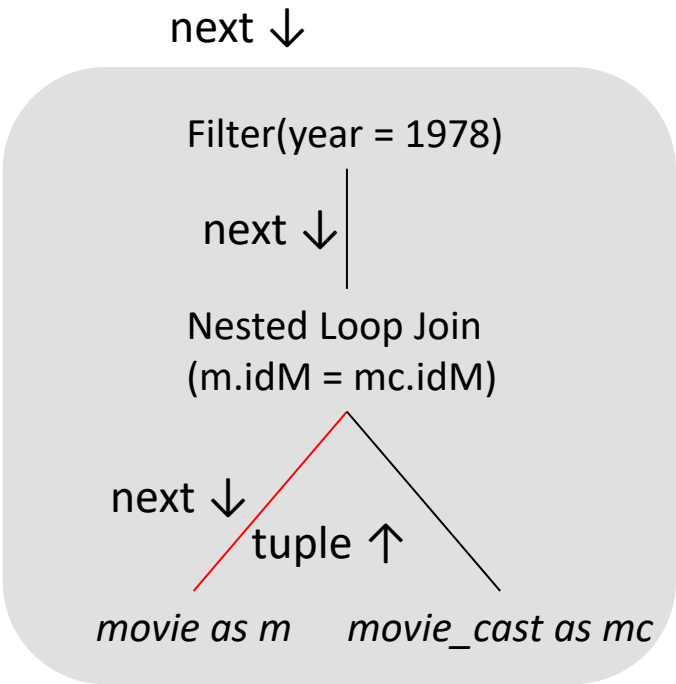


# Fluxo de Execução

- next: inicia em um nó e é propagado aos demais nós conectados

- O next retorna uma tupla
  - em alguns casos, o ponteiro do operador é avançado para que a próxima tupla seja retornada depois

- Essa propagação de tuplas pelos operadores, geradas uma de cada vez, é chamada de execução por pipeline

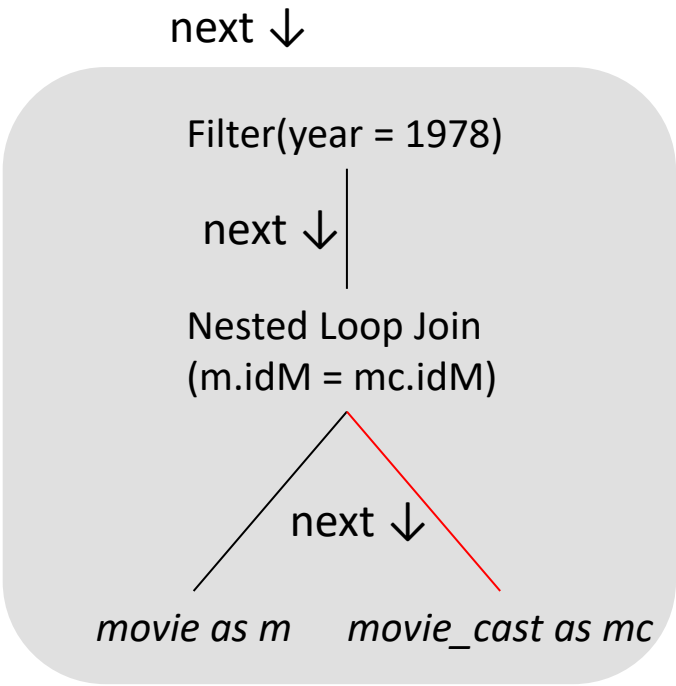


# Fluxo de Execução

- next: inicia em um nó e é propagado aos demais nós conectados

- O next retorna uma tupla
  - em alguns casos, o ponteiro do operador é avançado para que a próxima tupla seja retornada depois

- Essa propagação de tuplas pelos operadores, geradas uma de cada vez, é chamada de execução por pipeline

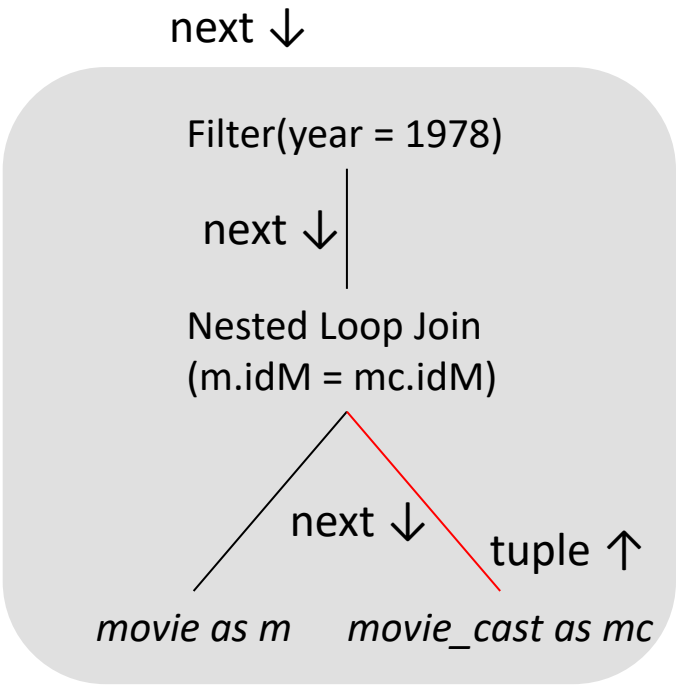


# Fluxo de Execução

- next: inicia em um nó e é propagado aos demais nós conectados

- O next retorna uma tupla
  - em alguns casos, o ponteiro do operador é avançado para que a próxima tupla seja retornada depois

