Tipos avançados de junções

Sumário

- Junção externa
- Semi-junção
  - Semi-junção vs junção regular
  - In vs Exists
- Anti-junção
  - anti-junção vs junção externa
  - Not In vs Not Exists

Junção Externa

- Considerando que uma junção tem duas partes
  - Parte principal
  - Parte secundária

- A junção externa traz todos os registros da parte principal
  - Mesmo que alguns desses registros não tenham correspondência com nenhum registro da parte secundária

- Possibilidades mais usadas
  - LEFT (OUTER) JOIN: O lado da esquerda da junção externa é a parte principal
  - RIGHT (OUTER) JOIN: O lado da direita da junção externa é a parte principal

Junção Externa

- Ex.

```
SELECT m.title, mc.c_name
FROM movie m LEFT JOIN movie_cast mc ON m.idM = mc.idM
```

- Partes da junção
  - Principal: movie
  - Secundária: movie_cast

- Retorno da junção
  - Todos os filmes
    - combinados com os nomes de personagens(c_name) vindos de movie_cast
    - Filmes sem correspondência em movie_cast também são retornados
      - Nesses casos, a coluna c_name é preenchida com o valor NULO

Junção Externa

- As estratégias de junção externa podem ser classificadas em
  - Left Outer Join
    - A parte principal fica do lado externo (esquerdo) da junção
  - Right Outer Join
    - A parte principal fica do lado interno (direito) da junção

Junção Externa

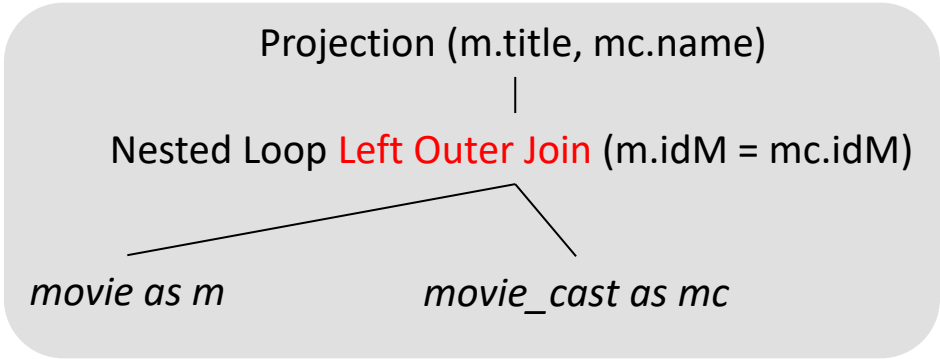
- Exemplo 1
  - retornar títulos de filmes e nomes de seus personagens.
  - Filmes sem personagens também devem ser retornados
    - Com o nome do personagem nulo

```
SELECT m.title, mc.c_name
FROM movie m
LEFT JOIN movie_cast mc ON m.idM = mc.idM
```

Junção Externa

- O Nested Loop pode ser usado para implementar a junção externa
  - A parte principal (movie) fica no lado externo da junção
  - Para cada registro da parte principal, as correspondências são buscadas
    - Se um registro não possuir correspondência
      - Ele é complementado com nulo
    - Obs. Essa verificação adicional torna a junção externa um pouco menos eficiente do que a junção regular

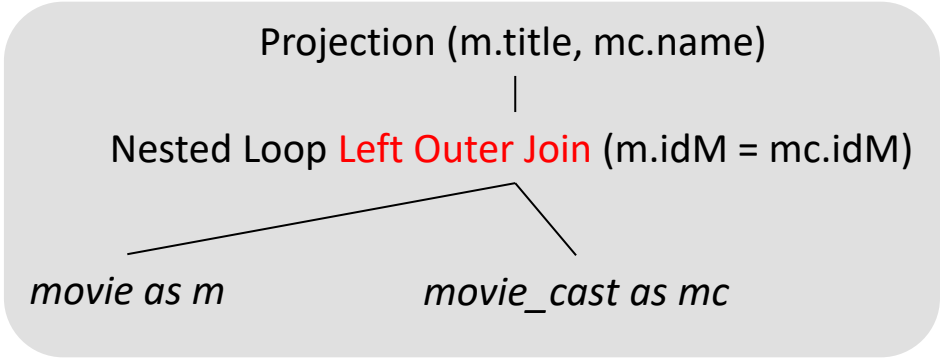
```
SELECT m.title, mc.c_name
FROM movie m
LEFT JOIN movie_cast mc ON m.idM = mc.idM
```



Junção Externa

- O mesmo plano se aplica usando RIGHT JOIN na consulta SQL, com movie do lado direito
  - A parte principal (movie) continua no lado externo da junção

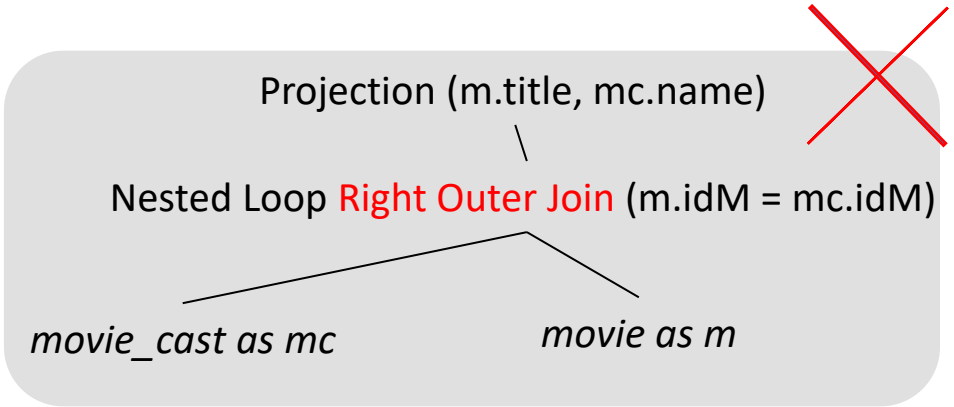
```
SELECT m.title, mc.c_name
FROM movie_cast mc
RIGHT JOIN movie m ON mc.idM = m.idM
```



Junção Externa

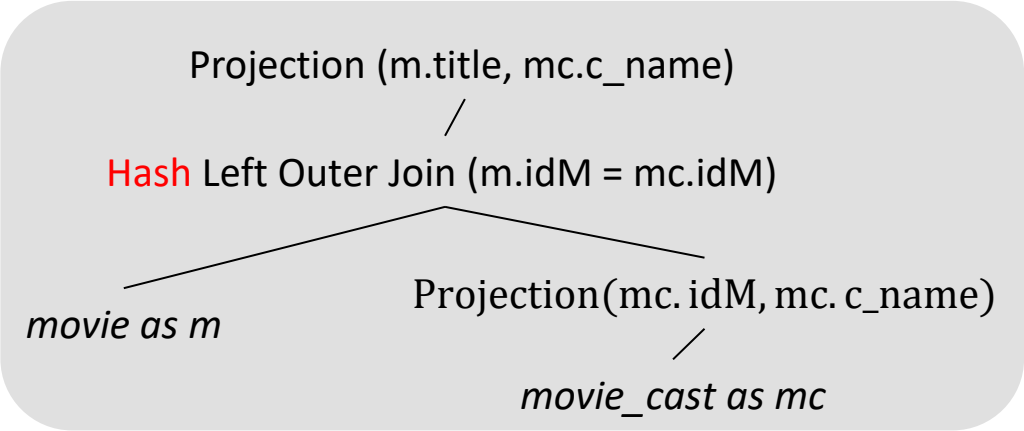
- Limitação do Nested Loop para junção externa
  - A parte principal deve sempre ficar do lado externo (esquerdo)
  - Ou seja, não existe o Nested Loop Right Outer Join
  - Para ter maior flexibilidade, o SGBD pode usar outras estratégias, como variações do Hash Join

```
SELECT m.title, mc.c_name
FROM movie m
LEFT JOIN movie_cast mc ON m.idM = mc.idM
```



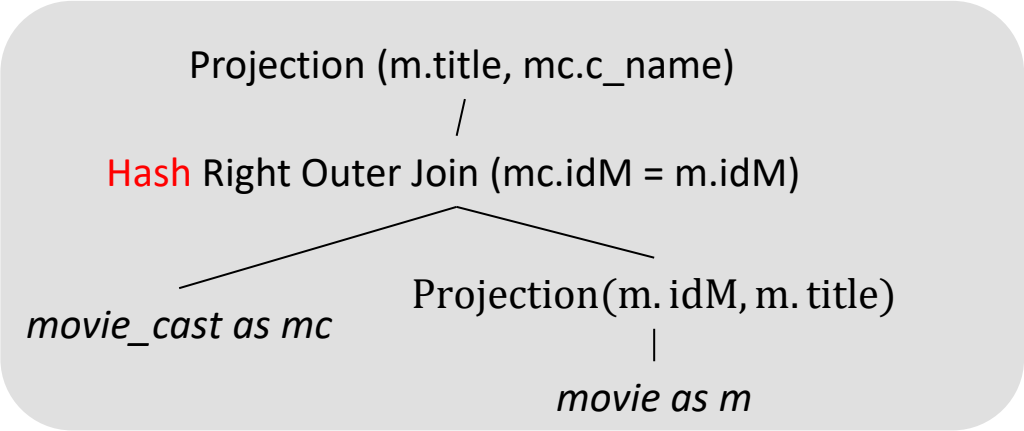
Junção Externa

- O plano abaixo utiliza materialização com Hash Left Outer Join
  - Parte principal na esquerda (left)
  - Parte secundária materializada em uma tabela hash



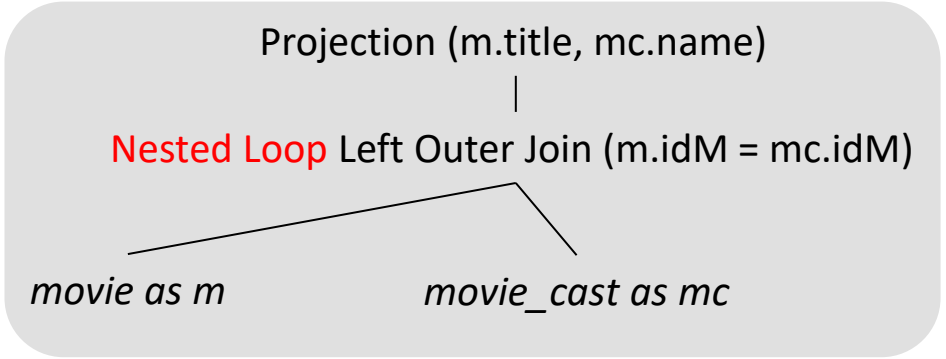
Junção Externa

- Agora a materialização foi realizada com o Hash Right Outer Join
  - Parte principal na direita (right)
  - Parte principal materializada em uma tabela hash

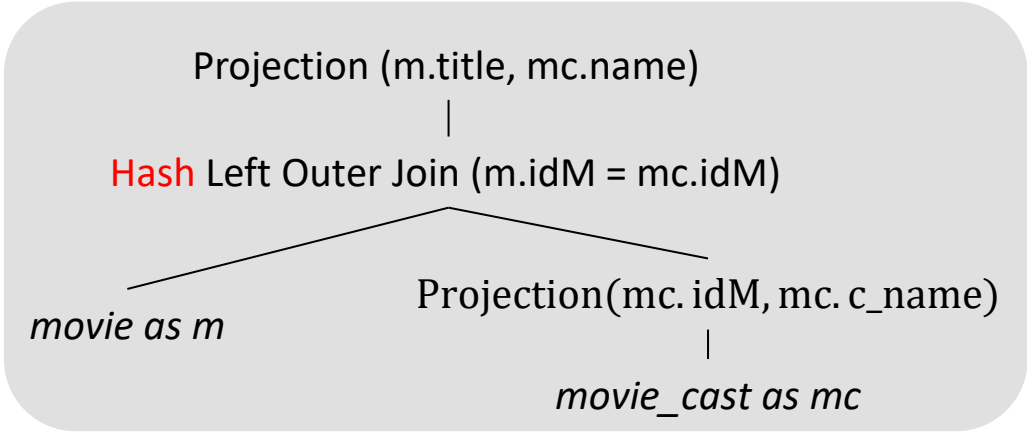


Junção Externa

- Os planos A e B mostram essas duas variações do left outer join
  - Plano A: com Nested Loop Left Outer Join
  - Plano B: com Hash Left Outer Join
- Qual deles é mais eficiente?



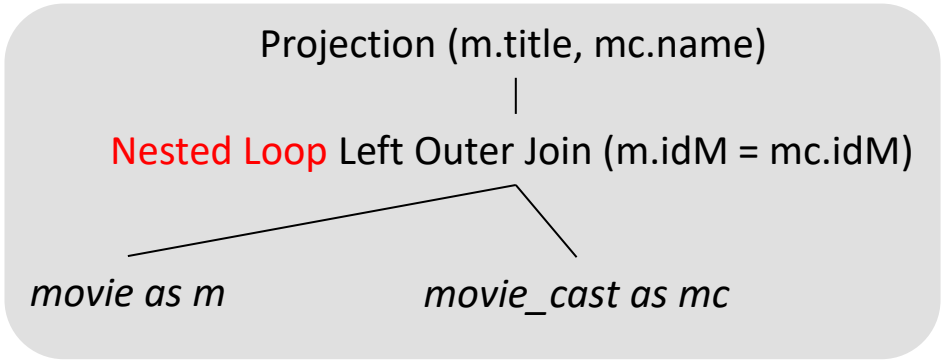
Plano A



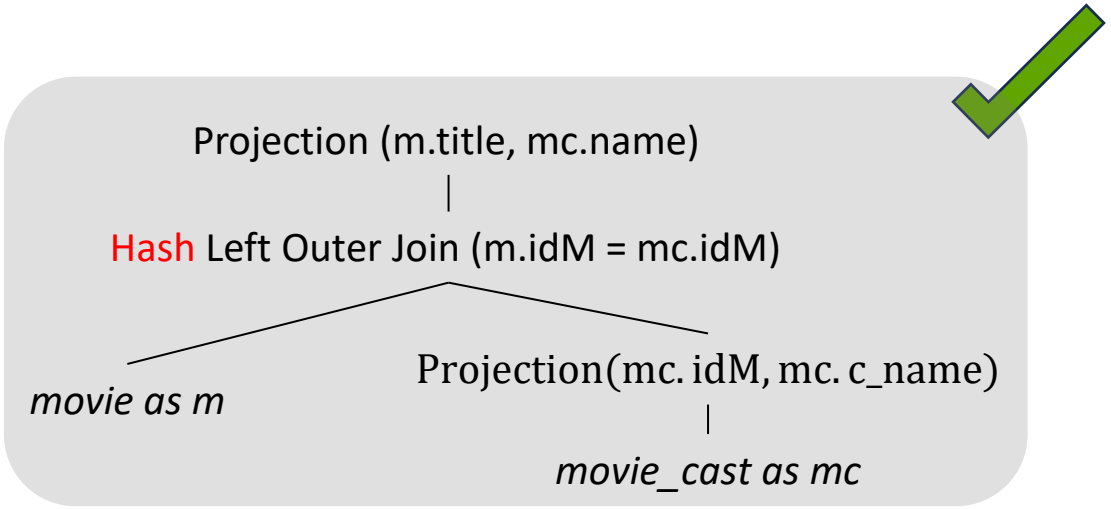
Plano B

Junção Externa

- Plano B é mais eficiente
  - A busca é feita sobre uma tabela hash
  - Contudo, consome memória



