Ordem das junções

Ordem das junções

- Em uma junção
  - Qual tabela deixar do lado externo?

- E quando há mais do duas tabelas?
  - No exemplo abaixo, há três tabelas: movie, movie_cast, person
    - Qual a melhor ordem para as junções?

```
SELECT m.title, p.name
FROM movie m JOIN movie_cast mc ON m.movie_id = mc.movie_id
JOIN person p ON mc.person_id = p.person_id
```

Ordem das junções

- Tanto o Nested Loop Join quanto o Hash Join visam reduzir o esforço na busca por correspondências
  - Quanto menos páginas for necessário transferir do disco, melhor

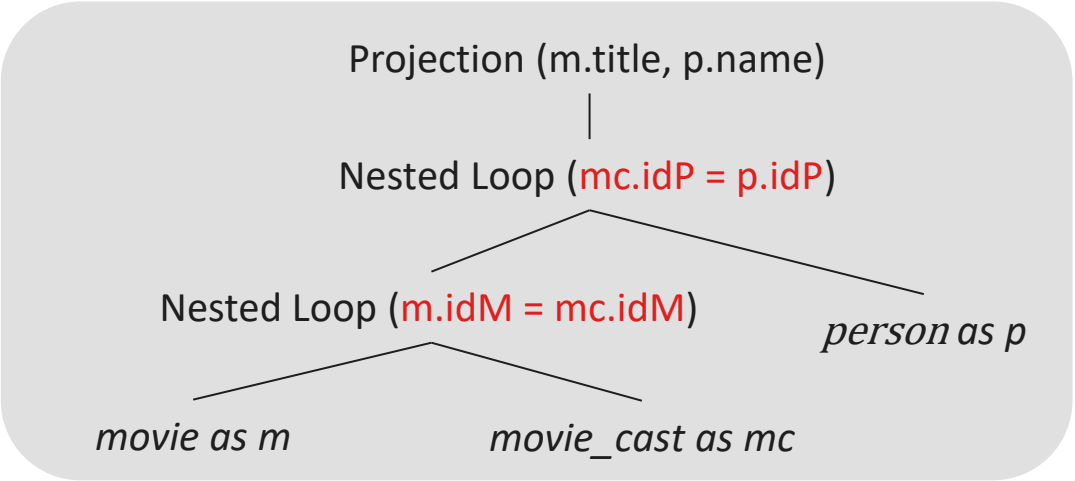
- O Hash Join ainda tem uma preocupação adicional
  - Reduzir consumo de memória

Sumário

- Critérios usados
  - correspondência direta
  - reduzir transferências de páginas/consumo de memória
- Mais do que duas tabelas
  - Nested Loop
  - Hash Join
- Influência de filtros em junções
  - otimização push down
  - mudança do lado das tabelas
  - mudança do algoritmo de junção

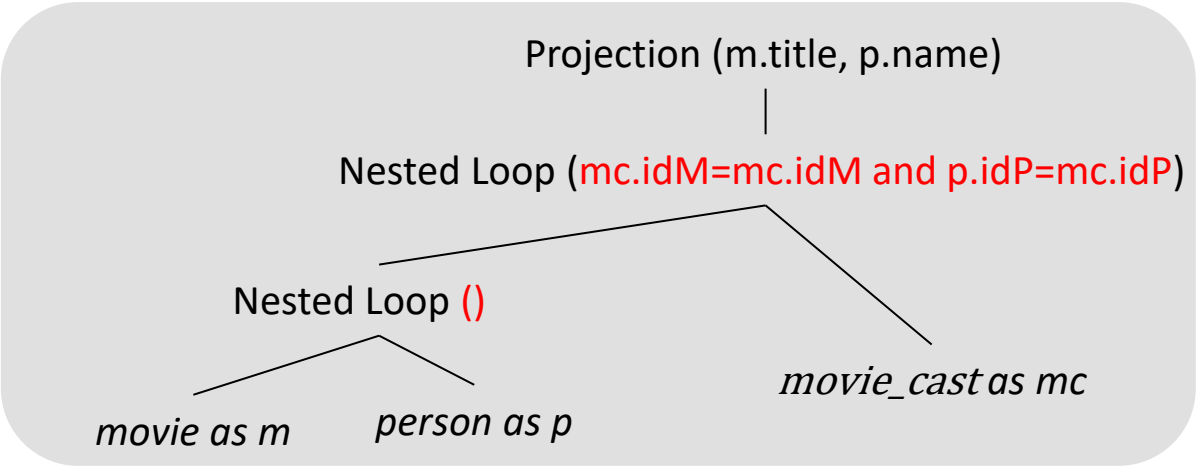
Correspondência Direta

- No plano abaixo
  - As duas junções possuem algum predicado de correspondência
    - m.idM = mc.idM
    - mc.idP = p.idP