- Essa propagação de tuplas pelos operadores, geradas uma de cada vez, é chamada de execução por pipeline

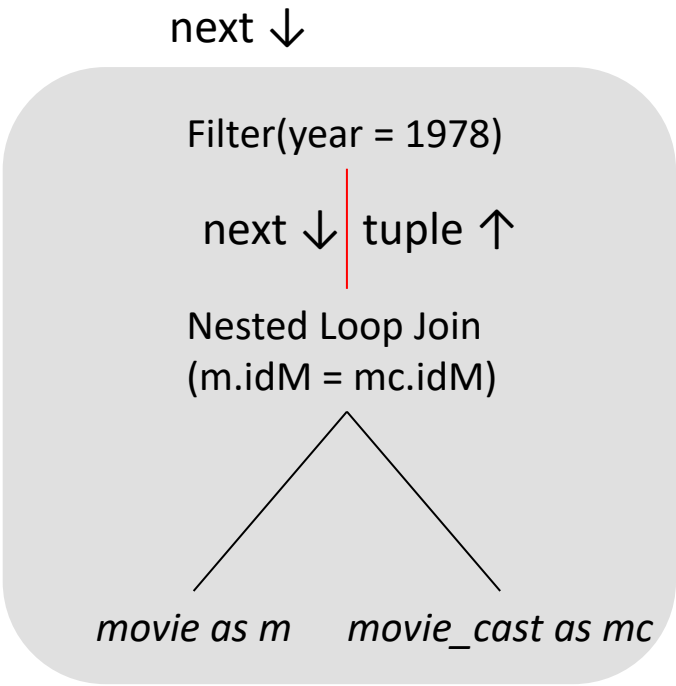


# Fluxo de Execução

- next: inicia em um nó e é propagado aos demais nós conectados

- O next retorna uma tupla
  - em alguns casos, o ponteiro do operador é avançado para que a próxima tupla seja retornada depois

- Essa propagação de tuplas pelos operadores, geradas uma de cada vez, é chamada de execução por pipeline

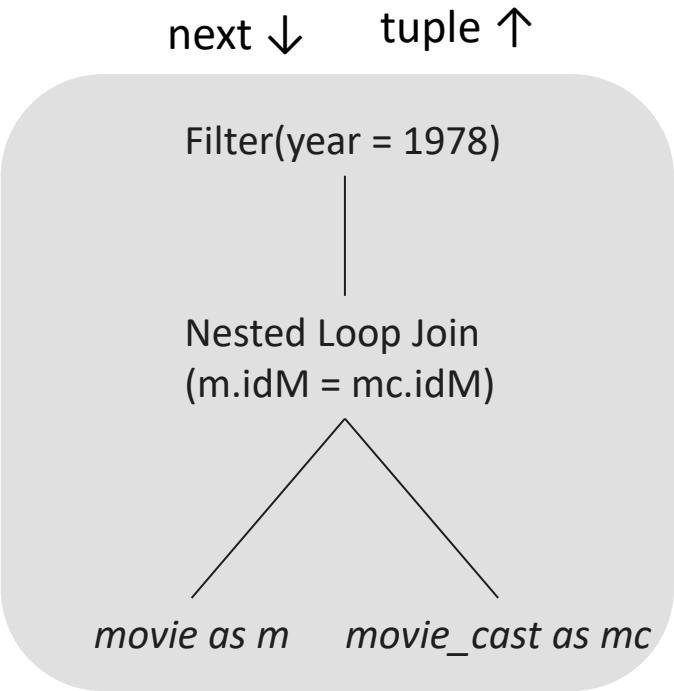


# Fluxo de Execução

- next: inicia em um nó e é propagado aos demais nós conectados

- O next retorna uma tupla
  - em alguns casos, o ponteiro do operador é avançado para que a próxima tupla seja retornada depois

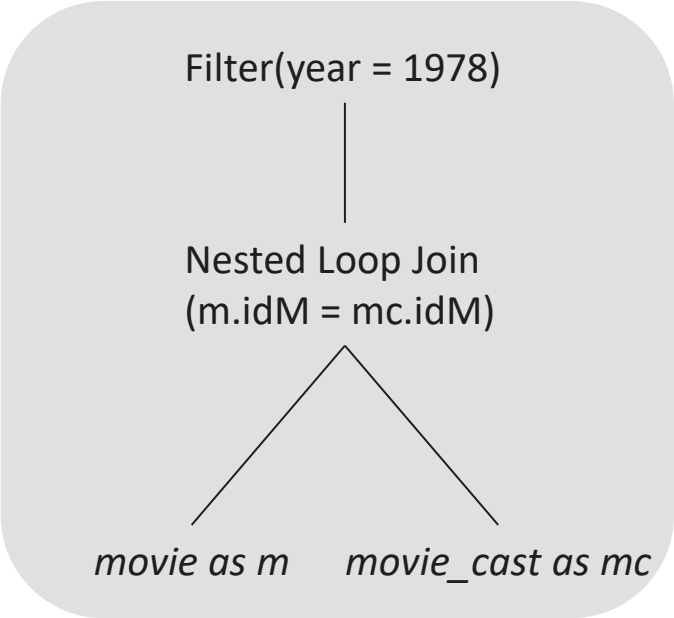
- Essa propagação de tuplas pelos operadores, geradas uma de cada vez, é chamada de execução por pipeline



# Fluxo de Execução

- Nem sempre uma chamada next acarreta em uma chamada next no nível abaixo

- Ex. No operador Nested Loop Join, uma chamada next só avança o ponteiro da esquerda  (movie) quando tods as correspondências(movie_casts) de movie atual forem processadas