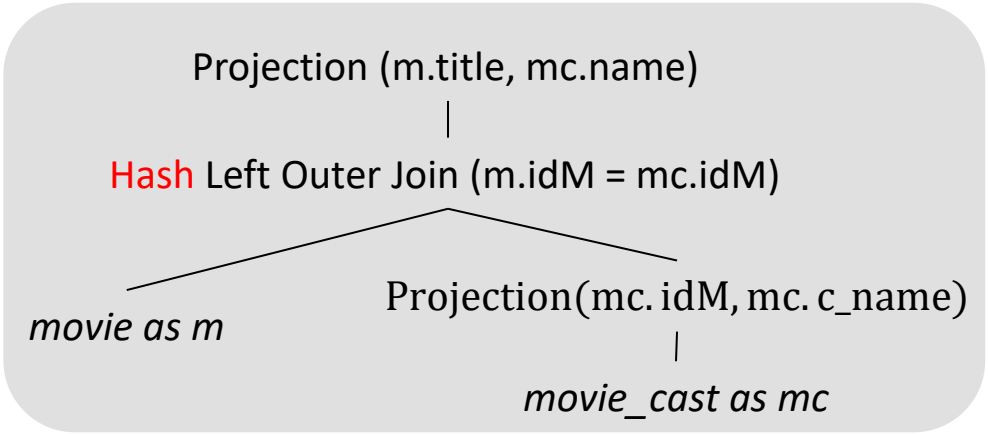
Plano A



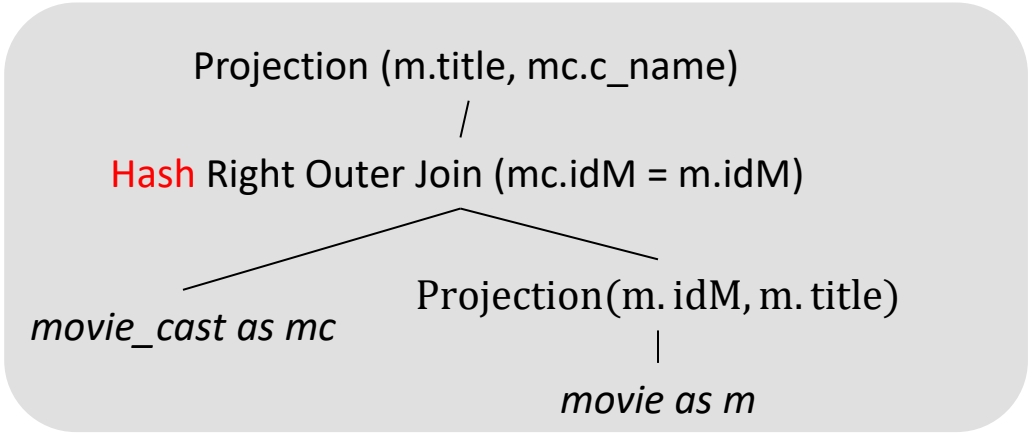
Plano B

Junção Externa

- Agora os planos A e B mostram essas duas variações do Hash outer join
  - Plano A: com a parte principal à esquerda
  - Plano B: com a parte principal à direita
- Qual deles consome menos memória?



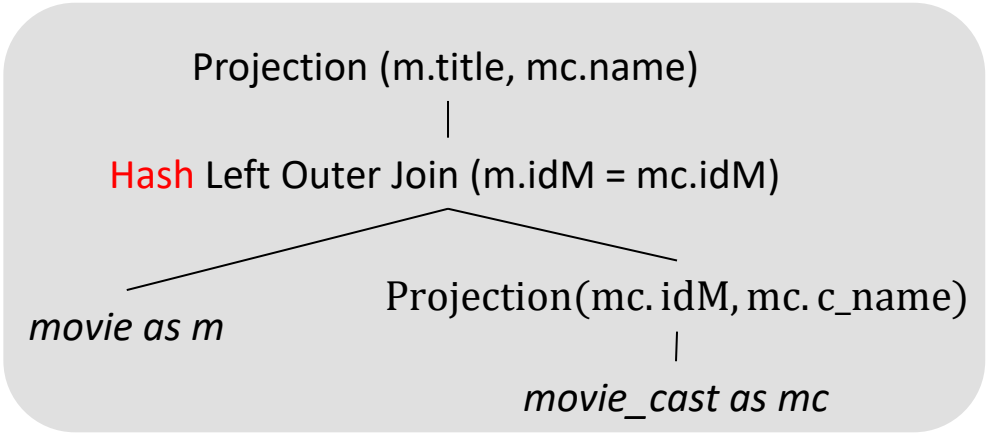
Plano A



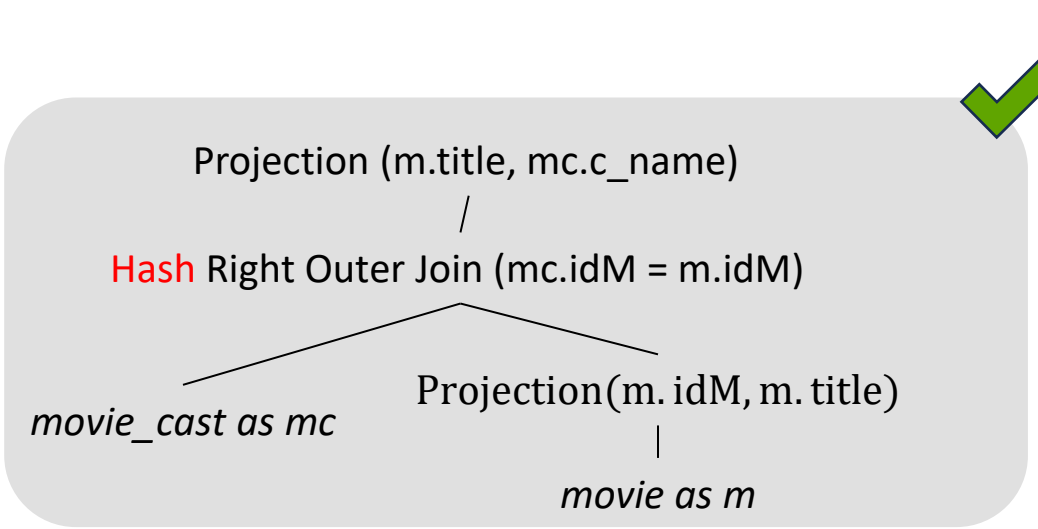
Plano B

Junção Externa

- O plano B consome menos memória
  - A tabela hash de movie é menor do que a tabela hash de movie_cast



Plano A



Plano B

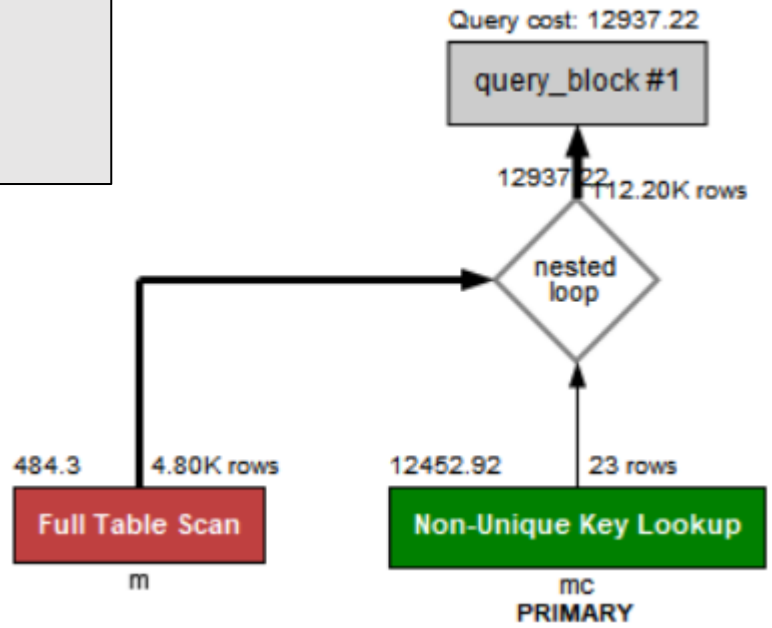
Junção Externa

```
SELECT m.title, mc.c_name
FROM movie m
LEFT JOIN movie_cast mc ON m.movie_id = mc.movie_id
```

MySQL escolheu o Nested Loop

A versão visual é incompleta: ela só indica que se trata de um Nested Loop.

O left Join só aparece no plano escrito



MySQL

-> Nested loop left join
   -> Table scan on m
   -> Index lookup on mc using PRIMARY (movie_id=m.movie_id)

Junção Externa

```
SELECT m.title, mc.c_name
FROM movie m
LEFT JOIN movie_cast mc ON m.movie_id = mc.movie_id
```

Já o PostgreSQL usou um Hash Rigth Join (Hash Right Outer Join).

Por que não um Hash Left Outer Join? Possivelmente para reduzir o consumo de memória da tabela hash

PostgreSQL

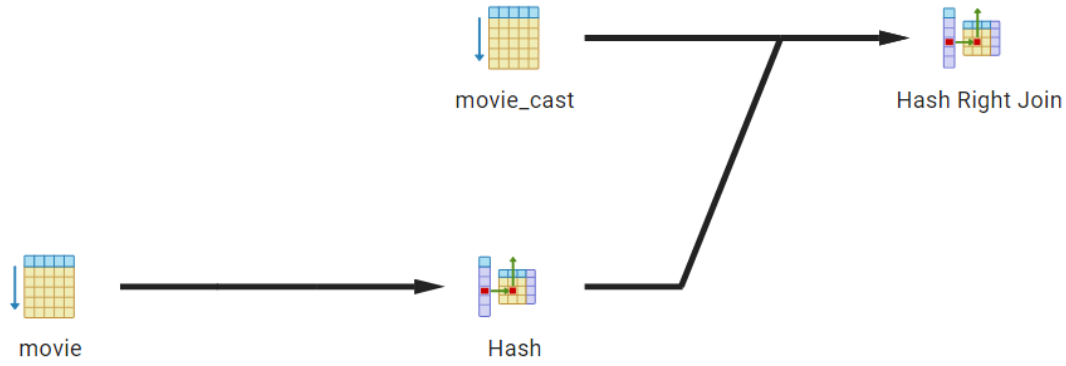
Hash Right Join

Hash Cond: (mc.movie_id = m.movie_id)

-> Seq Scan on movie_cast mc

-> Hash

-> Seq Scan on movie m



Junção Externa

- Junção externa é diferente da junção regular!
  - Seu processamento é um pouco mais custoso
  - Pode dificultar a troca do lado da junção
    - Ou exigir que seja usada uma solução materializada
      - Que aumenta o consumo de memória

- Por isso, certifique-se de que
  - as tuplas sem correspondência são realmente importantes para a aplicação
  - não seja possível eliminar a junção externa sem que isso altere o resultado

Sumário

- Junção externa
- Semi-junção
  - Semi-junção vs junção regular
  - In vs Exists
- Anti-junção
  - anti-junção vs junção externa
  - Not In vs Not Exists

Semi-junção

- Considerando que uma junção tem duas partes
  - parte principal
  - parte secundária

- Em uma semi-junção
  - Apenas tuplas da parte principal são retornadas
  - Cada tupla pode ser retornada apenas uma vez
    - Caso haja correspondências com a parte secundária

Semi-junção

- Em SQL, semi-junções são expressas na forma de subconsultas
  - A parte principal é a consulta externa
  - A parte secundária é a subconsulta

- Os exemplos abaixo mostram duas formas de uso (EXISTS e IN)
  - Parte principal: movie
  - Parte secundária: movie_cast

```
SELECT title
FROM movie m WHERE EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

```
SELECT title
FROM movie WHERE idM IN
        (SELECT idM FROM movie_cast)
```

Semi-junção

- A linguagem SQL fornece diversos operadores para subconsultas
- Ex.
    - IN
    - EXISTS
    - ALL
    - SOME
- O IN e o EXISTS são os mais comumente usados

Semi-junção

- EXISTS
  - a subconsulta contém uma coluna de correlação (m.idM)
  - Não importa o que a subconsulta retorna
    - Basta verificar que algo está sendo retornado
    - No exemplo, o retorno é uma constante (1)

```
SELECT title
FROM movie m WHERE EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

Semi-junção

- IN
  - A subconsulta é independente da consulta principal
    - Não possui coluna de correlação
  - As correspondências são feitas comparando colunas da consulta principal com o retorno da subconsulta

```
SELECT title
FROM movie WHERE idM IN
        (SELECT idM FROM movie_cast)
```

Semi-junção

- As estratégias de semi-junção podem ser classificadas em
  - Left Semi Join
    - A parte principal fica do lado externo (esquerdo) da junção
  - Right Semi Join
    - A parte principal fica do lado interno(direito) da junção

Semi-junção

- Exemplo 1: título de filmes que contenham membros do elenco registrados

```
SELECT title
FROM movie m WHERE EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

- Como essa consulta é representada internamente?

Semi-junção

- A consulta é um exemplo de semi-junção
  - Retorna títulos de filme que contenham membros do elenco
  - Cada filme que satisfaça essa restrição é retornado uma única vez
  - Nenhuma coluna de elenco é retornada

```
SELECT title
FROM movie m WHERE EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

Semi-junção

- A mesma consulta poderia ser representada usando IN

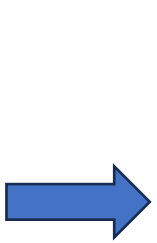
```
SELECT title
FROM movie m WHERE EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

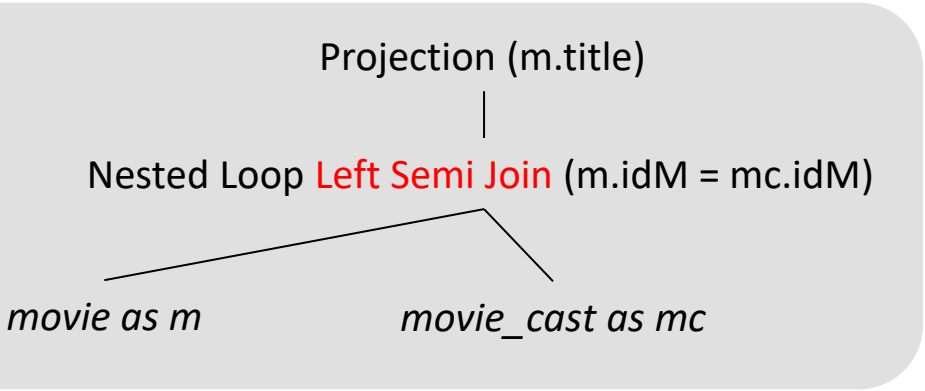
```
SELECT title
FROM movie WHERE idM IN
        (SELECT idM FROM movie_cast)
```

Semi-junção

- O plano abaixo usa um Nested Loop Left Semi Join
  - A parte principal (movie) fica no lado esquerdo (left) da junção
    - Apenas registros da parte principal podem ser retornados
  - A junção não busca correspondências
    - Apenas verifica se elas existem
      - Por isso, pouco importa o que foi usado no SELECT da subconsulta