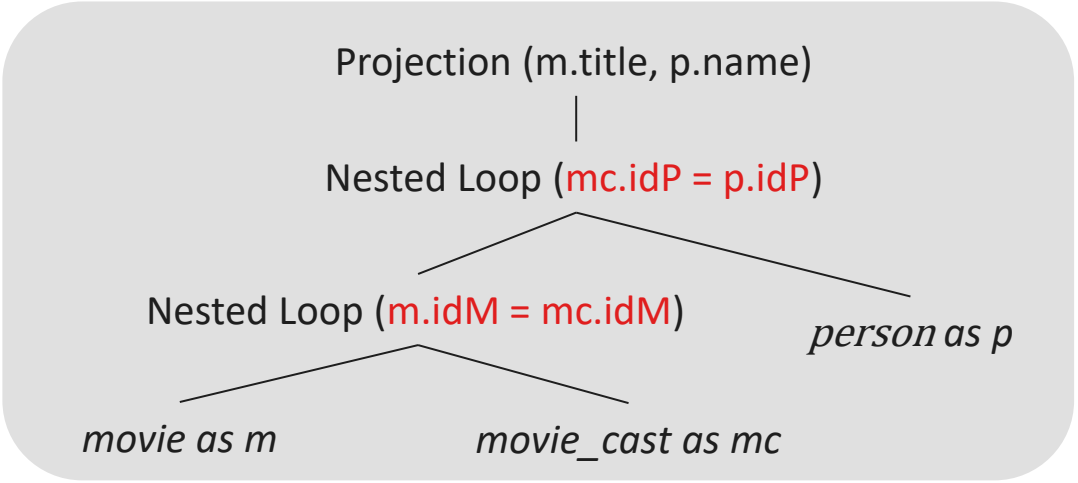
Correspondência Direta

- Neste outro plano
  - A primeira junção não possui predicado de junção
  - Isso gera um número grande de tuplas
    - produto cartesiano entre movie e person
  - Levando a um aumento considerável de seeks em movie_cast

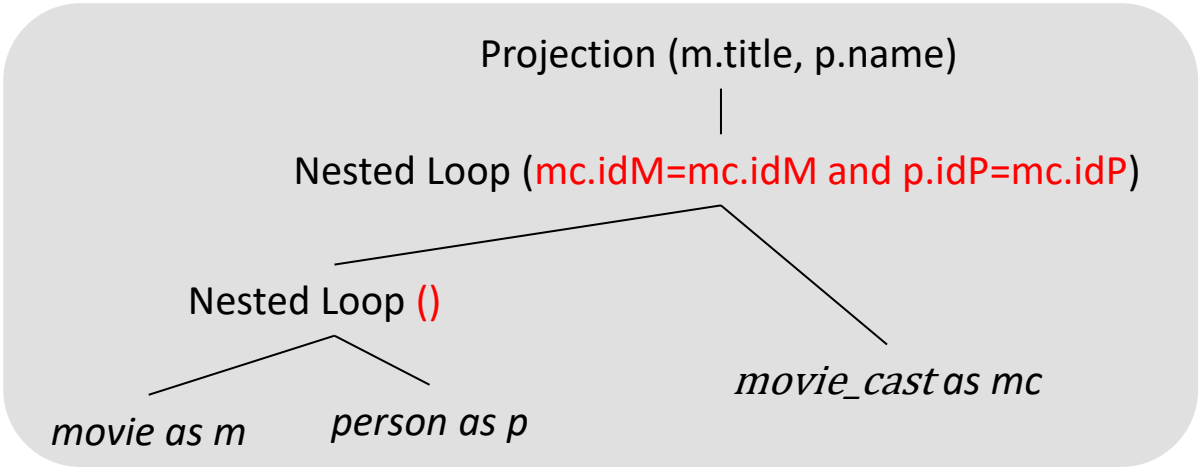


Correspondência Direta

- Qual é melhor?