# Fluxo de Execução

- Operadores podem ser do tipo materializados
  - Ex. o operador hash

- São chamados assim porque mantém uma cópia de todas as tuplas que chegam até ele
  - Essa cópia acarreta em consumo de memória

- Operadores materializados são bloqueantes
  - Ao chegar a primeira chamada do tipo next(), eles devem ler todas as tuplas do operador com quem estiverem conectados
  - Só então a chamada next() é atendida

  - Ou seja, parte da execução não ocorre dentro do pipeline

# Fluxo de Execução

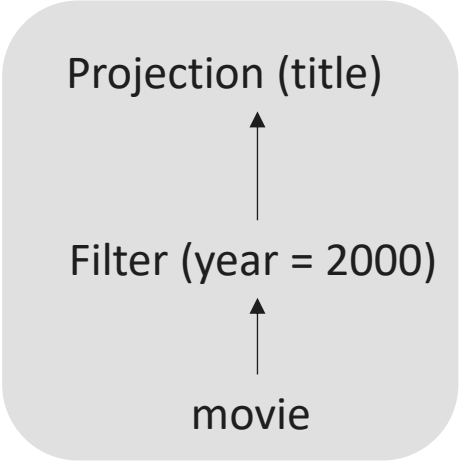
- Operadores podem ser indexados

- São chamados assim porque eles conseguem resolver filtros de forma eficiente sobre uma chave de busca específica

- Exemplos típicos são os nós fonte que acessam árvores B+
  - Mas nós intermediários também podem ser usados como índices
    - Ex. o operador Hash

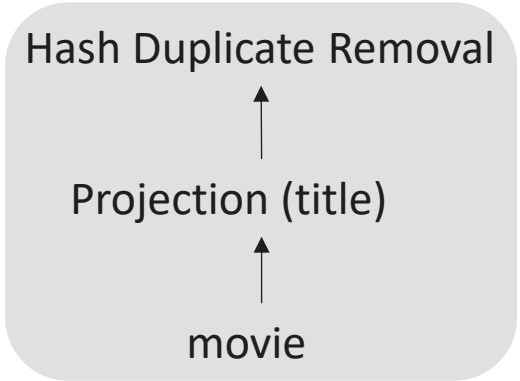
# Fluxo de Execução

- Exemplo 1
  - Toda a execução acontece dentro do pipeline
    - Quando o operador Projection recebe um next(), ele propaga a chamada para o Filter
    - Por sua vez, o Filter dispara uma chamada next() para o nó movie
    - Os resultados são gerados um de cada vez



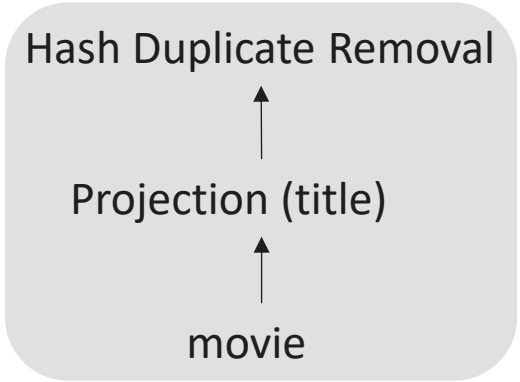
# Fluxo de Execução

- Exemplo 2
  - O operador Hash Duplicate Removal usa uma tabela hash para remover as duplicatas
    - Ou seja, é um operador materializado



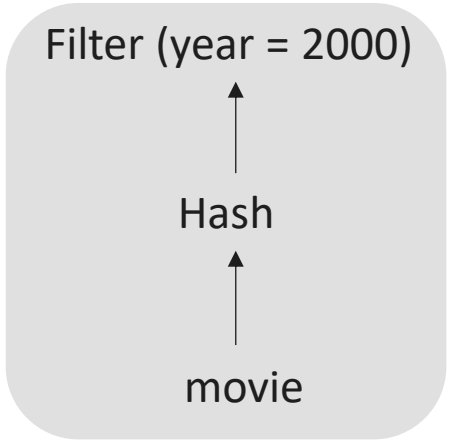
# Fluxo de Execução

- Exemplo 2
  - Operadores materializados são bloqueantes
    - Quando o operador Hash Duplicate Removal recebe um next()
      - Ele precisa chamar next() para todos os resultados encontrados pelo operador Projection
    - Ou seja, a execução não segue o fluxo de pipeline: a primeira tupla de resposta só é gerada depois que todas as tuplas estiverem disponíveis



# Fluxo de Execução

- Exemplo 3
  - O operador Hash é materializado e indexado
  - Ele gera, em tempo de execução, uma estrutura de dados para atender ao filtro que vem do operador de cima (filtro sobre year)
  - Por ser materializado, esse operador é bloqueante