```
SELECT title
FROM movie m WHERE EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

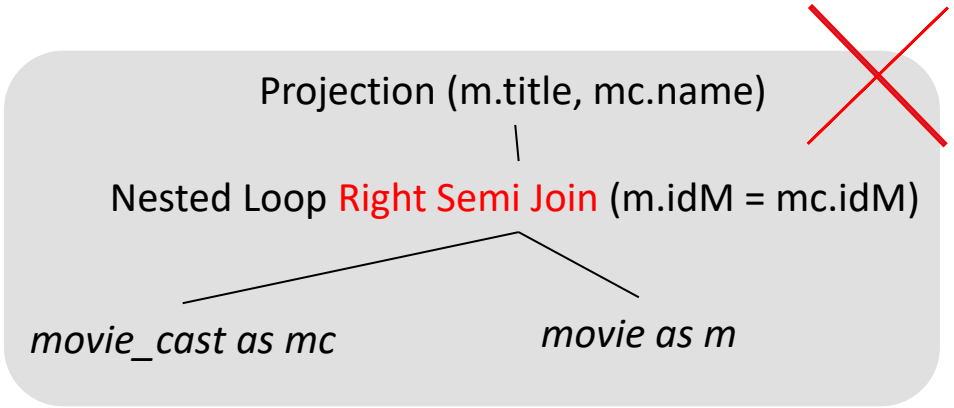




Semi-junção

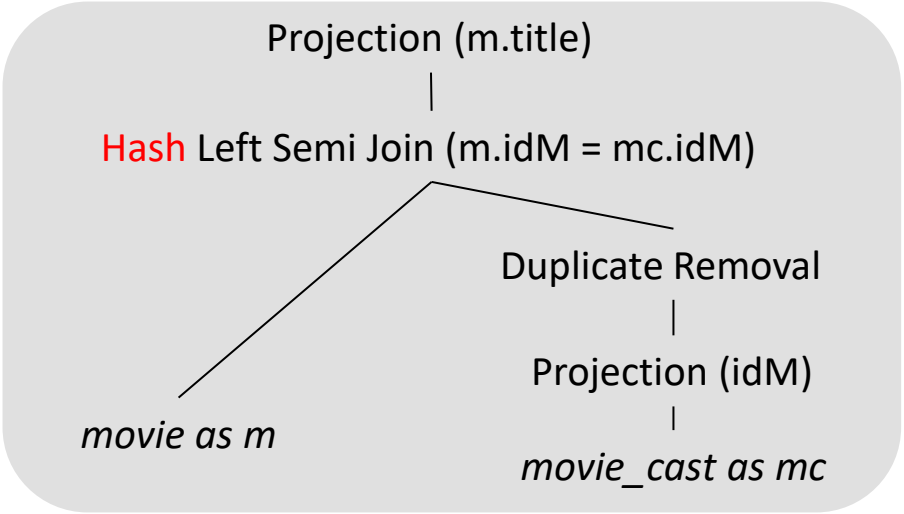
- Limitação do Nested Loop para semi junção
  - A parte principal deve sempre ficar do lado esquerdo
  - Ou seja, não existe o Nested Loop Right Semi Join
  - Para ter maior flexibilidade, o SGBD pode usar alguma outra estratégia
    - Como variações do Hash Join

```
SELECT title
FROM movie m WHERE EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```



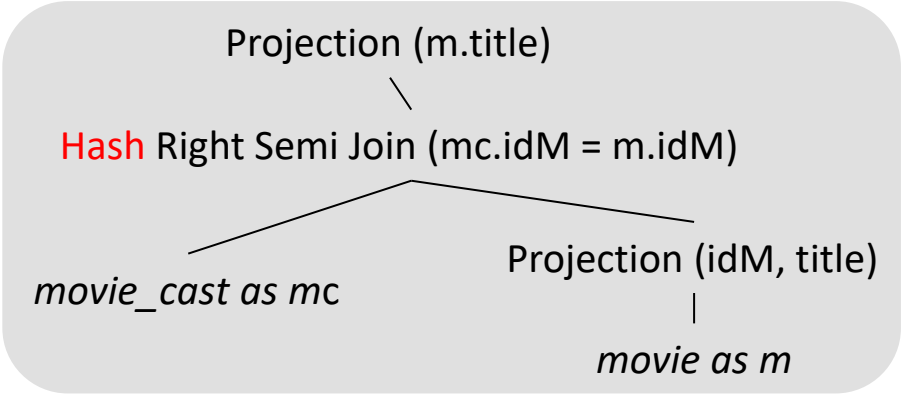
Semi-junção

- O plano abaixo usa um Hash Left Semi Join
  - Parte principal do lado esquerdo (left)
  - Parte secundária materializada em uma tabela hash, que contém apenas valores únicos de idM
    - A verificação do semi Join é feita sobre a tabela hash



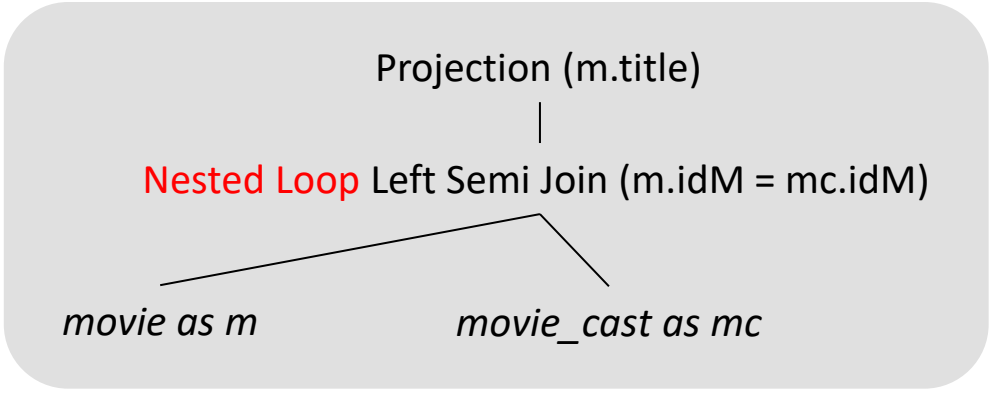
Semi-junção

- O plano abaixo usa um Hash Right Semi Join
  - Parte principal do lado direito (right)
  - Parte principal materializada em uma tabela hash

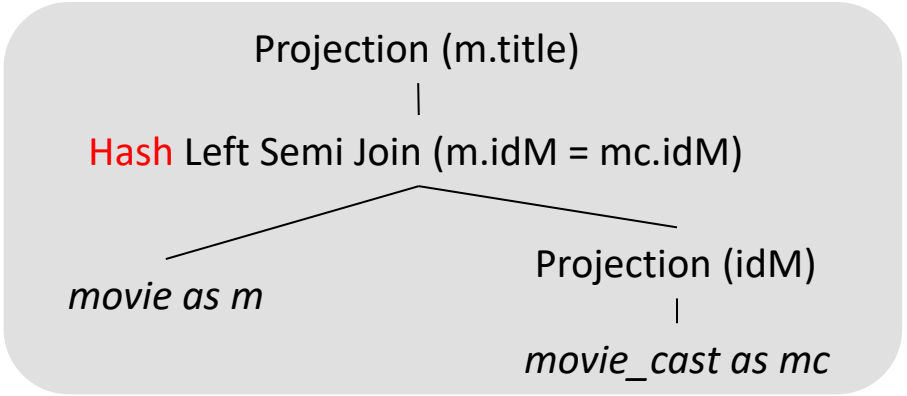


Semi-junção

- Os planos A e B mostram essas duas possiblidades de Left Semi Join
  - Plano A: com Nested Loop Left Semi Join
  - Plano B: com Hash Left Semi Join
- Qual deles é mais eficiente?



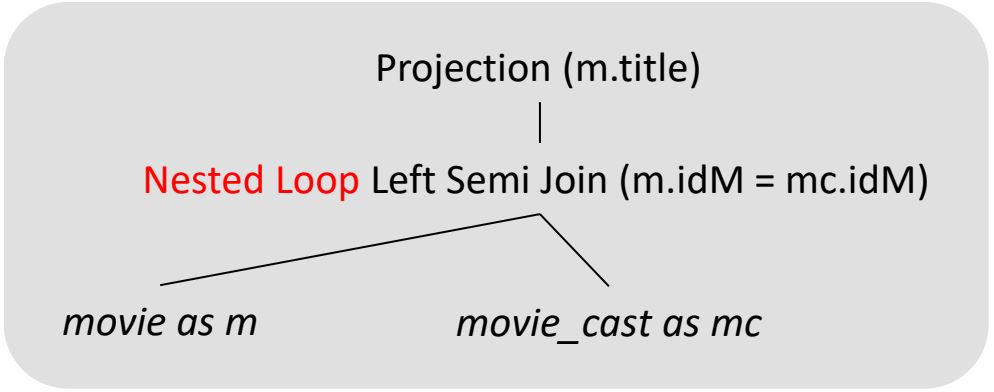
Plano A



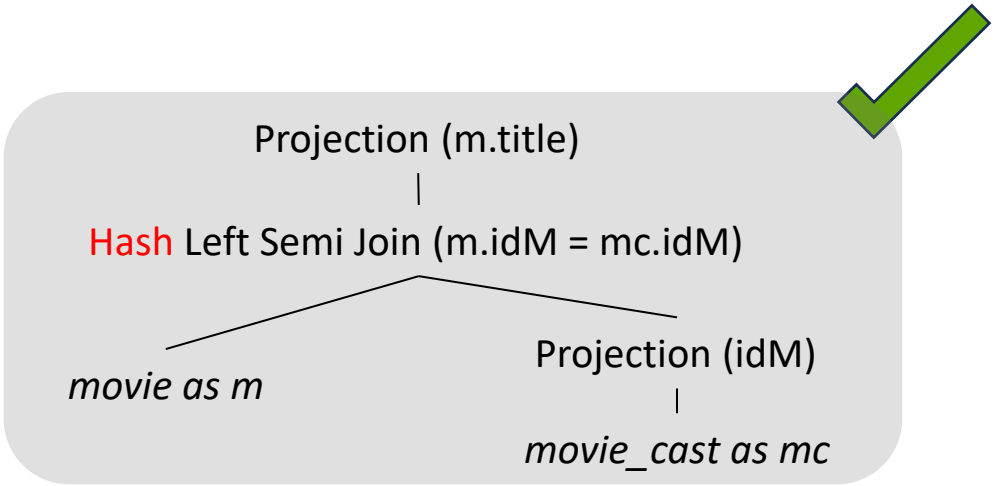
Plano B

Semi-junção

- Plano B é mais eficiente
  - A busca é feita sobre uma tabela hash
  - No entanto, consome memória



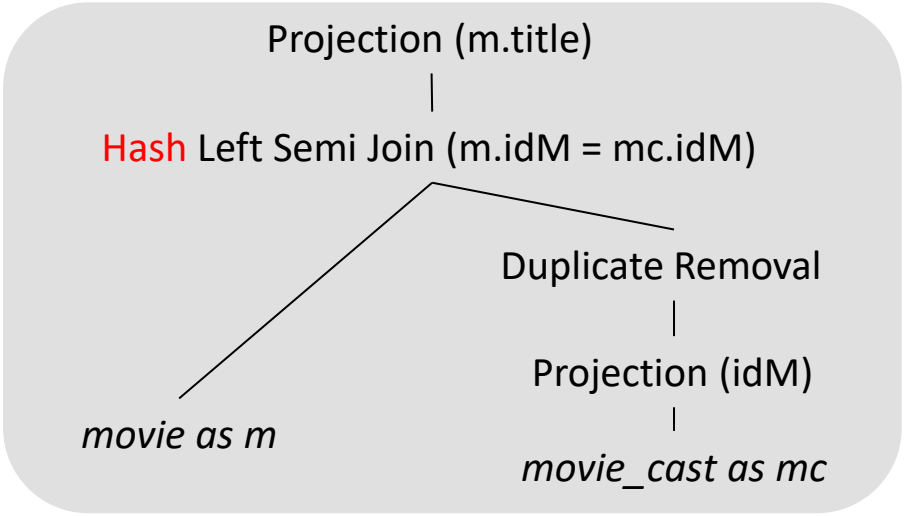
Plano A



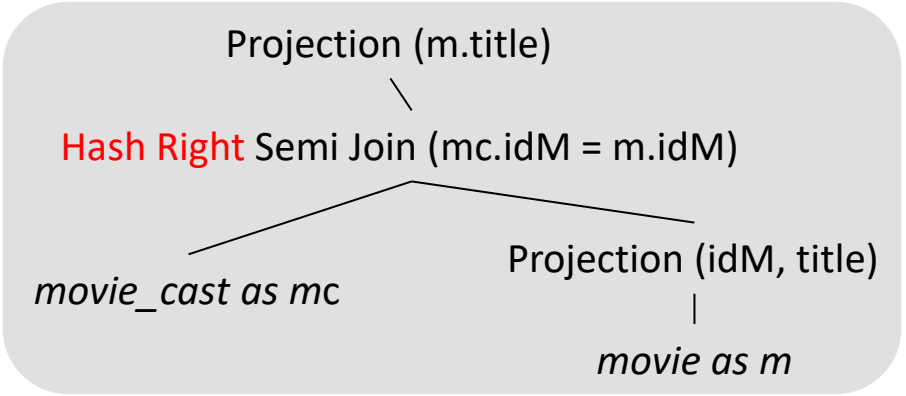
Plano B

Semi-junção

- Os planos A e B mostram essas duas possiblidades de Hash Semi Join
  - Plano A: com a parte principal na esquerda
  - Plano B: com a parte principal na direita
- Qual deles consome menos memória?



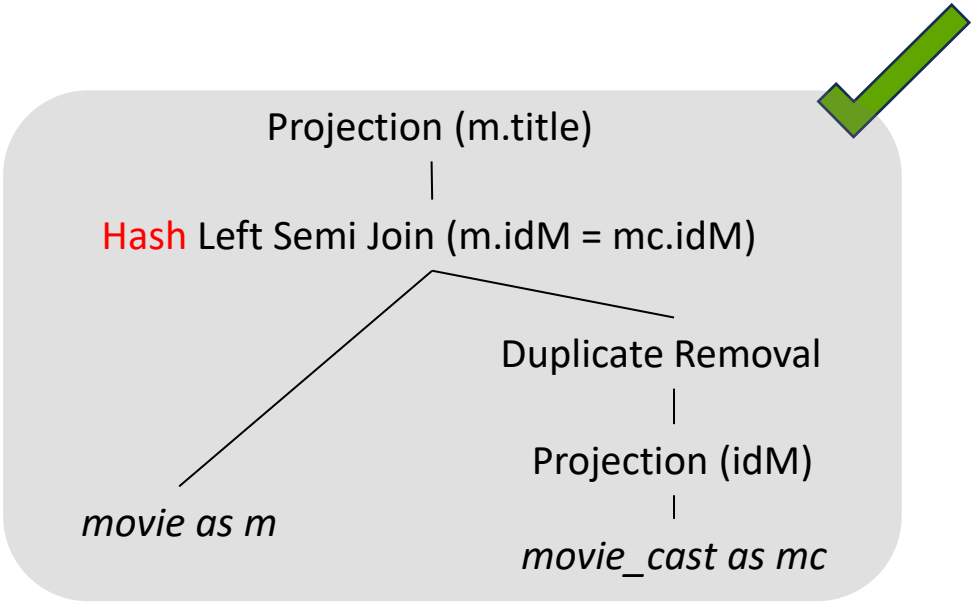
Plano A



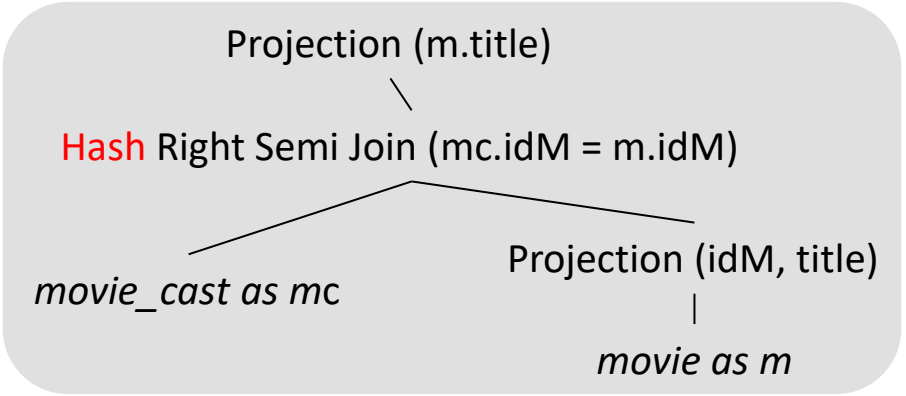
Plano B

Semi-junção

- Plano A consome menos memória
  - Apenas a coluna idM é materializada
  - Tem menos idMs únicos em movie_cast do que em movie



Plano A



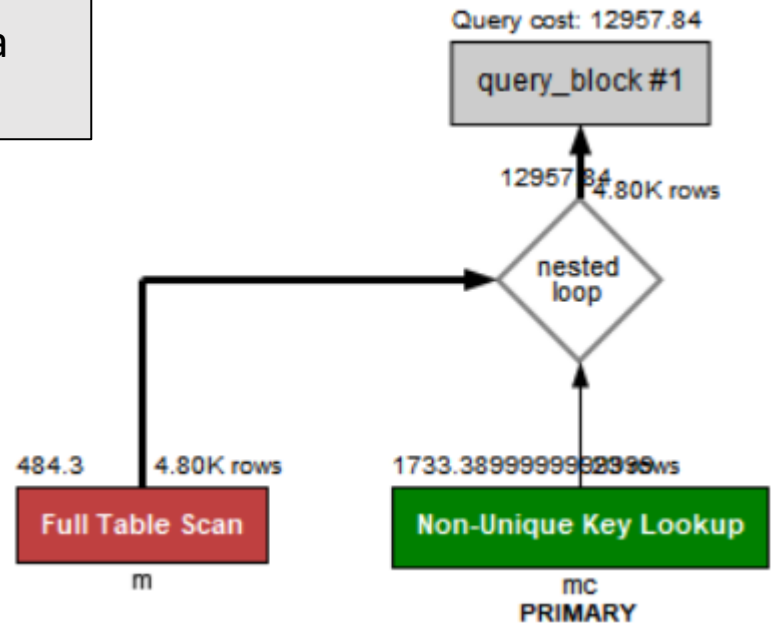
Plano B

Semi-junção