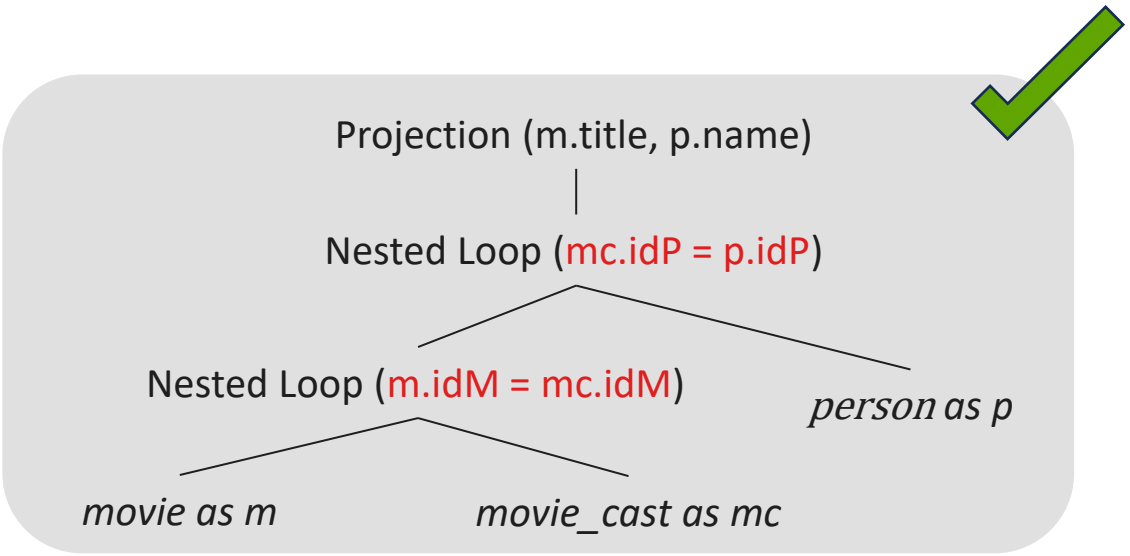
Plano A



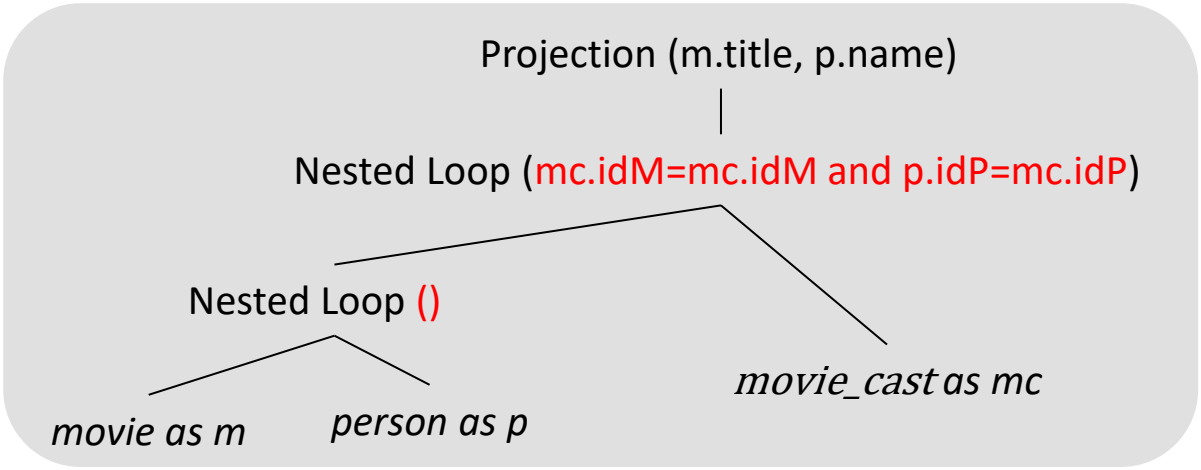
Plano B

Correspondência Direta

- É melhor seguir um caminho que use os critérios de junção
- Isso evita a geração do produto cartesiano



Plano A



Plano B

Sumário

- Critérios usados
  - correspondência direta
  - reduzir transferências de páginas/consumo de memória
- Mais do que duas tabelas
  - Nested Loop
  - Hash Join
- Influência de filtros em junções
  - otimização push down
  - mudança do lado das tabelas
  - mudança do algoritmo de junção

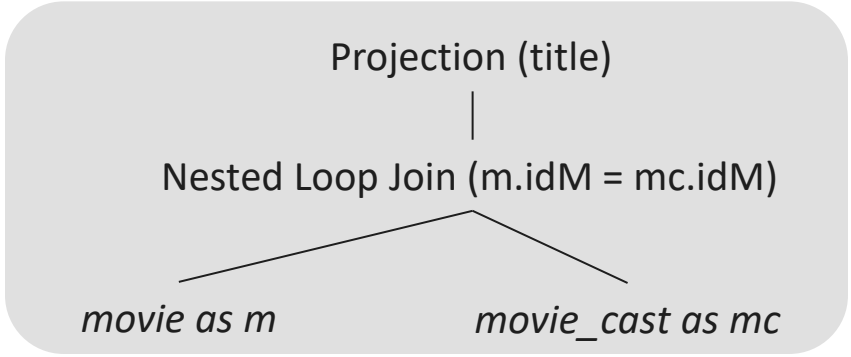
Reduzir transferências de páginas/consumo de memória