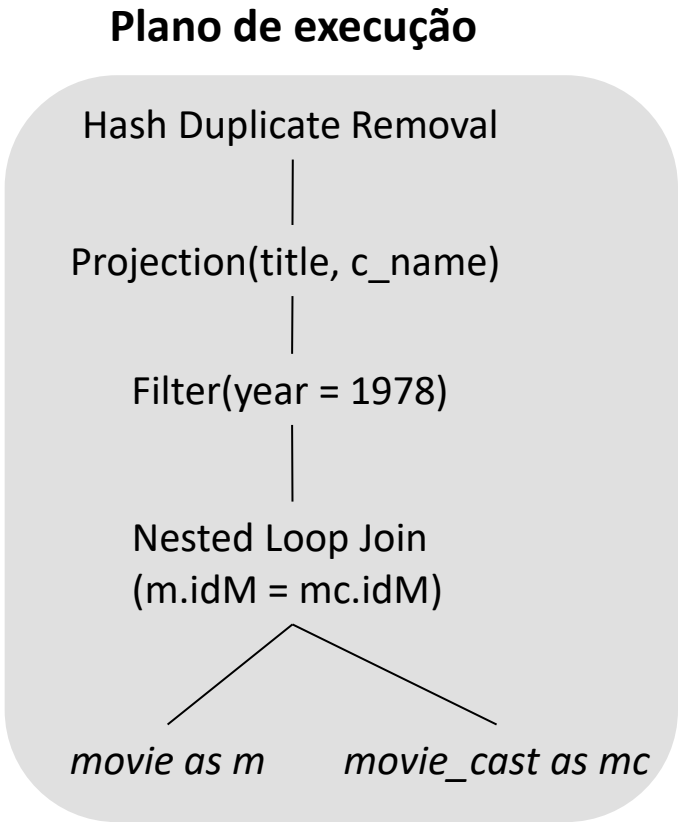
# Fluxo de Execução

- Vimos que um operador solicita tuplas por meio de uma chamada do tipo next()
  - Mas quais tuplas o next() devolve?

- Antes de fazer chamadas next(), um operador deve indicar que tipo de solicitação será feita
  - Duas possibilidades
    - read_all: o operador quer receber todas as tuplas
    - read_some: o operador quer receber apenas as tuplas que satisfaçam algum filtro, e o operador conectado consegue resolvê-lo

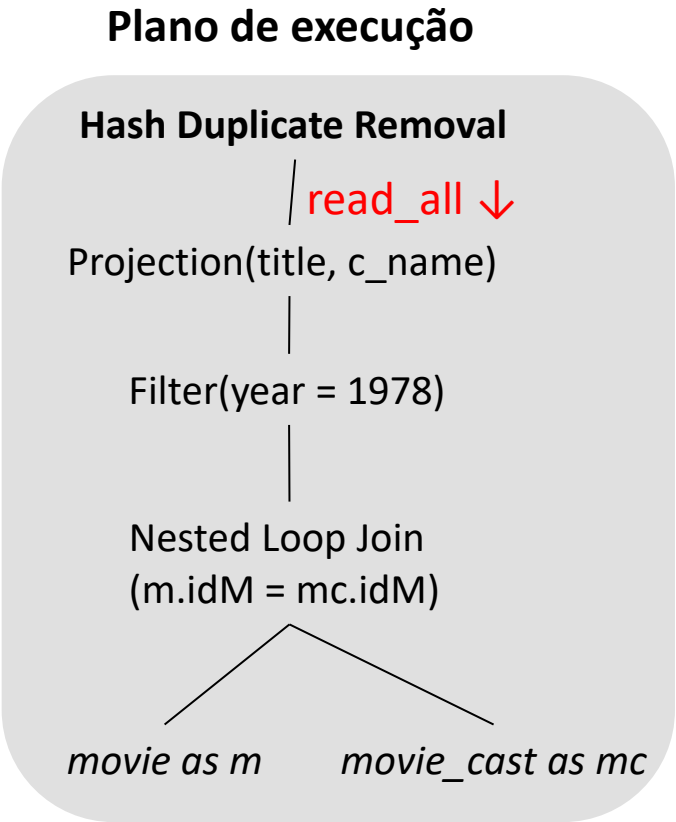
# Fluxo de Execução

- A árvore ao lado mostra o tipo de solicitação que cada operador utiliza



# Fluxo de Execução

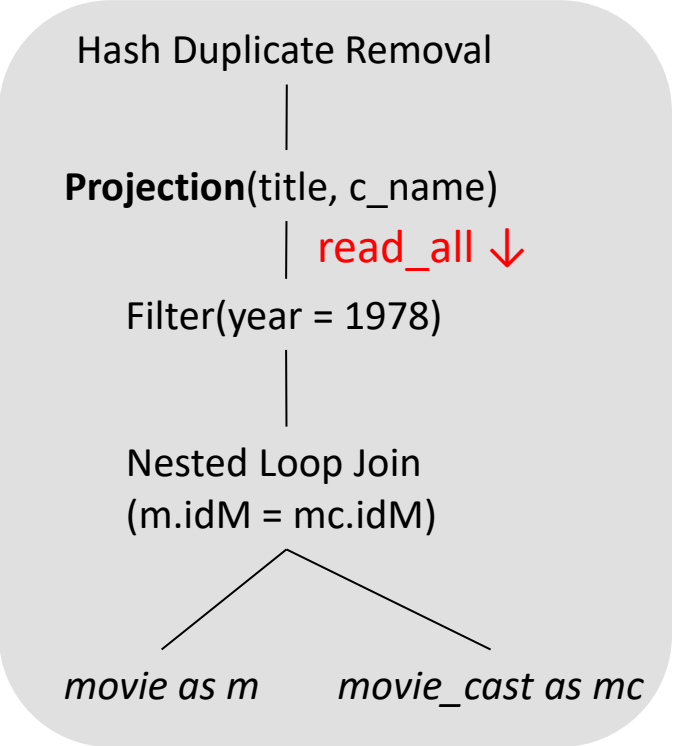
- Exemplos de operadores que utilizam read_all
  - O operador Hash Duplicate Removal
    - Precisa ler todas as tuplas para remover duplicatas
    - É um operador materializado: as tuplas devem ser todas lidas antes de algum retorno ser gerado