```
SELECT title
FROM movie m WHERE EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

O MySQL usou o Nested Loop Semi Join

O covering index indica que nenhuma complementação foi necessária: basta verificar se alguma entrada no índice foi encontrada



MySQL

-> Nested loop semijoin
    -> Table scan on m
    -> Covering index lookup on mc using PRIMARY (movie_id=m.movie_id)

Semi-junção

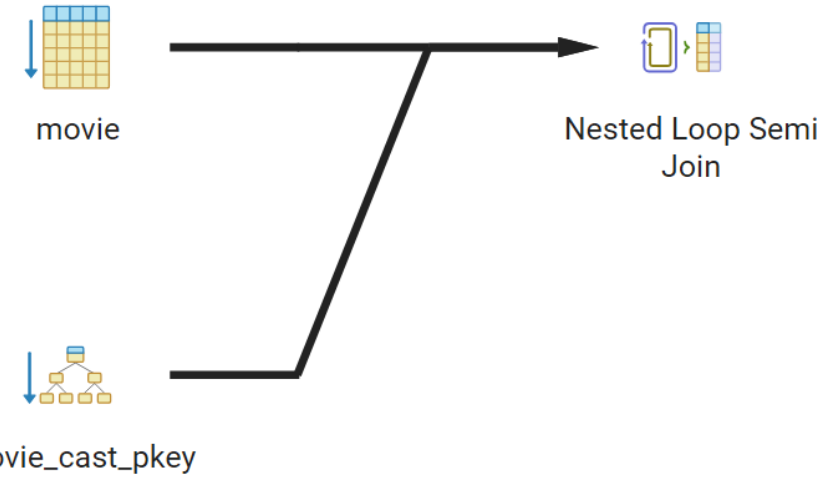
```
SELECT title
FROM movie m WHERE EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

O PostgreSQL também usou o algoritmo de semi-junção baseado em Nested Loop

O 'index only scan' também indica que não foi necessária complementação

PostgreSQL

Nested Loop Semi Join
  -> Seq Scan on movie m
  -> Index Only Scan using movie_cast_pkey on movie_cast mc
     Index Cond: (movie_id = m.movie_id)



Sumário

- Junção externa
- Semi-junção
  - Semi-junção vs junção regular
  - In vs Exists
- Anti-junção
  - anti-junção vs junção externa
  - Not In vs Not Exists

Semi-junção vs junção regular

- Como vimos, uma semi-junção é expressa por meio de uma subconsulta

```
SELECT idM, title
FROM movie WHERE idM IN
        (SELECT idM FROM movie_cast)
```

Com semi-junção

Semi-junção vs junção regular

- Uma semi-junção também pode ser expressa como uma junção regular

```
SELECT idM, title
FROM movie WHERE idM IN
        (SELECT idM FROM movie_cast)
```

Com semi-junção

```
SELECT DISTINCT idM, title
FROM movie m
JOIN movie_cast mc ON m.idM = mc.idM
```

Sem semi-junção

Semi-junção vs junção regular

- Quando uma junção regular é equivalente a uma semi-junção?
  - O SELECT retorna apenas dados da parte principal(movie)
  - As colunas retornadas são únicas na parte principal (idM, title)
  - O DISTINCT remove múltiplas ocorrências para um mesmo filme
    - Para os filmes que tenham vários movie_casts

```
SELECT idM, title
FROM movie WHERE idM IN
        (SELECT idM FROM movie_cast)
```

Com semi-junção

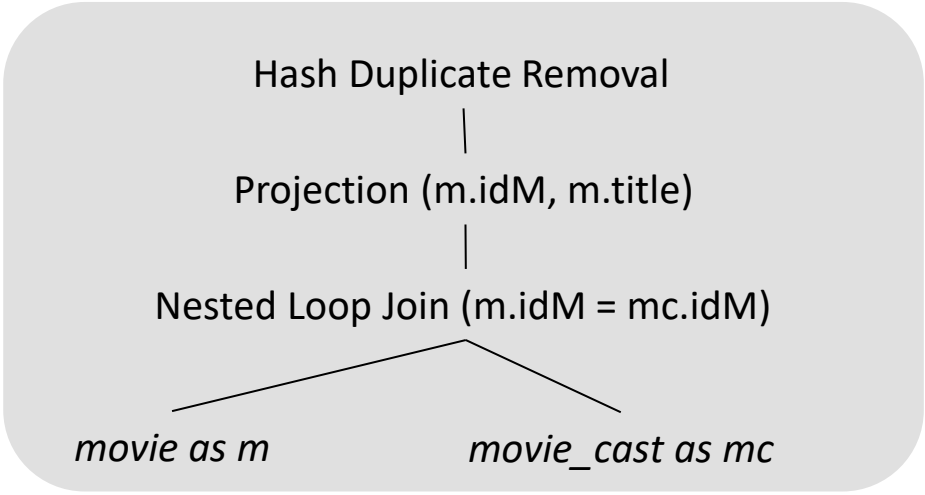
```
SELECT DISTINCT idM, title
FROM movie m
JOIN movie_cast mc ON m.idM = mc.idM
```

Sem semi-junção

Semi-junção vs junção regular

- Plano de execução usando junção regular
  - Um filme cruza com todos os seus movie_casts
  - Os cruzamentos sobressalentes são descartados
    - Com o operador Hash Duplicate Removal

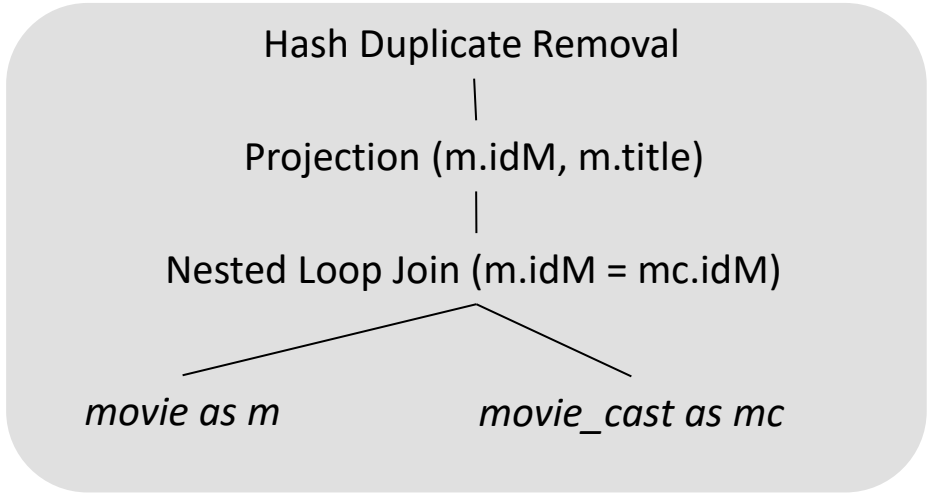
```
SELECT DISTINCT idM, title
FROM movie m JOIN movie_cast mc
ON m.idM = mc.idM
```



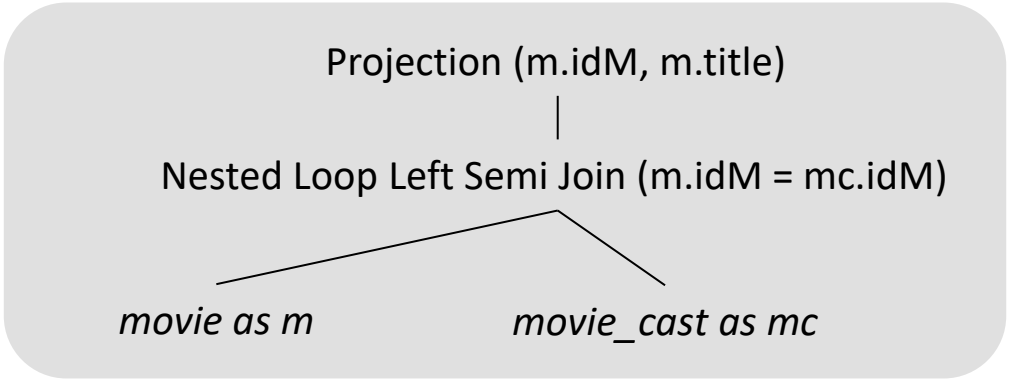
Plano A

Semi-junção vs junção regular

- Qual estratégia é melhor?



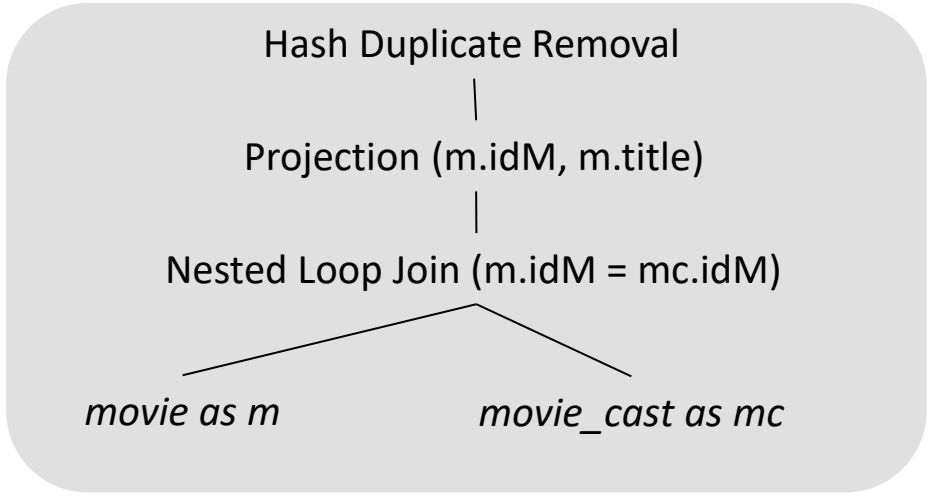
Plano A (usando junção regular)



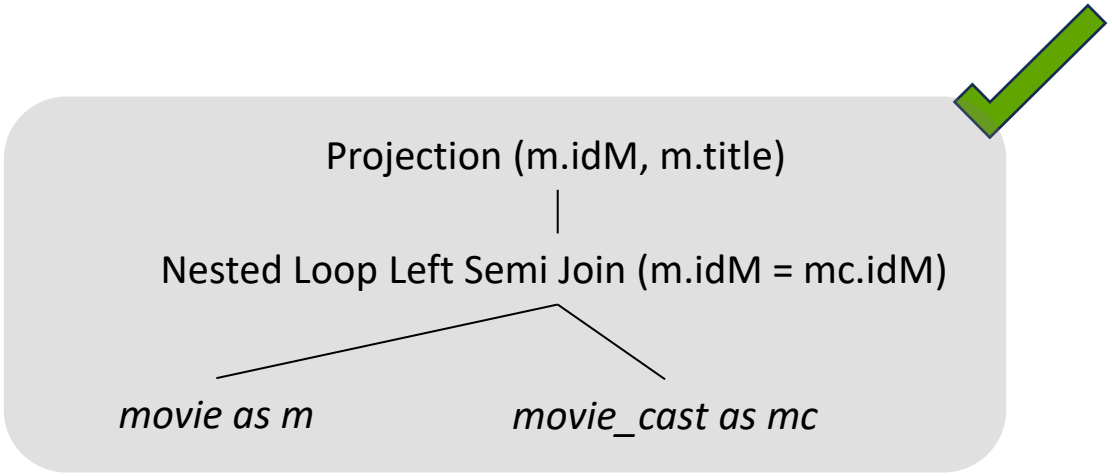
Plano B: usando semi-junção

Semi-junção vs junção regular

- O Plano B é melhor
  - Para cada filme, ocorre apenas uma verificação no lado interno
  - Em nenhum momento é realizado algum cruzamento
    - Isso evita o overhead de remoção de duplicatas



Plano A (usando junção regular)



Plano B: usando semi-junção

Semi-junção vs junção regular

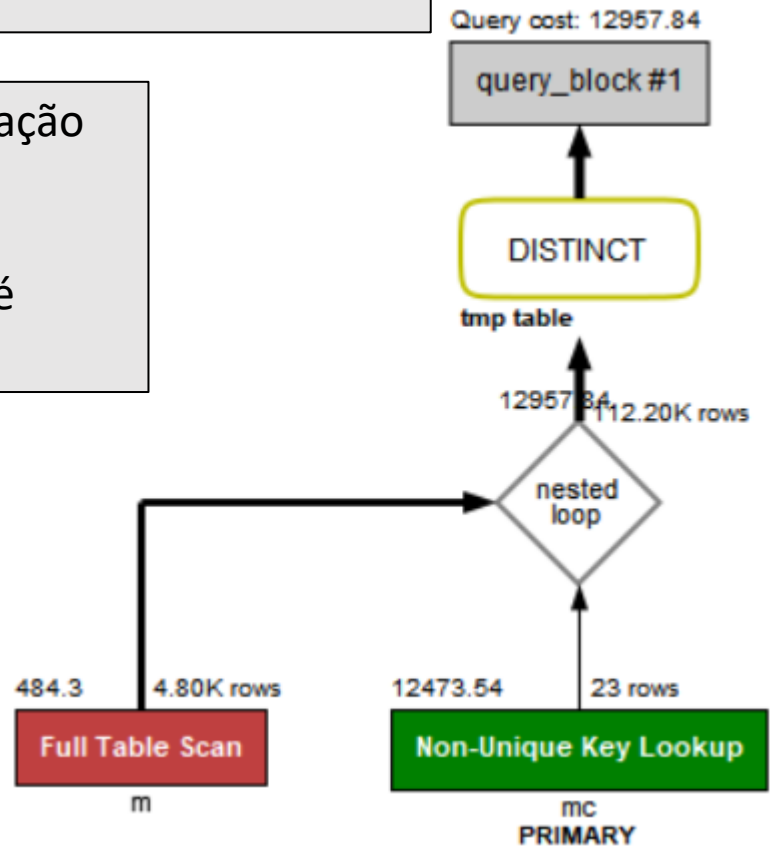
```
SELECT DISTINCT idM, title
FROM movie m
JOIN movie_cast mc ON m.idM = mc.idM
```

Em uma consulta sem semi-junção, o MySQL faz o lookup sem complementação (covering index) e limita a busca ao primeiro registro recuperado (limit)

É um plano parecido ao que usa semi-junção. Mas por algum motivo ainda é mantida uma deduplicação após a junção.

MySQL

-> Table scan on <temporary>
  -> Temporary table with deduplication
    -> Nested loop inner join
      -> Table scan on m
      -> Limit: 1 row(s)
        -> Covering index lookup on mc using PRIMARY (movie_id=m.movie_id)



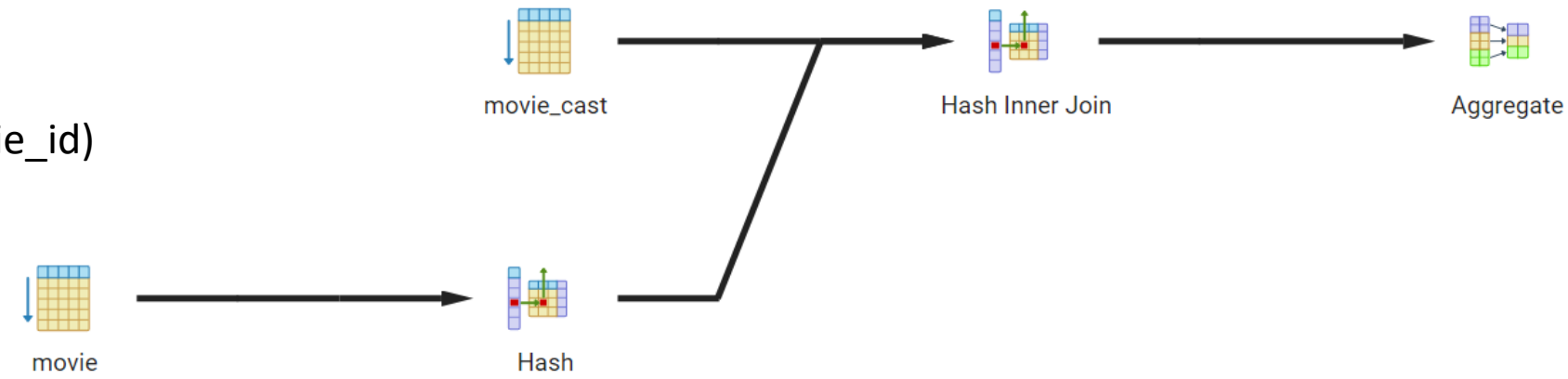
Semi-junção vs junção regular