- Regra geral
  - Realizar junções entre tabelas que estão conectadas por predicados de junção

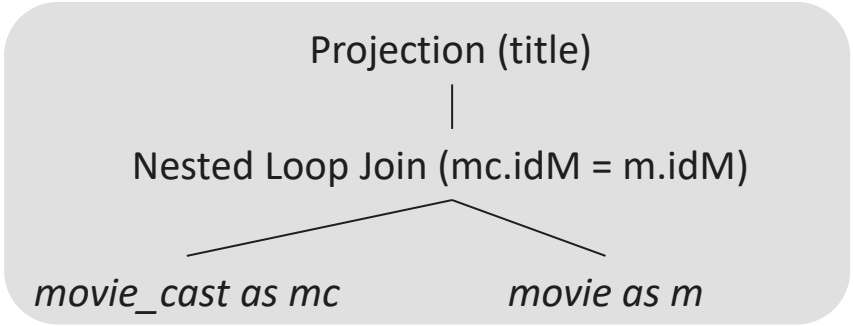
- Mas qual tabela deixar do lado externo?
  - Nested Loop Join
    - se preocupa em reduzir o número de seeks
    - para isso, é importante manter a menor tabela do lado externo da junção
  - Hash Join
    - se preocupa mais em reduzir o consumo de memória
    - para isso, é importante manter a menor tabela do lado interno da junção

Reduzir transferências de páginas/consumo de memória

- Os dois planos usam Nested Loop Join
  - Plano A: Um seek para cada movie
  - Plano B: Um seek para cada movie_cast
- Qual é melhor?



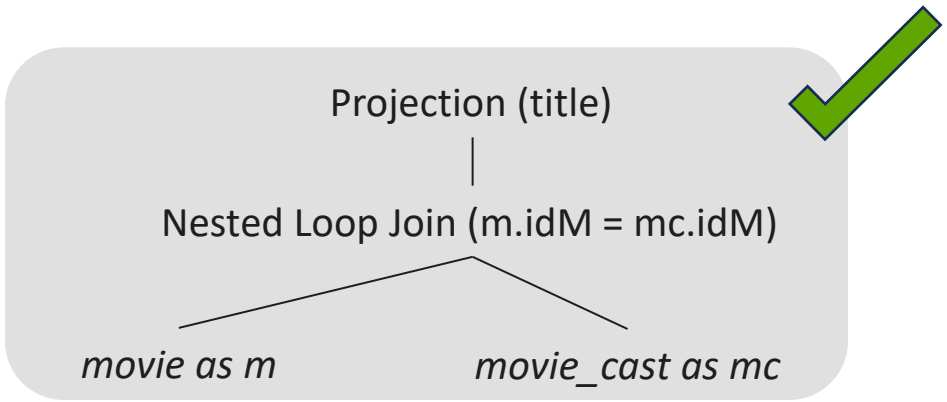
Plano A



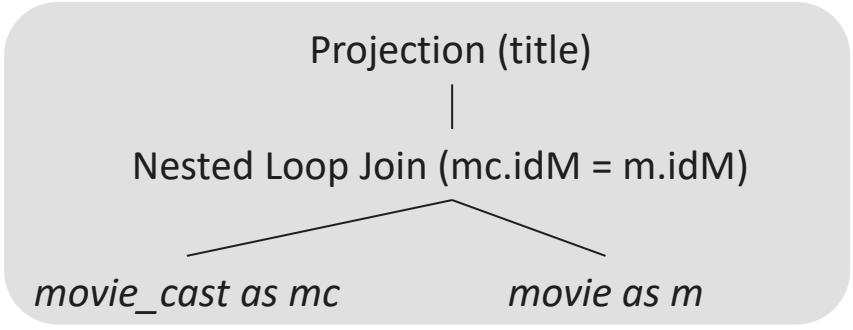
Plano B

Reduzir transferências de páginas/consumo de memória

- Os dois planos usam Nested Loop Join
  - Plano A: Um seek para cada movie
  - Plano B: Um seek para cada movie_cast
- O plano A é melhor
  - Menos seeks são realizados