# Fluxo de Execução

- Exemplos de operadores que utilizam read_all
  - O operador Projection
    - Precisa ler todas as tuplas para aplicar a remoção de colunas a cada um deles
    - As tuplas são lidas um de cada vez, dentro do pipeline

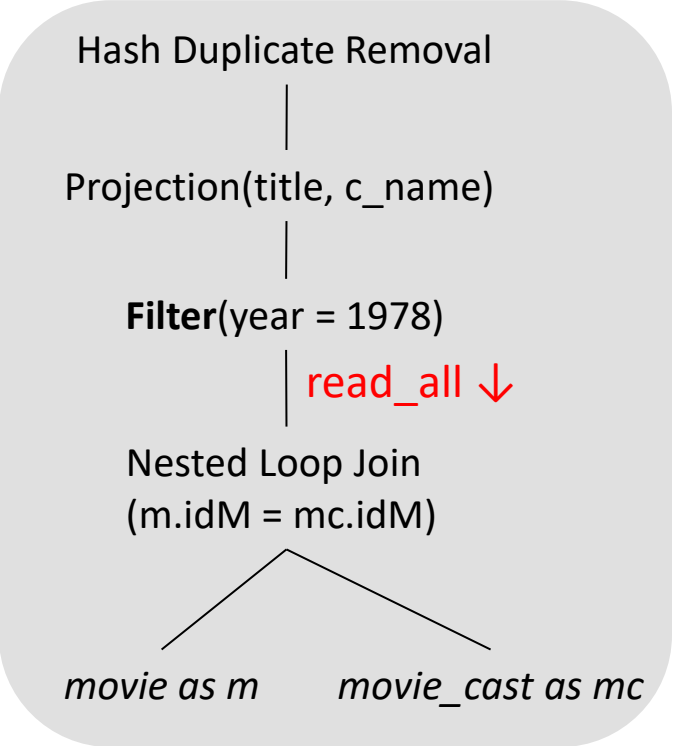
**Plano de execução**



# Fluxo de Execução

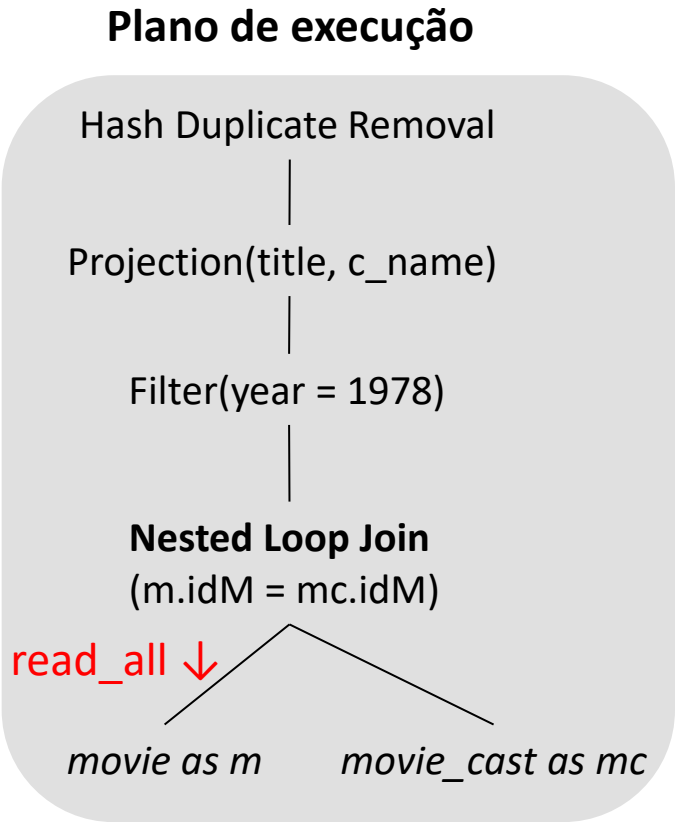
- Exemplos de operadores que utilizam read_all
  - O operador Filter
    - Precisa de apenas algumas tuplas
    - Poderia usar o read_some, se o operador a ele conectado conseguisse resolver filtros
    - Por isso, ele lê todas as tuplas e resolve o filtro internamente
    - As tuplas são lidas uma de cada vez, dentro do pipeline

**Plano de execução**



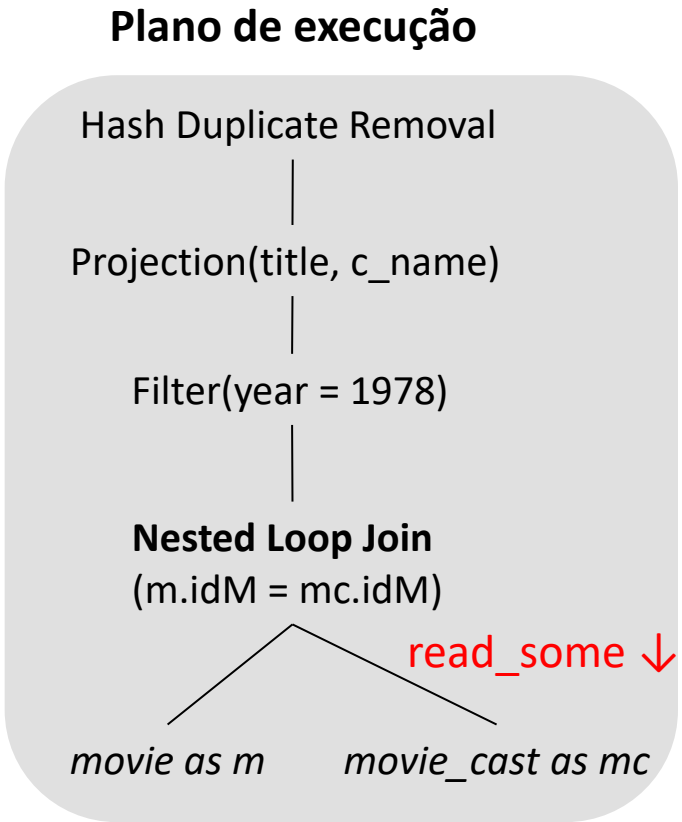
# Fluxo de Execução

- Exemplos de operadores que utilizam read_all
  - O operador Nested Loop Join
    - Precisa ler todas as tuplas que vem do lado esquerdo da junção
    - As tuplas são lidas uma de cada vez, dentro do pipeline