```
SELECT DISTINCT idM, title
FROM movie m
JOIN movie_cast mc ON m.idM = mc.idM
```

Para a consulta usando junção regular com DISTINCT, o PostgreSQL também seguiu um caminho que exije a deduplicação no final (HashAggregate)

PostgreSQL

```
HashAggregate
  Group Key: m.movie_id, m.title
  ->  Hash Join
        Hash Cond: (mc.movie_id = m.movie_id)
        ->  Seq Scan on movie_cast mc
        ->  Hash
              ->  Seq Scan on movie m
```



Semi-junção vs junção regular

- Na dúvida, use subconsulta
  - deixa claro para o SGBD que se trata de uma semi-junção
  - proporciona ao otimizador uma série de caminhos alternativos até a resposta
  - Quando uma semi-junção é especificada sem usar subconsulta
    - o SGBD pode não reconhecer que se trata de uma semi-junção e deixa de seguir caminhos que seriam interessantes

- Por via das dúvidas, convém testar o plano gerado pelo uso da junção regular

Sumário

- Junção externa
- Semi-junção
  - Semi-junção vs junção regular
  - In vs Exists
- Anti-junção
  - anti-junção vs junção externa
  - Not In vs Not Exists

IN vs EXISTS

- O EXISTS tem um poder de expressão maior do que o IN
  - EXISTS permite usar qualquer tipo de expressão para testar existência
  - IN só permite comparações por igualdade

IN vs EXISTS

- No exemplo
  - Deseja-se descobrir títulos de filmes que tenham personagens cujo nome esteja contido como parte do título
  - Com IN
    - Só podemos saber se o personagem tem o nome igual ao título, mas não se está contido
  - Com EXISTS
    - foi possível verificar se o nome está contido no título aplicando a função CONCAT sobre a variável de correlação

```
SELECT title
FROM movie m WHERE EXISTS
      (SELECT 1 FROM movie_cast mc
      WHERE m.idM = mc.idM AND
      m.title LIKE CONCAT('%', mc.character_name, '%') )
```

```
SELECT title
FROM movie m WHERE
    (idM, title) IN
          (SELECT idM, c_name FROM movie_cast mc )
```

IN vs EXISTS

- E para os casos em que a comparação é por igualdade, qual usar?

- Antigamente
  - o DBA precisava se preocupar com a escolha, pois havia diferenças significativas entre os planos gerados

- Hoje em dia, essa questão não é mais tão relevante
  - Os otimizadores de consulta modernos conseguem escolher um plano otimizado, independente de como a consulta foi escrita

- Mesmo assim, vale a pena investigar se os planos gerados são realmente equivalentes

Sumário

- Junção externa
- Semi-junção
  - Semi-junção vs junção regular
  - In vs Exists
- Anti-junção
  - anti-junção vs junção externa
  - Not In vs Not Exists

Anti-junção

- Considerando que uma junção tem duas partes
  - parte principal
  - parte secundária

- Em uma anti-junção
  - Apenas tuplas da parte principal são retornadas
  - Cada tupla pode ser retornada apenas uma vez
    - Caso não haja correspondências com a parte secundária

Anti-junção

- Em SQL, anti-junções são expressas na forma de subconsultas
  - A parte principal é a consulta externa
  - A parte secundária é a subconsulta

- Os exemplos abaixo mostram duas formas de uso (NOT EXISTS e NOT IN)
  - Parte principal: movie
  - Parte secundária: movie_cast

```
SELECT title
FROM movie m WHERE NOT EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

```
SELECT title
FROM movie WHERE idM NOT IN
        (SELECT idM FROM movie_cast)
```

Anti-junção

- As estratégias de anti-junção podem ser classificadas em
  - Left Anti Join
    - A parte principal fica do lado externo (esquerdo) da junção
  - Right Anti Join
    - A parte principal fica do lado interno (direito) da junção

Anti-junção

- Exemplo: título de filmes que não contenham membros do elenco registrados

```
SELECT title
FROM movie m WHERE NOT EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

- Como essa consulta é representada internamente?

Anti-junção

- A consulta é um exemplo de anti-junção
  - Retornar títulos de filme que **não** contenham membros do elenco
  - Cada filme que satisfaça essa restrição é retornado uma única vez
  - Nenhuma coluna de movie_cast é retornada

```
SELECT title
FROM movie m WHERE NOT EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

Anti-junção

- Neste caso, a consulta também poderia ser expressa usando NOT IN

- Obs. Nem sempre o NOT IN resultado em uma consulta equivalente
  - A presença de valores nulos para movie_cast.idM afeta o resultado

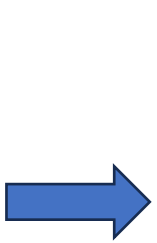
```
SELECT title
FROM movie m WHERE NOT EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

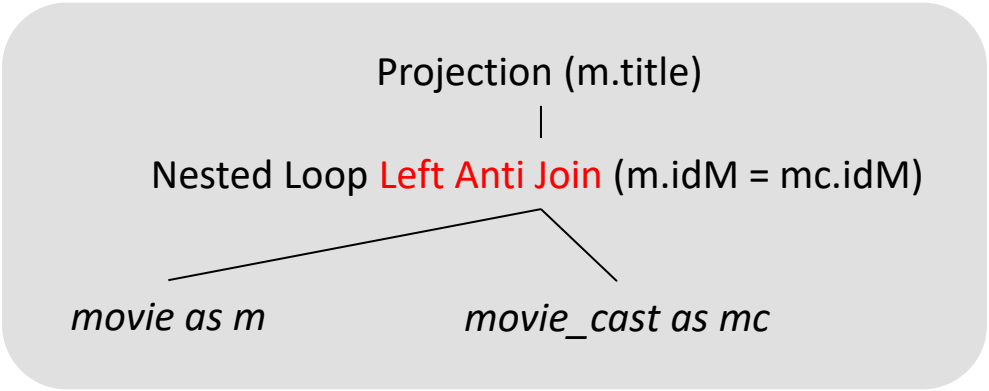
```
SELECT title
FROM movie WHERE idM NOT IN
        (SELECT idM FROM movie_cast)
```

Anti-junção

- O plano abaixo usa um Nested Loop Left Anti Join
  - A parte principal fica no lado esquerdo (left) da junção
    - Apenas registros da parte principal podem ser retornados
  - A junção não busca correspondências
    - Apenas verifica se elas existem
      - Por isso pouco importa o que foi usado no SELECT da subconsulta

```
SELECT title
FROM movie m WHERE NOT EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```

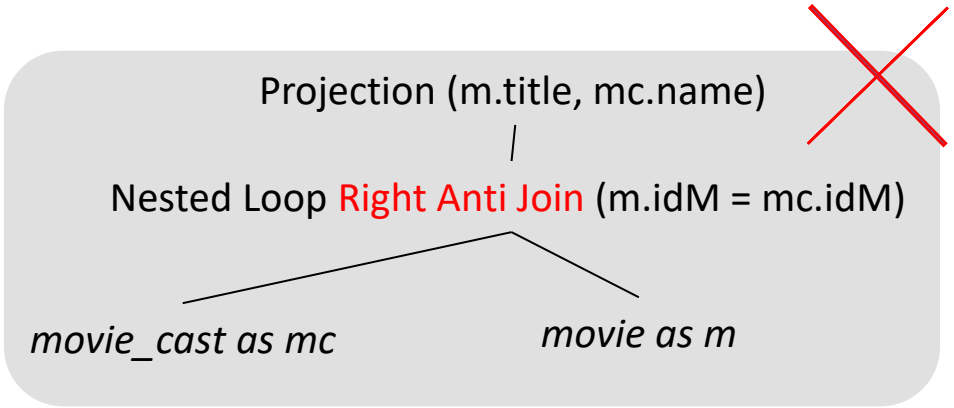




Anti-junção

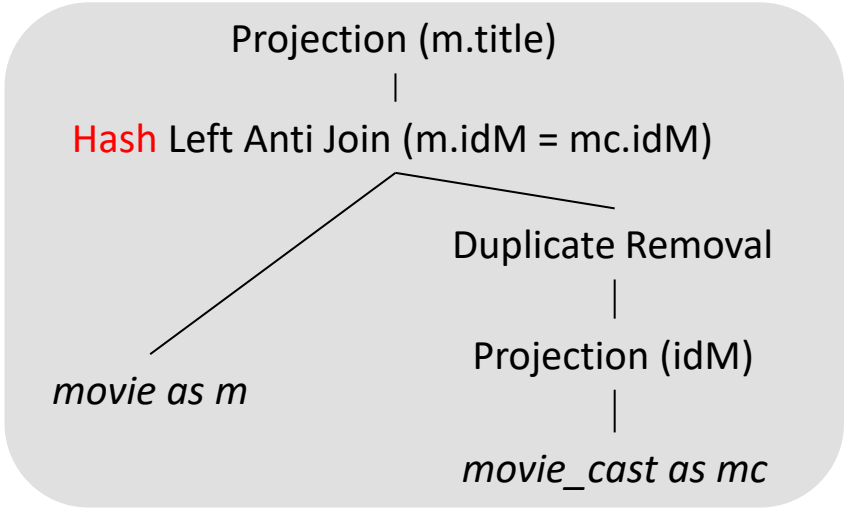
- Limitação do Nested Loop para anti junção
  - A parte principal deve sempre ficar do lado esquerdo
  - Ou seja, não existe o Nested Loop Right Anti Join
  - Para ter maior flexibilidade, o SGBDs pode oferecer outras estratégias, como variações do Hash Join

```
SELECT title
FROM movie m WHERE EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM )
```