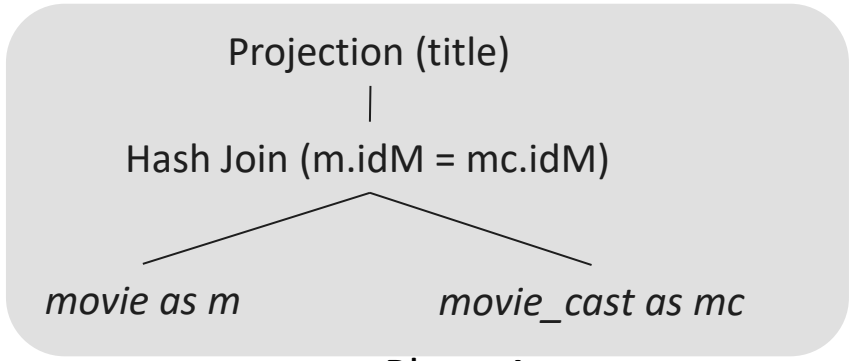
Plano A



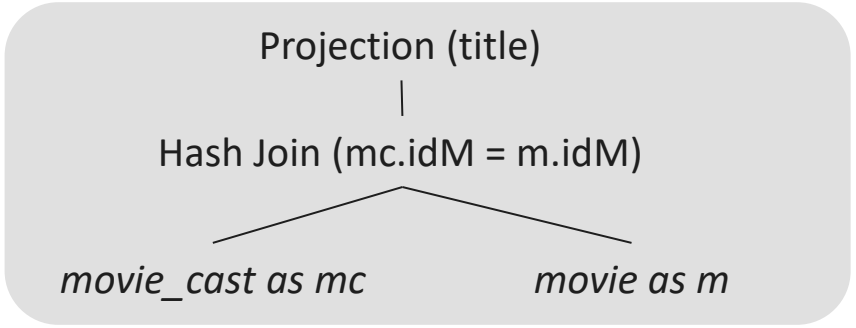
Plano B

Reduzir transferências de páginas/consumo de memória

- Agora os dois planos usam Hash Join
  - Plano A: tabela hash com todos os movie_casts
  - Plano B: tabela hash com todos os movies
- Qual é melhor?



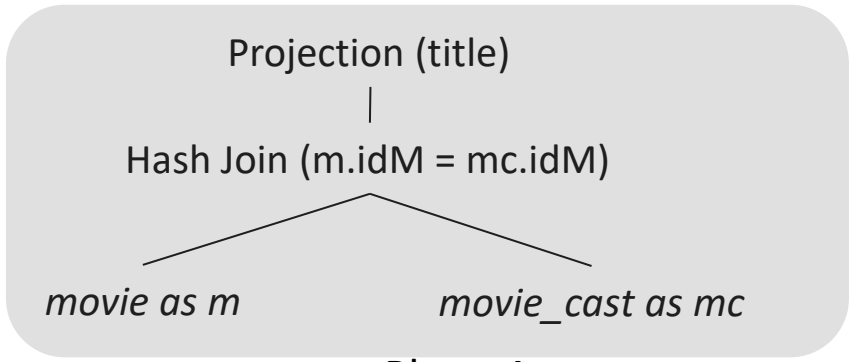
Plano A



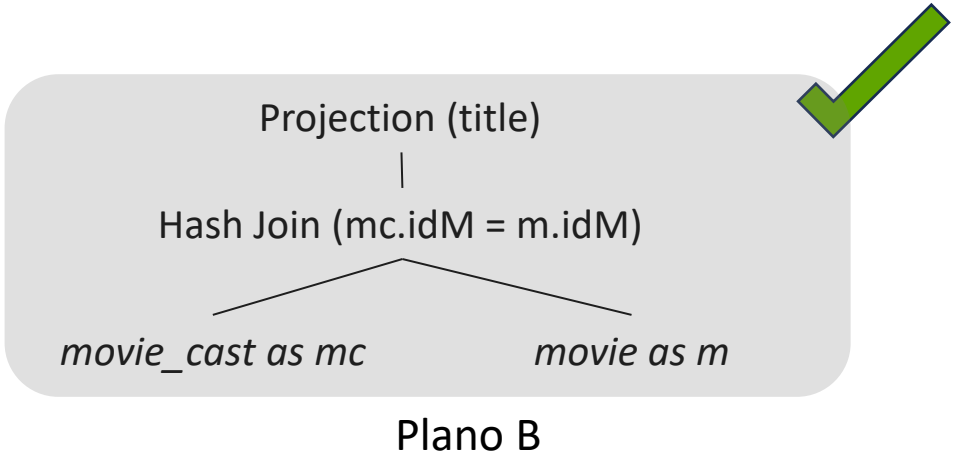
Plano B

Reduzir transferências de páginas/consumo de memória

- Agora os dois planos usam Hash Join
  - Plano A: tabela hash com todos os movie_casts
  - Plano B: tabela hash com todos os movies
- No quesito consumo de memória, o plano B é melhor
  - A tabela hash ocupa menos espaço