# Fluxo de Execução

- Exemplo de operador que utiliza read_some
  - O operador Nested Loop Join
    - Precisa ler apenas as tuplas do lado direito que tenham m.idM = mc.idM
    - E o operador conectado (movie_cast) consegue realizar essa busca



# Sumário

- Etapas no processamento de uma consulta SQL
  - Otimização de consultas
- Plano de execução
  - Fluxo de Execução
  - Read all
  - Read some

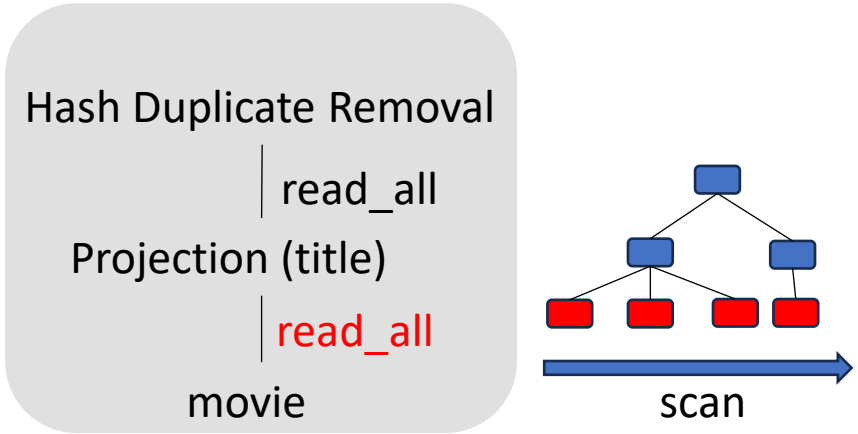
# Read All

- Ao solicitar um read_all, um operador estabelece uma conexão com o nó conectado, indicando que quer ler todas as tuplas

- Se o operador conectado for fonte
  - No momento da conexão, é estabelecido um cursor de leitura, apontando para a primeira tupla
  - Ex. Em uma árvore B+
    - a primeira tupla é o primeiro registro do nó folha mais à esquerda da árvore
    - Os nós folha são lidos conforme novas tuplas são requisitadas por meio de chamadas next

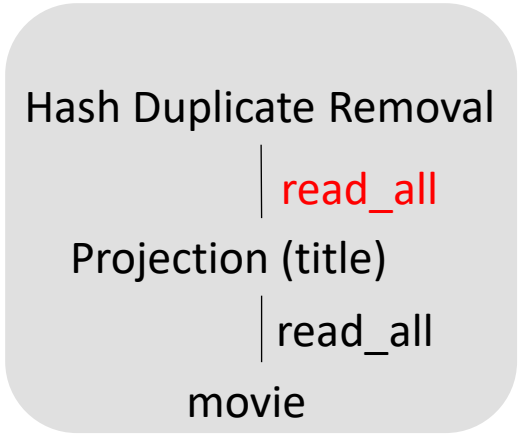
# Read All

- Exemplo 1
  - A projeção precisa de todos os registros
    - Por isso, ela solicita um read_all ao operador conectado
  - Como o operador conectado é uma árvore B+
    - O read_all leva a uma varredura (scan) no nível folha da árvore
      - Essa operação pode ser chamado de table scan



# Read All

- Exemplo 1
  - O Hash Duplicate Removal precisa de todos os registros
    - Por isso, ele solicita um read_all
  - Devido ao pipeline
    - O read_all entre Hash Duplicate Removal->Projection leva ao read_all entre Projection->movie
    - O next() em Projection leva ao next() em movie, fazendo avançar o cursor de leitura da tabela



# Sumário

- Etapas no processamento de uma consulta SQL
  - Otimização de consultas
- Plano de execução
  - Fluxo de Execução
  - Read all
  - Read some

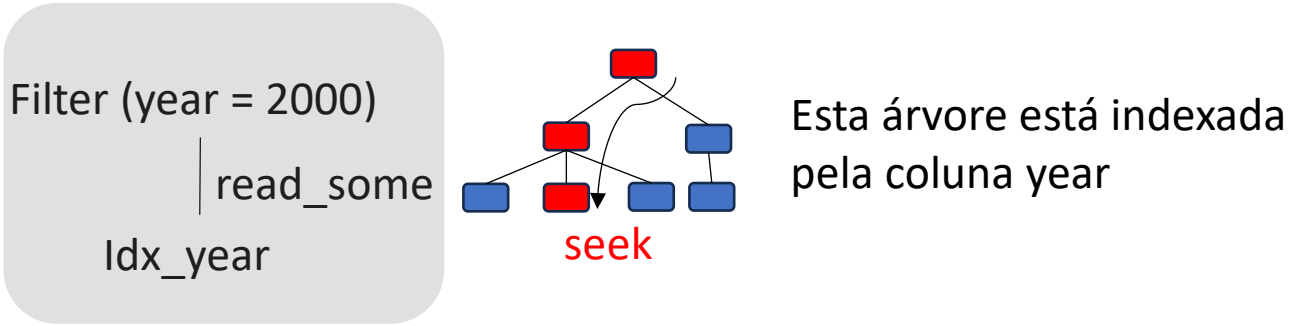
# Read Some

- Ao solicitar um read_some, um operador estabelece uma conexão com o nó conectado, indicando que quer ler todas as tuplas que satisfazem uma condição de filtragem
  - O operador conectado se responsabiliza por localizar as tuplas que satisfazem a condição
  - As chamadas next() retornam apenas as tuplas válidas