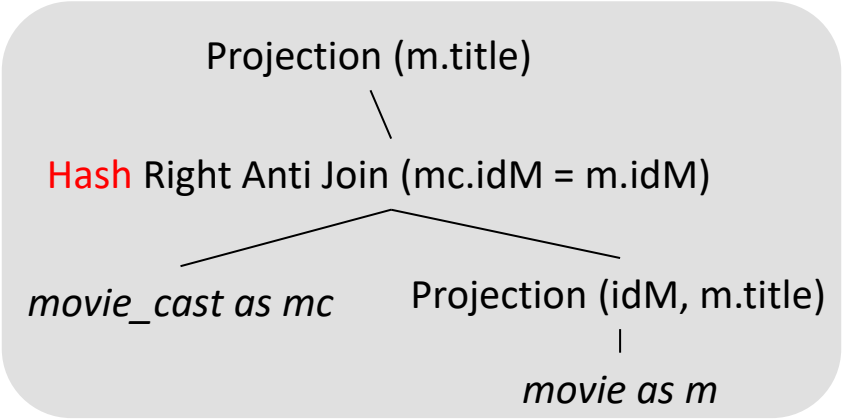
Anti-junção

- O plano abaixo usa um **Hash Left Anti Join**
  - A parte principal fica na esquerda (left)
  - A parte secundária é armazenada em uma tabela hash, contendo apenas idMs únicos
    - A verificação do anti Join é feita sobre a tabela hash



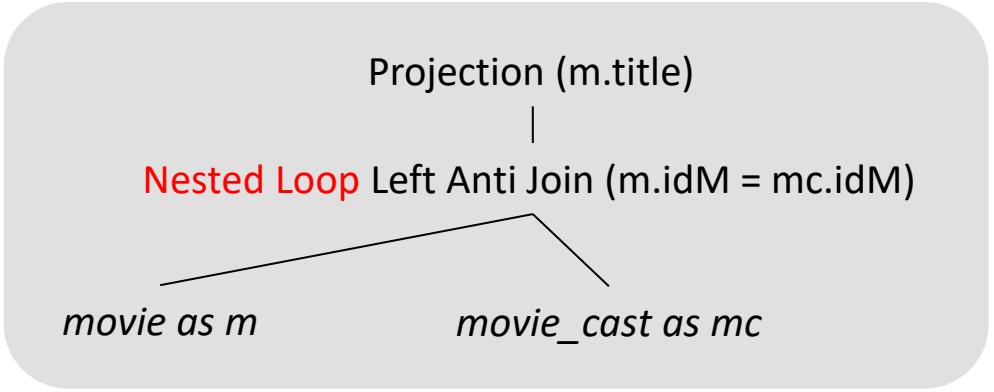
Anti-junção

- O plano abaixo usa um **Hash Right Anti Join**
  - A parte principal fica na direita (right)
  - A parte principal é armazenada em uma tabela hash

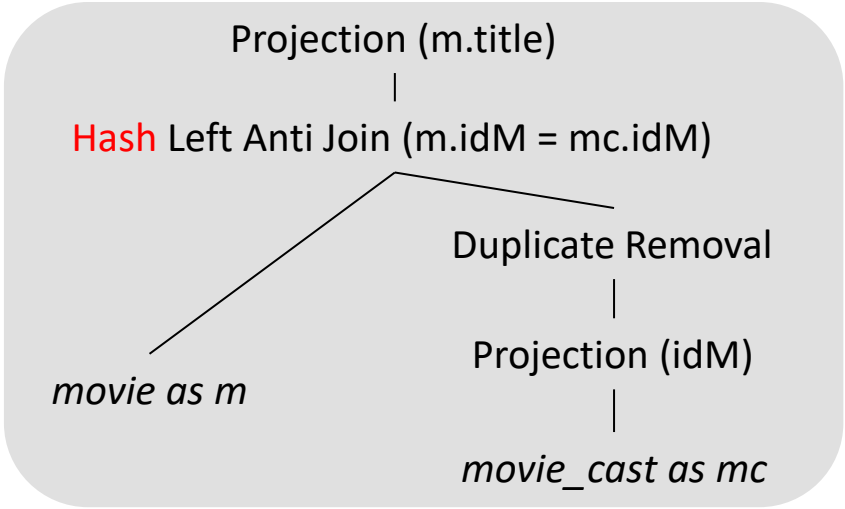


Anti-junção

- Os planos A e B mostram as duas possiblidades de Left Anti Join
  - Plano A: com Nested Loop Left Anti Join
  - Plano B: com Hash Left Anti Join
- Qual deles é mais eficiente?



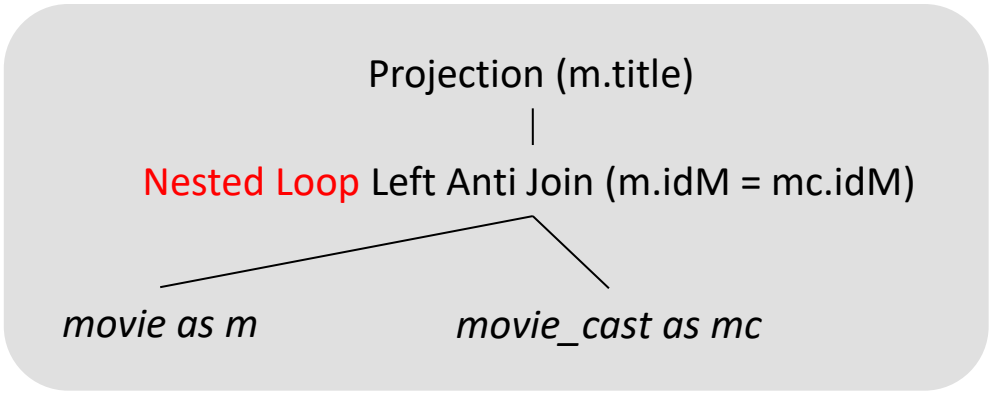
Plano A



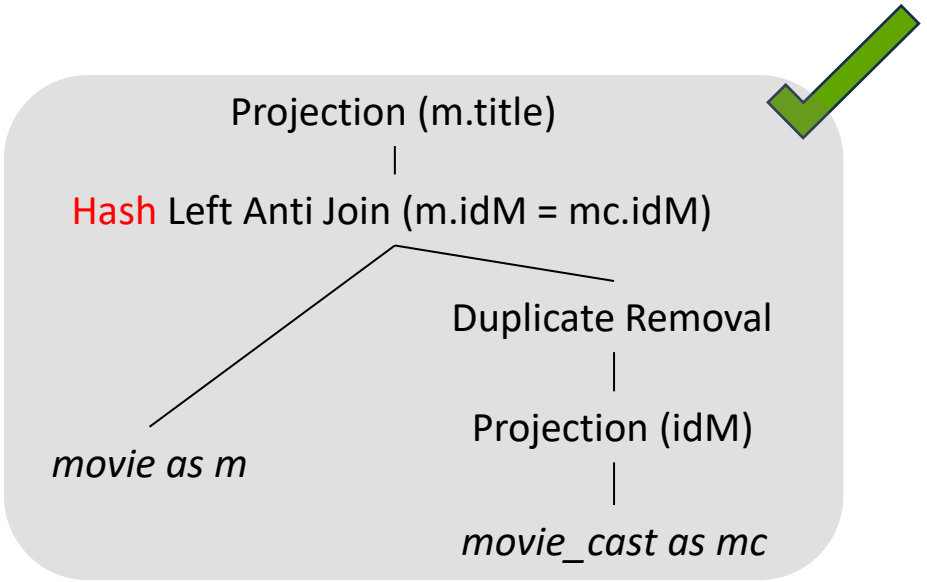
Plano B

Anti-junção

- Plano B é mais eficiente
  - A busca da junção é feita em uma tabela hash
  - Contudo, consome memória



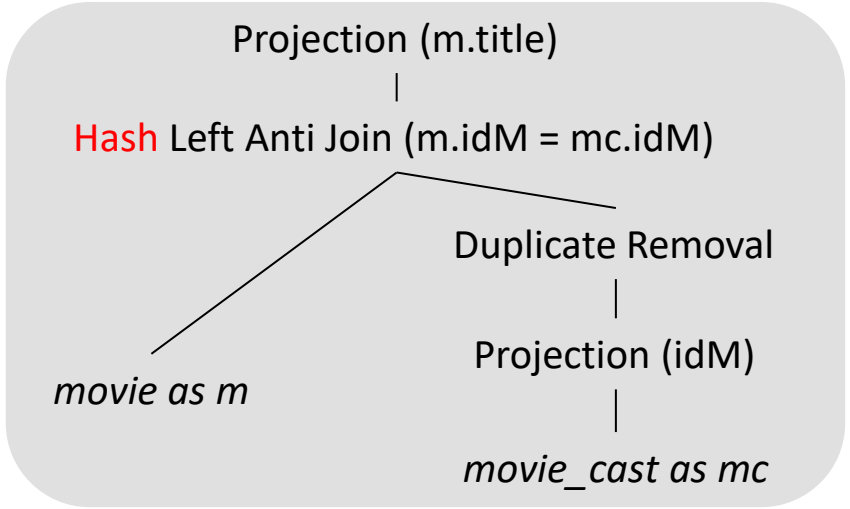
Plano A



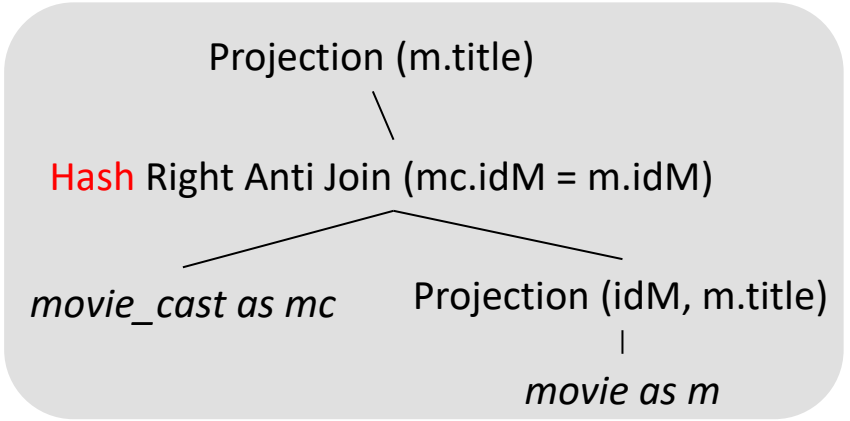
Plano B

Anti-junção

- Os planos A e B mostram as duas possiblidades de Hash Anti Join
  - Plano A: com a parte principal no lado esquerdo
  - Plano B: com a parte principal do lado direito
- Qual deles consome menos memória?



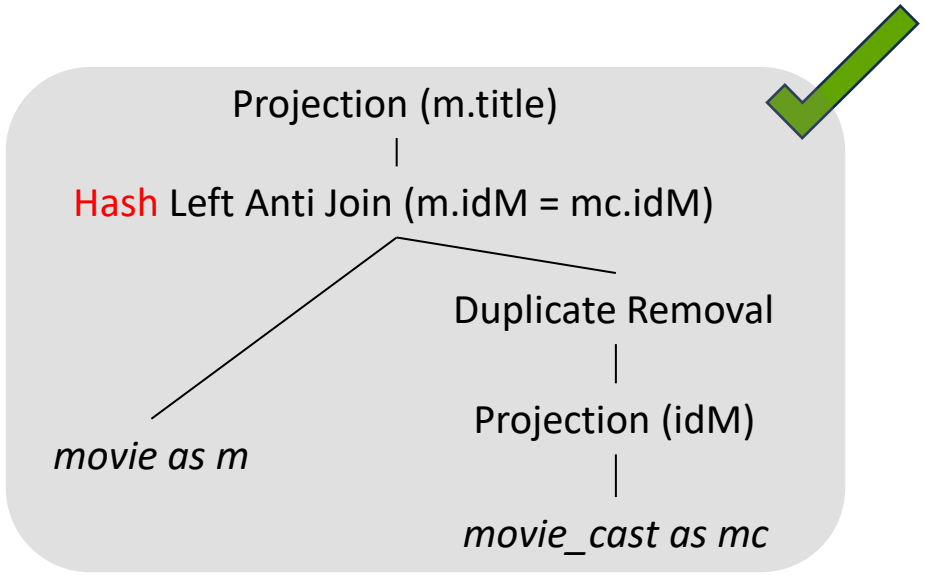
Plano A



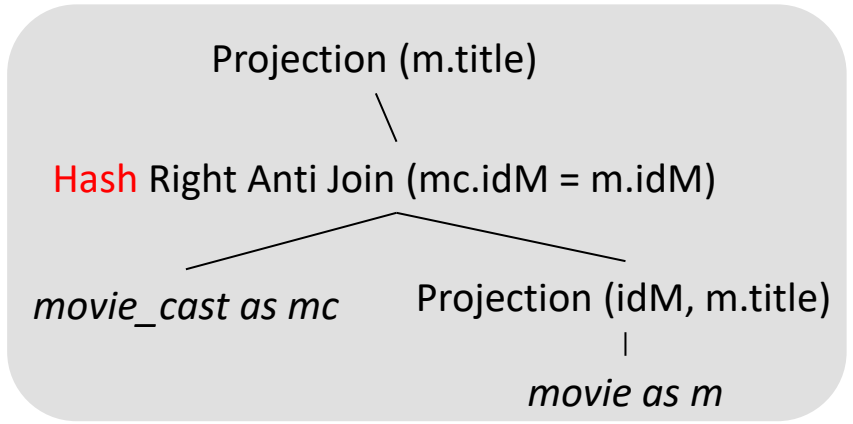
Plano B

Anti-junção

- O plano A consome menos memória
- Mesmo com a tabela movie_cast tendo mais registros
  - Apenas idMs únicos são materializados
- Além disso, nenhuma coluna textual precisa ser materializada