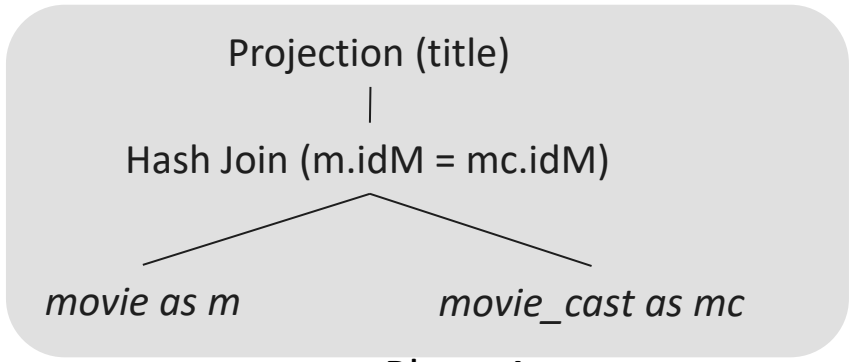
Plano A



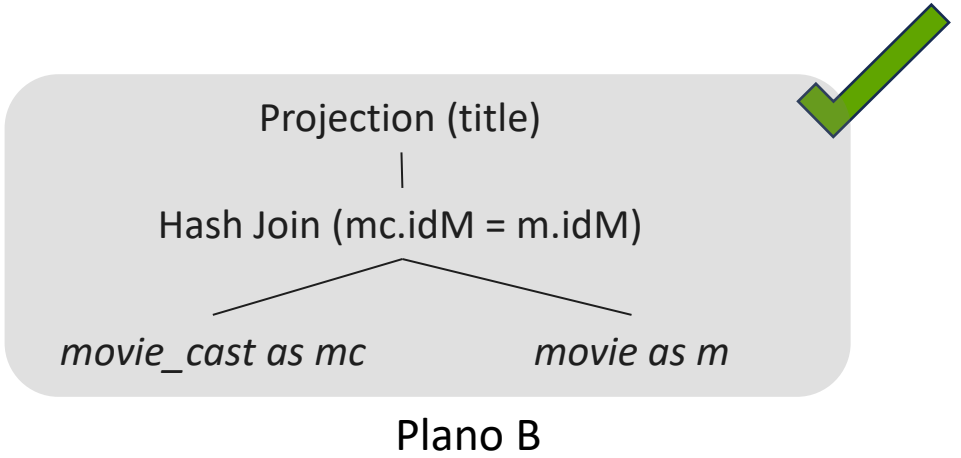
Plano B

Reduzir transferências de páginas/consumo de memória

- Outro fator: uniformidade da tabela hash
  - Plano A: menos uniforme, pois a chave idM se repete em movie_cast
  - Plano B: mais uniforme, pois a chave idM não se repete em movie



Plano A



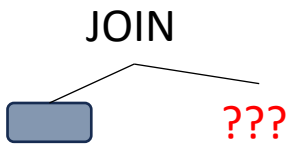
Plano B

Sumário

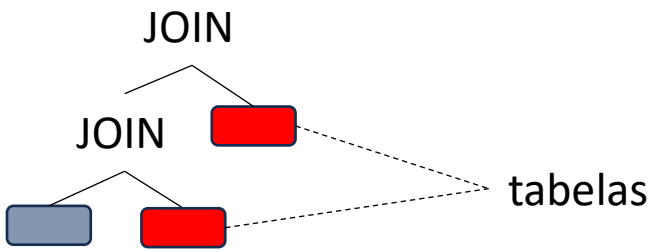
- Critérios usados
  - correspondência direta
  - reduzir transferências de páginas/consumo de memória
- Mais do que duas tabelas
  - Nested Loop
  - Hash Join
- Influência de filtros em junções
  - otimização push down
  - mudança do lado das tabelas
  - mudança do algoritmo de junção

Mais do que duas tabelas

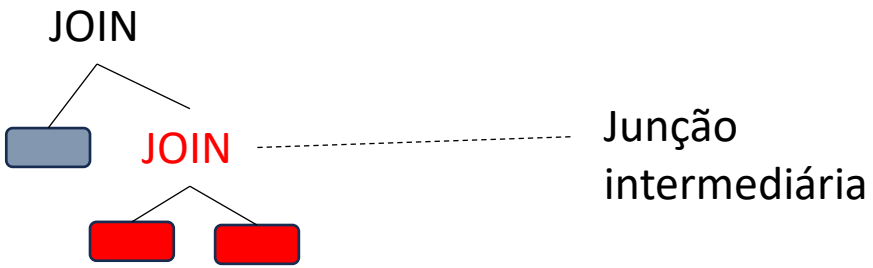
- E quando há mais do que duas tabelas, o que colocar no lado interno de uma junção?