- Obs. O operador só solicita um read_some se o operador conectado for indexado.
  - Caso contrário, ele solicita um read_all

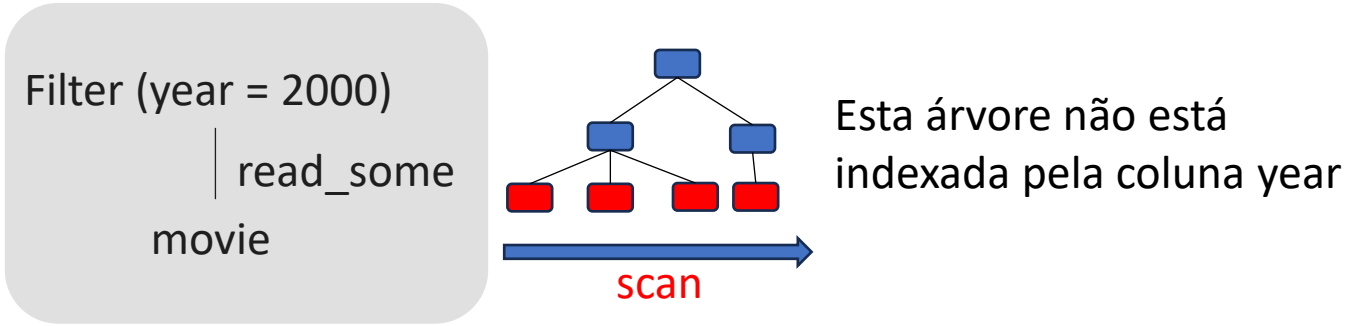
# Read Some

- Exemplo 1
  - O operador Filter só quer algumas tuplas
    - Como o operador conectado é um índice, é realizado um read_some
  - O índice é uma árvore B+ contendo a coluna year como chave de busca
    - O filtro pode ser resolvido usando a chave de busca
      - Por isso, a cada chamada next(), é realizado um scan no nível folha da árvore (index seek)



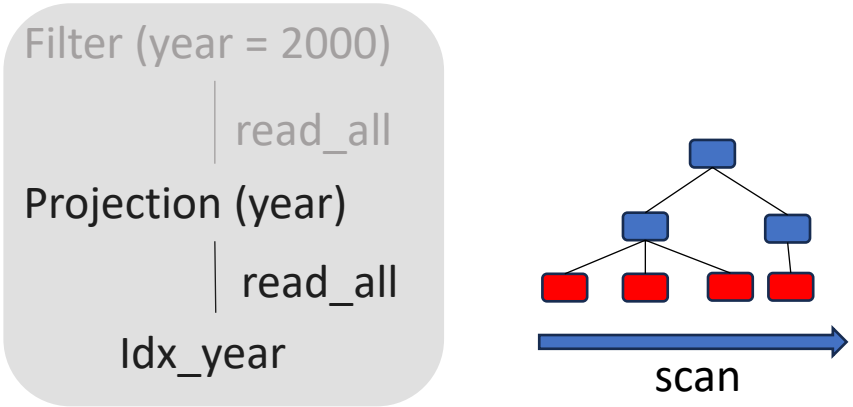
# Read Some

- Exemplo 2
  - O operador Filter só quer algumas tuplas
    - Como o operador conectado é um índice, é realizado um read_some
  - O índice é uma árvore B+ contendo a coluna idM como chave de busca
    - O filtro **não** pode ser resolvido usando a chave de busca
      - Por isso, é realiza um scan no nível folha da árvore (table scan)
        - Cada chamada next() faz avançar o ponteiro de leitura da árvore, até que um registro válido seja encontrado



# Read Some

- Exemplo 3
  - O operador Projection precisa de todos os registros
    - Por isso, ele solicita um read_all
  - É realizado um scan no nível folha da árvore (table scan)
  - Conforme ocorrem as chamadas next(), o ponteiro de leitura no nível folha é atualizado

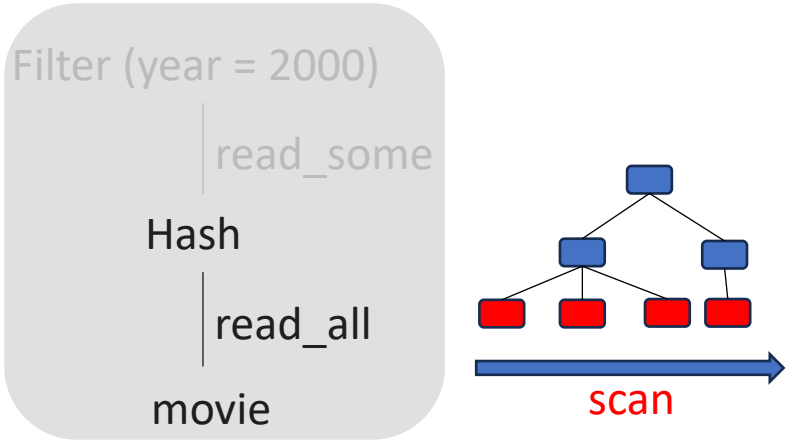


# Read Some

- Exemplo 3
  - O operador Filter só quer algumas tuplas
    - Como o operador conectado **não** é um índice, é realizado um read_all
  - Devido ao pipeline
    - O next() em Filter leva ao next() em idx_year, fazendo avançar o cursor de leitura do índice
  - As tuplas que não satisfazem o filtro só são identificadas e removidas dentro do operador Filter