Plano A



Plano B

Anti-junção

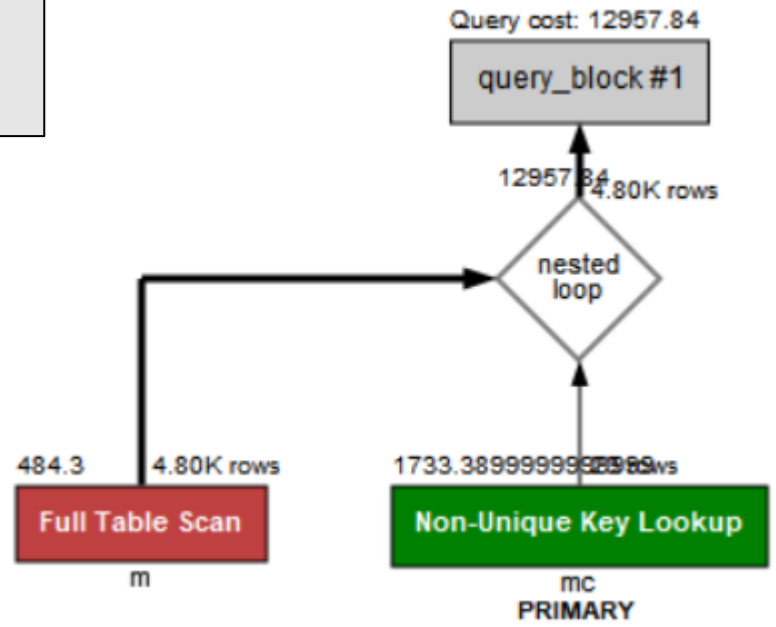
```
SELECT title
FROM movie m WHERE NOT EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM)
```

O MySQL usou uma anti-junção com o Nested Loop

O covering index indica que não foi feita a complementação

MySQL

-> Nested loop antijoin
   -> Table scan on m
   -> Covering index lookup on mc using PRIMARY (movie_id=m.movie_id)



Anti-junção

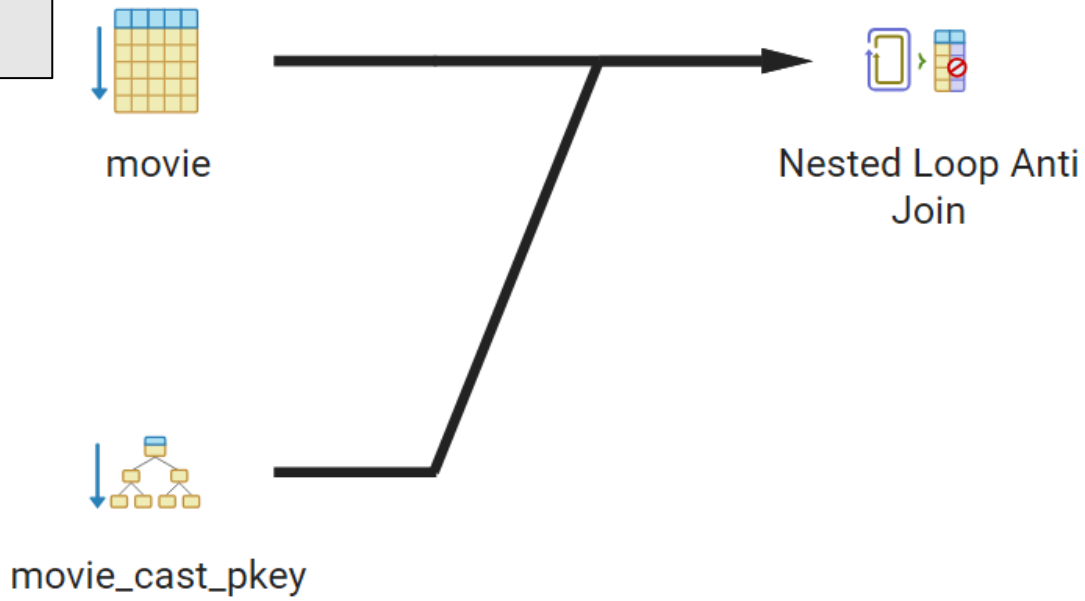
```
SELECT title
FROM movie m WHERE NOT EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.idM = mc.idM)
```

O PostgreSQL também usa o Nested Loop Anti Join

O index only scan indica que nao foi feita a complementação

PostgreSQL

Nested Loop Anti Join
-> Seq Scan on movie m
-> Index Only Scan using movie_cast_pkey on movie_cast mc
Index Cond: (movie_id = m.movie_id)



Anti-junção

```
SELECT title
FROM movie m WHERE NOT EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.movie_id = mc.movie_id AND cast_order > 220)
```

Colocando um filtro seletivo na parte secundária, o PostgreSQL usou o Hash Right Anti Join

Assim, o filtro foi mantido no lado externo da consulta

PostgreSQL

Hash Right Anti Join

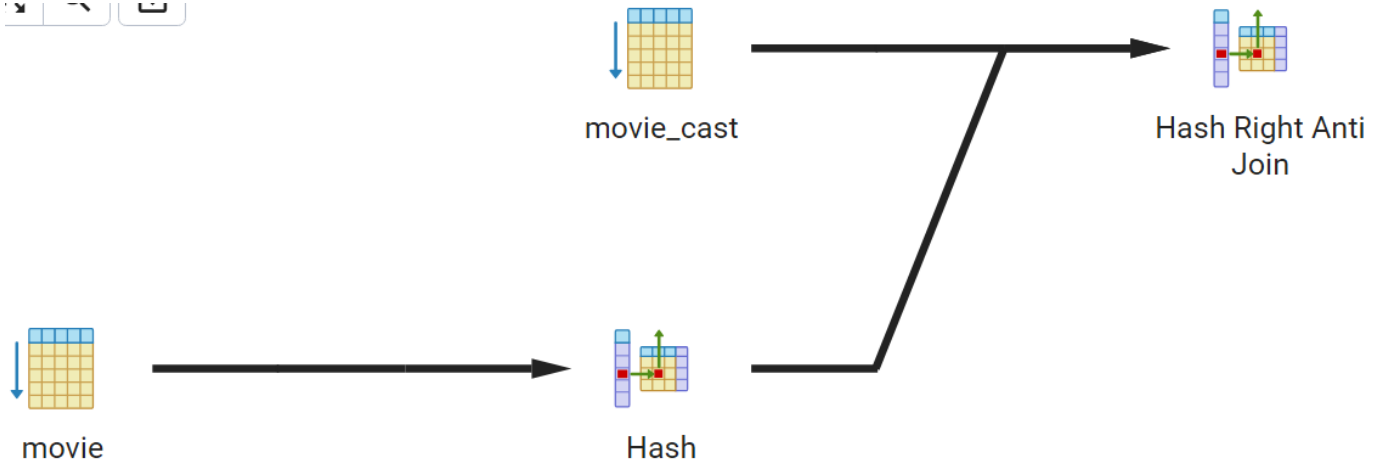
Hash Cond: (mc.movie_id = m.movie_id)

-> Seq Scan on movie_cast mc

Filter: (cast_order > 220)

-> Hash

-> Seq Scan on movie m



Sumário

- Junção externa
- Semi-junção
  - Semi-junção vs junção regular
  - In vs Exists
- Anti-junção
  - anti-junção vs junção externa
  - Not In vs Not Exists

Anti-junção vs junção externa

- Como vimos, uma anti-junção é expressa por meio de uma subconsulta

```
SELECT title
FROM movie WHERE title NOT IN
      (SELECT c_name FROM movie_cast)
```

Com anti-junção

Anti-junção vs junção externa

- Uma anti-junção também pode ser expressa como uma junção externa

```
SELECT title
FROM movie WHERE title NOT IN
      (SELECT c_name FROM movie_cast)
```

Com anti-junção

```
SELECT title
FROM movie m
LEFT JOIN movie_cast mc ON m.idM = mc.idM
WHERE mc.idM IS NULL
```

Sem anti-junção

Anti-junção vs junção externa

- Quando uma junção externa é equivalente a uma anti-junção?
  - O SELECT retorna apenas dados do lado principal(movie)
  - A cláusula WHERE remove todos os filmes que tenham associação com movie_cast
    - mc.idM is NULL

```
SELECT title
FROM movie WHERE title NOT IN
      (SELECT c_name FROM movie_cast)
```

Com anti-junção

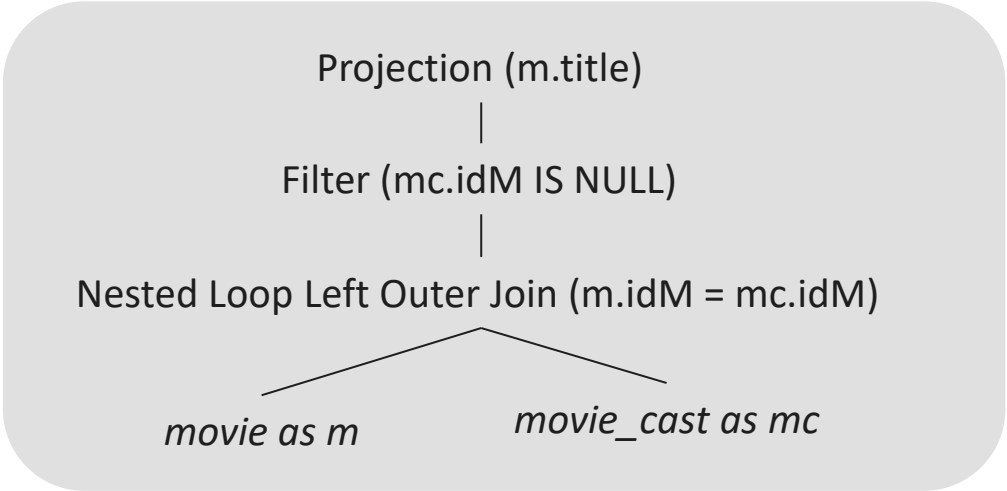
```
SELECT title
FROM movie m
LEFT JOIN movie_cast mc ON m.idM = mc.idM
WHERE mc.idM IS NULL
```

Sem anti-junção

Anti-junção vs junção externa

- No plano de execução usando junção externa
  - Um filme precisa cruzar com todos os seus movie_casts correspondentes
  - Filmes sem correspondência têm colunas preenchidas com nulo
  - Em seguida, todos as correspondências identificadas são descartadas
    - Ou seja, todo o trabalho de localização das correspondências é desfeito

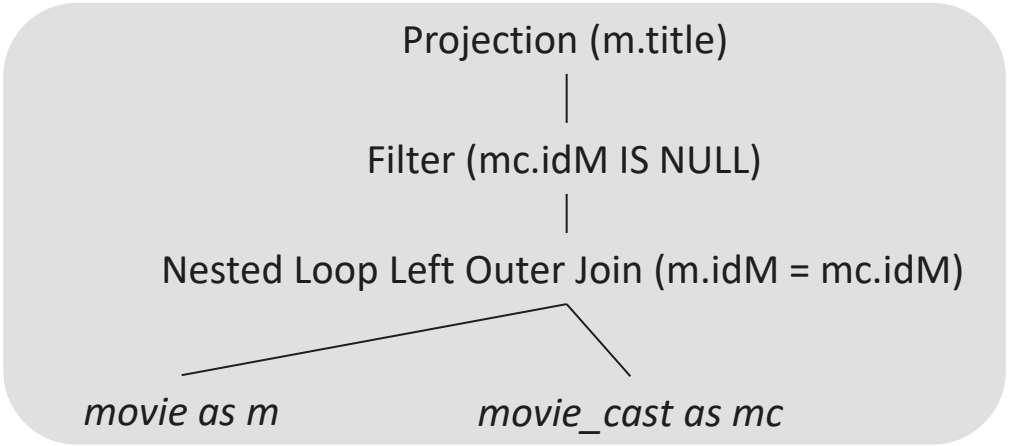
```
SELECT title
FROM movie m
LEFT JOIN movie_cast mc ON m.idM = mc.idM
WHERE mc.idM IS NULL
```



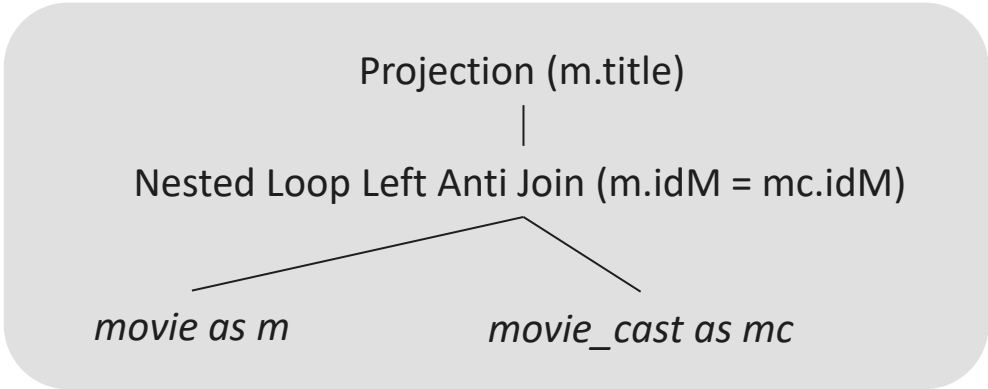
Plano A

Anti-junção vs junção externa

- Qual estratégia é melhor?