- Duas possibilidades:



Usar uma tabela no lado interno



Usar uma junção intermediária no lado interno

Mais do que duas tabelas

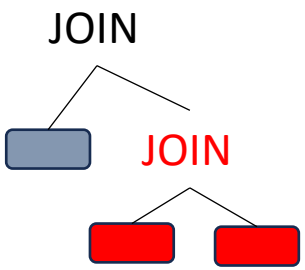
- Colocar tabelas no lado interno é uma estratégia interessante
- Com Nested Loop Join
  - Isso pode ajudar a reduzir o esforço da busca
- Com Hash Join
  - isso pode ajudar a reduzir consumo de memória

Sumário

- Critérios usados
  - correspondência direta
  - reduzir transferências de páginas/consumo de memória
- Mais do que duas tabelas
  - Nested Loop
  - Hash Join
- Influência de filtros em junções
  - otimização push down
  - mudança do lado das tabelas
  - mudança do algoritmo de junção

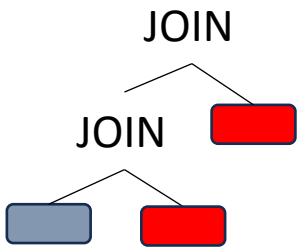
Nested Loop

- Ao manter uma junção intermediária no lado interno
  - Cada busca da junção de fora leva à execução da junção intermediária
  - Assim, a junção intermediária é executada múltiplas vezes
    - Uma vez para cada registro que vem da tabela externa



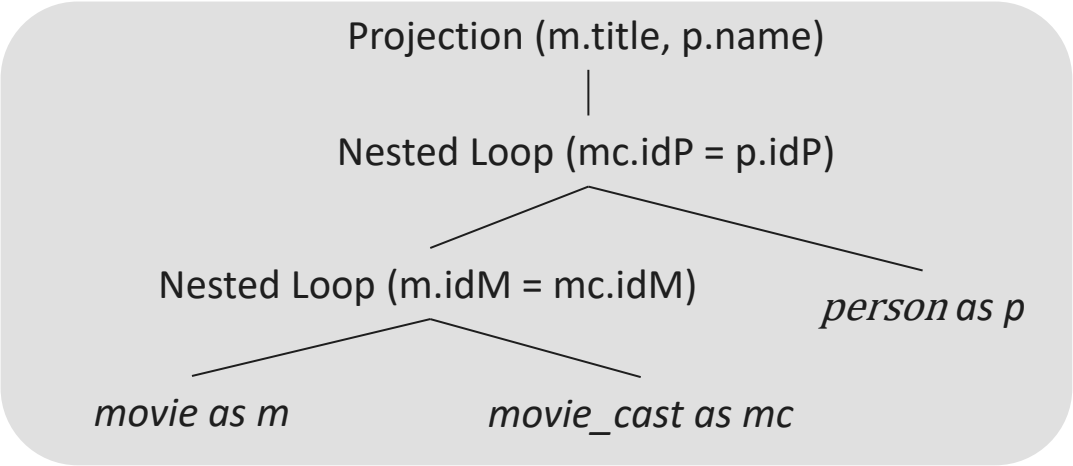
Nested Loop

- Ao manter uma tabela no lado interno
  - A busca não exige o reprocessamento de uma parte interna da árvore
  - Além disso, se a tabela for devidamente indexada, a busca pode ser feita por meio de seeks



Nested Loop

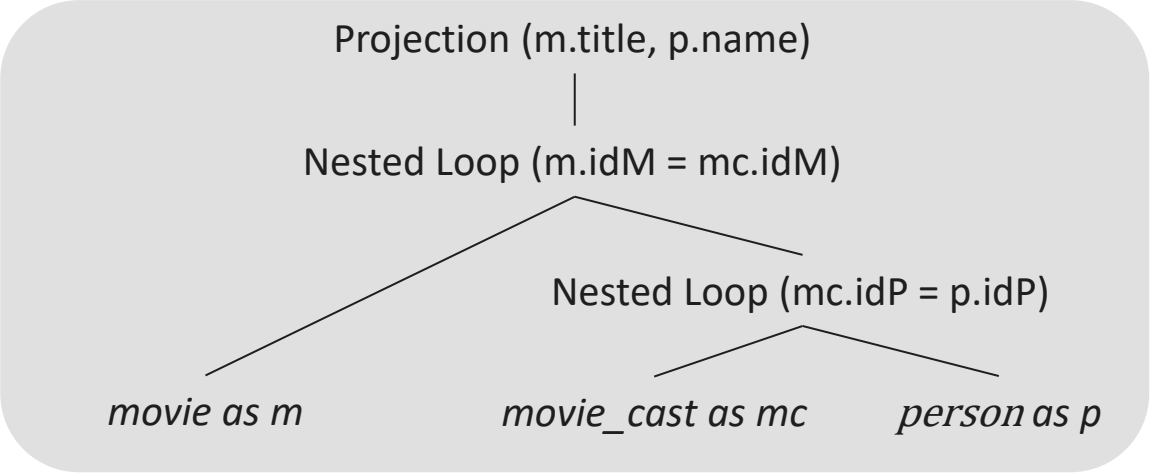
- No plano A
  - Os dois nested loop joins têm tabelas do lado interno
  - O filtro da junção é feito sobre arquivos indexados