# Read Some

- Exemplo 4
  - O operador Hash precisa de todos os registros
    - Por isso, ele solicita um read_all
  - É realizado um table scan para recuperar todas as tuplas da tabela conectada
  - As tuplas são salvas em uma tabela hash em memória, que usa year como chave de busca



# Read Some

- Exemplo 4
  - O operador Filter só quer algumas tuplas
    - Como o operador conectado é um índice, é realizado um read_some
  - O operador Hash usa year como chave de busca
    - Por isso, ele consegue localizar as entradas relevantes de forma eficiente por meio da tabela hash materializada

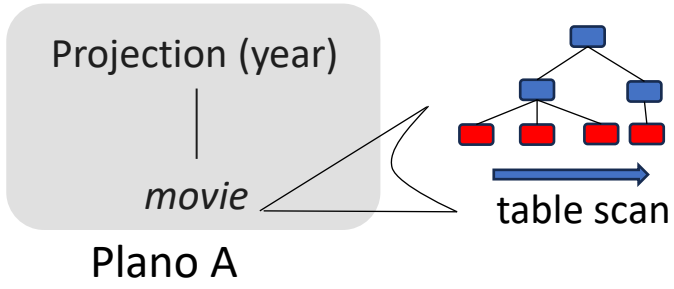
# Encerrando

- Terminologia usada para acessos à nós fonte
  - Scan
    - Table scan: quando o read_all é sobre uma tabela
    - Index scan: quando o read_all é sobre um índice
  - Seek
    - Index seek: quando o read_some é sobre um índice em disco que usa como chave de busca o atributo filtrado
- Os termos podem variar dependendo do SGBD escolhido

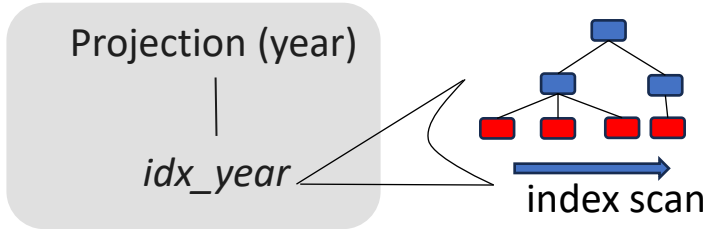
# Atividade Individual

- Plano A: acessa todo o arquivo de dados (table scan)
- Plano B: acessa todo o arquivo do índice secundário (index scan)

- Qual é melhor?



Plano A

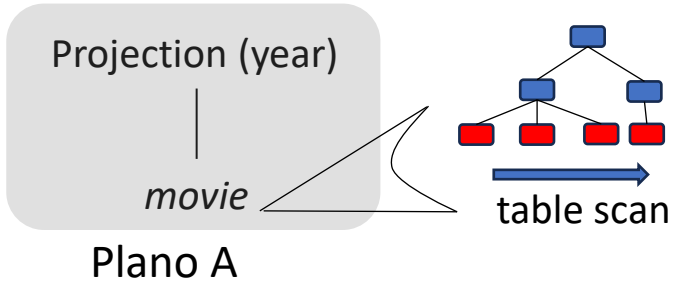


Plano B

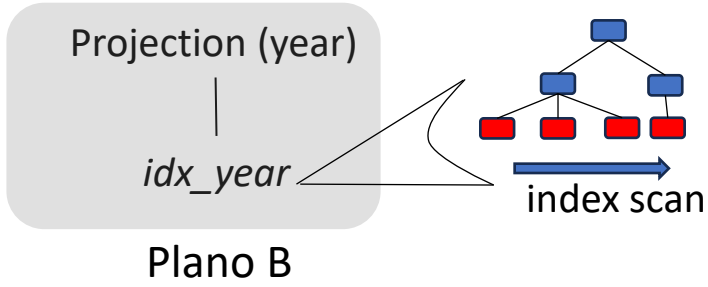
# Atividade Individual

- Crie os dois planos de execução no DBest

- Escreva um relatório em PDF indicando os resultados encontrados e responda por que um plano é superior ao outro




Plano A




Plano B

# Atividade Individual

- Para construir o plano B, será necessário criar um índice para a coluna year.
- O índice usará year como chave e movieId como valor.
- Para criar o índice
  - Faça uma projeção de year e movieId
  - Com o botão direito sobre a projeção, selecione a opção Export Table -> Non-unique Index.
  - Indique em que lugar no disco o índice deve ser criado.

# Atividade Individual

- Também será necessário comparar os dois planos gerados.

- O DBest possui um recurso no painel inferior chamado Comparator
  - Esse recurso compara planos de execução usando vários indicadores de custo

- Para usar o Comparator
  - Clique com o botão esquerdo sobre o operador que você deseja analisar e escolha a opção mark
    - Geralmente se deseja analisar a raiz da árvore, que representa a consulta completa
  - Marque os dois operadores raiz que serão comparados
    - Eles serão destacados com uma cor vermelha
  - Clique no “Comparator” e analise os indicadores

- Para esta atividade, o indicador de custo a analisar será o Accessed Blocks, que indica quantas vezes alguma página precisou ser acessada para responder a consulta