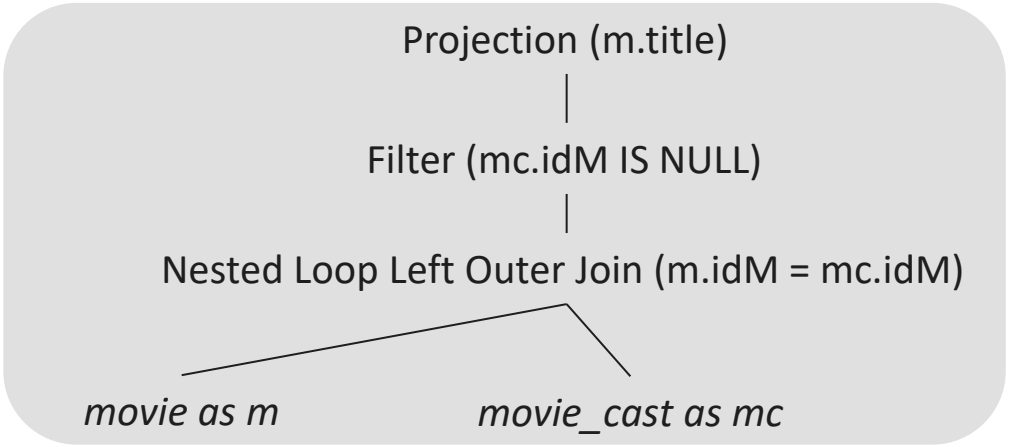
Plano A (com junção externa)



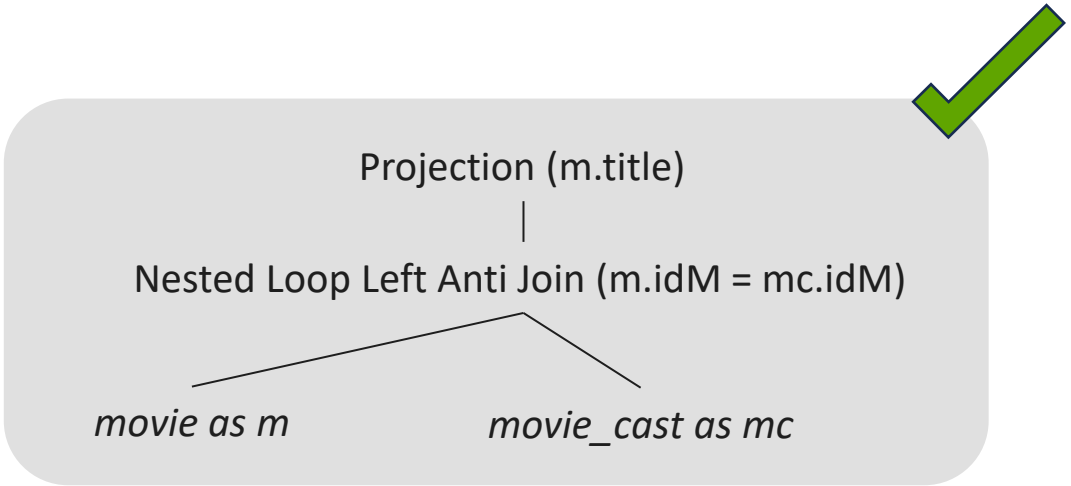
Plano B (com subconsulta)

Anti-junção vs junção externa

- Plano B é melhor
  - Para cada filme, ocorre uma verificação no lado interno (exists)
    - Se não existir correspondência, o filme é retornado
  - Em nenhum momento é realizado algum cruzamento
    - Isso evita o overhead de localização e posterior remoção via filtro



Plano A (com junção externa)



Plano B (com subconsulta)

Anti-junção vs junção externa

```
SELECT m.title
FROM movie m LEFT JOIN movie_cast mc ON m.idM = mc.idM
WHERE mc.idM IS NULL
```

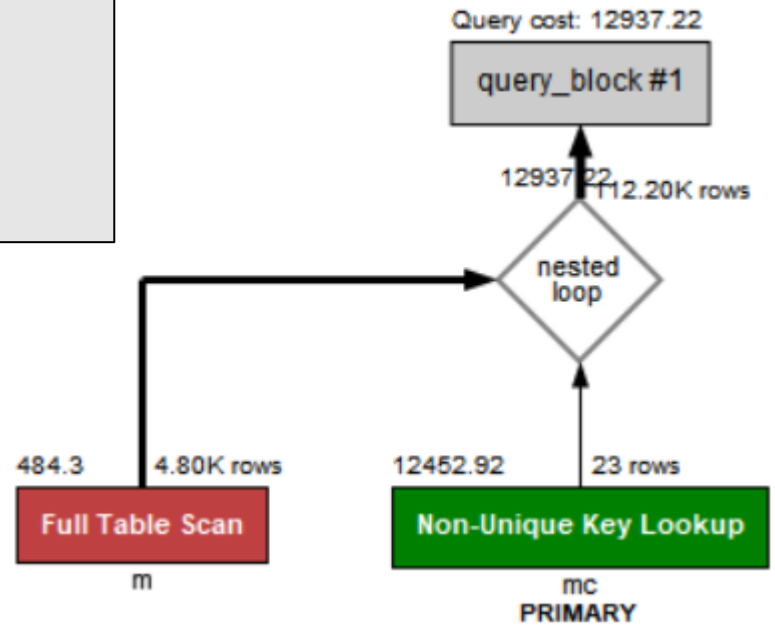
Na consulta com junção externa, o MySQL reconheceu que se tratava de uma anti-junção e usou o algoritmo mais adequado (anti-join).

Covering index indica que a complementação não foi feita

Curiosamente, foi mantido o filtro no final, mesmo sem necessidade

MySQL

-> Filter: (mc.movie_id is null)
  -> Nested loop antijoin
    -> Table scan on m
    -> Covering index lookup on mc using PRIMARY (movie_id=m.movie_id)



Anti-junção vs junção externa

```
SELECT m.title
FROM movie m LEFT JOIN movie_cast mc ON m.idM = mc.idM
WHERE mc.idM IS NULL
```

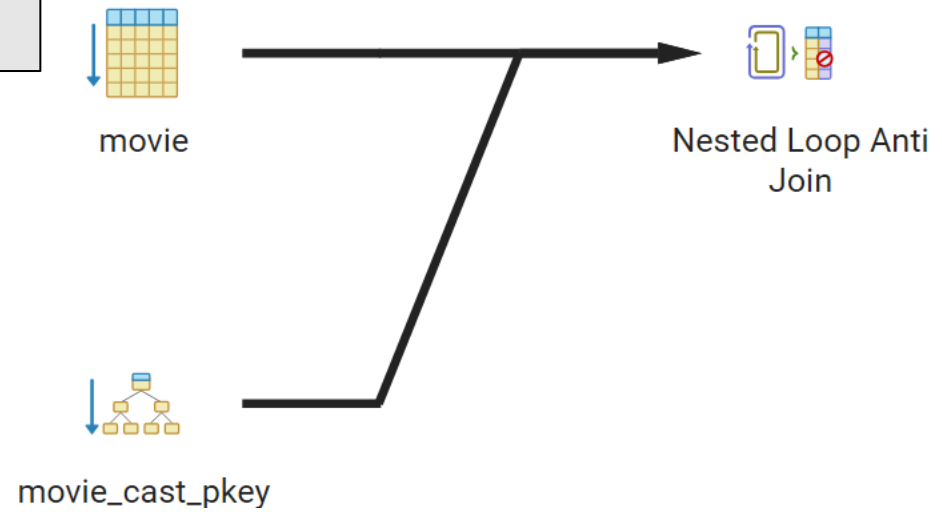
O PostgreSQL também reconheceu que se tratava de uma anti-junção

Index only scan indica que a complementação não foi feita

Diferentemente do MySQL, o filtro de nulo não foi executado

PostgreSQL

Nested Loop Anti Join
-> Seq Scan on movie m
-> Index Only Scan using movie_cast_pkey on movie_cast mc
Index Cond: (movie_id = m.movie_id)



Anti-junção vs junção externa

- O exemplo anterior mostrou que os SGBDs reconheceram a anti-junção escrita como uma junção externa
  - Porém, isso nem sempre ocorre

- Na dúvida, use subconsulta
  - deixa claro para o SGBD que se trata de uma anti-junção
  - proporciona ao otimizador uma série de caminhos alternativos até a resposta
  - Quando uma anti-junção é especificada sem usar subconsulta
    - o SGBD pode não reconhecer que se trata de uma semi-junção e deixa de seguir caminhos que seriam interessantes

- Mesmo assim, convém testar o plano gerado pelo uso da junção externa

Sumário

- Junção externa
- Semi-junção
  - Semi-junção vs junção regular
  - In vs Exists
- Anti-junção
  - anti-junção vs junção externa
  - Not In vs Not Exists

Anti-junção - NOT IN vs NOT EXISTS

- NOT EXISTS tem um poder de expressão maior do que o NOT IN
  - Assim como ocorre na relação entre EXISTS e IN

- Mas qual é mais eficiente?

Anti-junção - NOT IN vs NOT EXISTS

- Otimizadores de consulta modernos são capazes de reconhecer casos em que NOT IN e NOT EXISTS são equivalentes
  - Isso permite que sejam gerados os mesmos planos de execução

  - Porém, em alguns casos, o uso do NOT IN implica em uma solução que empregue materialização
    - Por quê?
      - Devido a semântica do NOT IN

Anti-junção - NOT IN vs NOT EXISTS

- NOT IN é **semanticamente** diferente de NOT EXISTS

- O NOT IN realiza comparações contra uma lista de valores calculada pela consulta
  - Caso essa lista contenha algum valor nulo
    - O NOT IN retornará um resultado vazio

Anti-junção - NOT IN vs NOT EXISTS

- Exemplo
  - Retornar filmes cujo título **não seja** o nome de nenhum personagem
  - Existem três personagens registrados
    - Rambo, Rocky, Alien
  - Com NOT IN, quais filmes seriam retornados?

```
SELECT title
FROM movie WHERE title NOT IN
      (SELECT c_name FROM movie_cast)
```

title	Retorna?
Rambo	
Forrest Gump	
Mad Max	

Lado externo

c_name
Rambo
Rocky
Alien

Lado interno

Anti-junção - NOT IN vs NOT EXISTS

- Forrest Gump e Mad Max não são nomes de personagens
- Por isso são retornados

```
SELECT title
FROM movie WHERE title NOT IN
      (SELECT c_name FROM movie_cast)
```

title	Retorna?
Rambo	Não
Forrest Gump	Sim
Mad Max	sim

Lado externo

c_name
Rambo
Rocky
Alien

Lado interno

Anti-junção - NOT IN vs NOT EXISTS

- E caso algum dos personagens não tivesse valor definido?

```
SELECT title
FROM movie WHERE title NOT IN
      (SELECT c_name FROM movie_cast)
```

title	Retorna?
Rambo	
Forrest Gump	
Mad Max	

Lado externo

c_name
Rambo
Rocky
null

Lado interno

Anti-junção - NOT IN vs NOT EXISTS

- E caso algum dos personagens não tivesse valor definido?
- Nesse caso, nenhum filme é retornado
  - Qualquer comparação com nulos é tratada como incerta pelo SGBD
  - E informações incertas nunca são retornadas

```
SELECT title
FROM movie WHERE title NOT IN
      (SELECT c_name FROM movie_cast)
```

title	Retorna?
Rambo	Não
Forrest Gump	Não
Mad Max	Não

Lado externo

c_name
Rambo
Rocky
null

Lado interno

Anti-junção - NOT IN vs NOT EXISTS

- Para alguns motores de execução, essa verificação de nulos é facilitada se o lado interno for materializado
- Por isso
  - o NOT IN normalmente leva a um plano com materialização

```
SELECT title
FROM movie WHERE title NOT IN
      (SELECT c_name FROM movie_cast)
```

title	Retorna?
Rambo	Não
Forrest Gump	Não
Mad Max	Não

Lado externo

c_name
Rambo
Rocky
null

Lado interno

Anti-junção - NOT IN vs NOT EXISTS

- Usando NOT EXISTS, esse comportamento não ocorre
  - Os nulos do lado interno nunca são comparados com tuplas do lado externo

```
SELECT title
FROM movie m WHERE NOT EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.title = mc.c_name )
```

title	Retorna?
Rambo	Não
Forrest Gump	Sim
Mad Max	Sim

Lado externo

c_name
Rambo
Rocky
null

Lado interno

Anti-junção - NOT IN vs NOT EXISTS

- Isso dá mais liberdade ao otimizador para analisar se a materialização compensa ou não

```
SELECT title
FROM movie m WHERE NOT EXISTS
        (SELECT 1 FROM movie_cast mc
         WHERE m.title = mc.c_name )
```

title	Retorna?
Rambo	Não
Forrest Gump	Sim
Mad Max	Sim

Lado externo

c_name
Rambo
Rocky
null

Lado interno

Anti-junção - NOT IN vs NOT EXISTS

- Alternativamente, pode-se fazer o NOT IN se comportar como o NOT EXISTS adicionando um filtro para NULO na subconsulta

```
SELECT title
FROM movie WHERE title NOT IN
      (SELECT c_name FROM movie_cast
       AND c_name IS NOT NULL)
```

title	Retorna?
Rambo	Não
Forrest Gump	Sim
Mad Max	Sim

Lado externo

c_name
Rambo
Rocky
null

Lado interno

Anti-junção - NOT IN vs NOT EXISTS

- Agora as duas consultas sempre trarão resultados equivalentes
- Contudo, será que os planos de execução serão iguais?
- Essa verificação fica como exercício

```
SELECT title
FROM movie m WHERE NOT EXISTS
   (SELECT 1 FROM movie_cast mc
         WHERE m.title = mc.character_name)
```

```
SELECT title
FROM movie WHERE title NOT IN
         (SELECT c_name FROM movie_cast
          WHERE c_name IS NOT NULL)
```

Em resumo

- Nested Loop Join sempre deixa a parte principal do lado externo
- Já o Hash Join possui maior flexibilidade
  - Qualquer parte pode aparecer do lado externo
- Se a consulta possuir filtro na parte secundária
  - Pode-se usar variações do Hash Join para manter esse filtro do lado externo
    - Contudo, isso acarreta em maior consumo de memória
- O PostgreSQL usa Hash Join (e suas variações) para uma gama maior de situações em relação ao MySQL

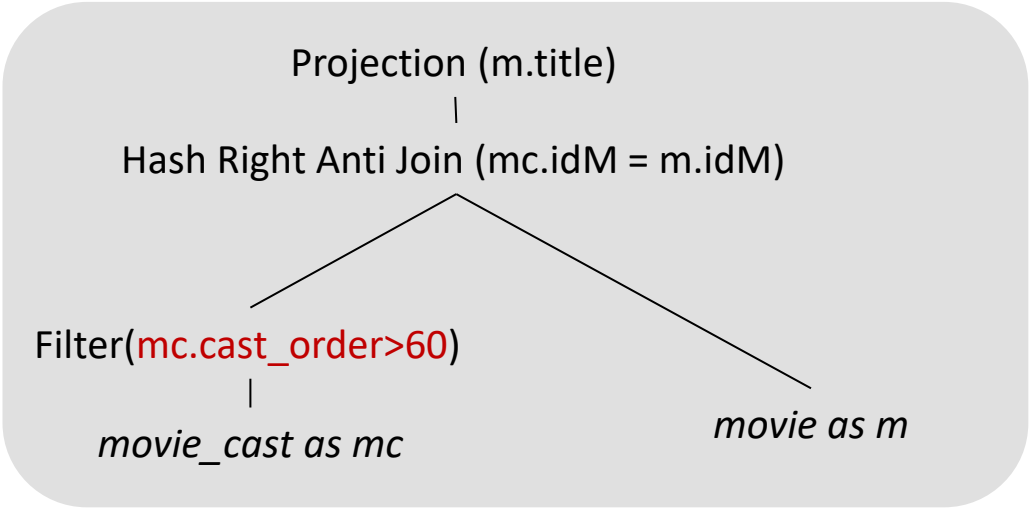
Atividade Individual

- A consulta abaixo pede títulos de filmes que não possuam mais do que 60 membros do elenco.

```
SELECT title
FROM movie WHERE idM NOT IN
        (SELECT idM FROM movie_cast
         WHERE cast_order > 60)
```

Atividade Individual

- O plano abaixo pode ser usado no DBest para encontrar filmes que não tenham membros de elenco com mais do que 60 personagens



Plano A

Atividade Individual

- O objetivo é encontrar um plano alternativo (plano B)
  - Esse plano não deve acessar mais páginas do que o plano A
  - E deve apresentar um consumo de memória menor

Plano B