Plano A

Nested Loop

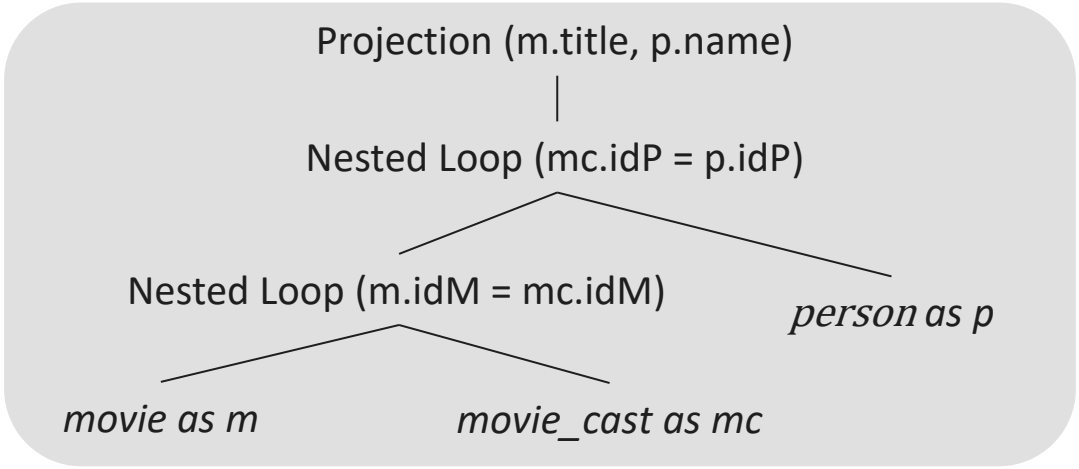
- No plano B
  - Uma das junções é realizada sobre o resultado de outra junção
    - A cada filme, toda a junção entre movie_cast e person é calculada
    - E o filtro da junção deve fazer uma varredura em cada tupla resultante da junção intermediária



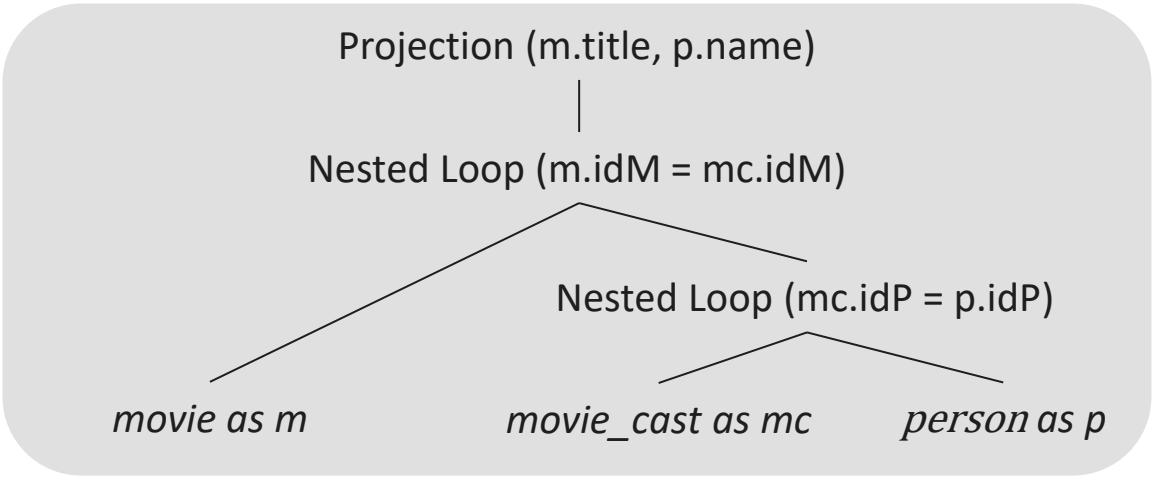
Plano B

Nested Loop

- Qual é melhor?



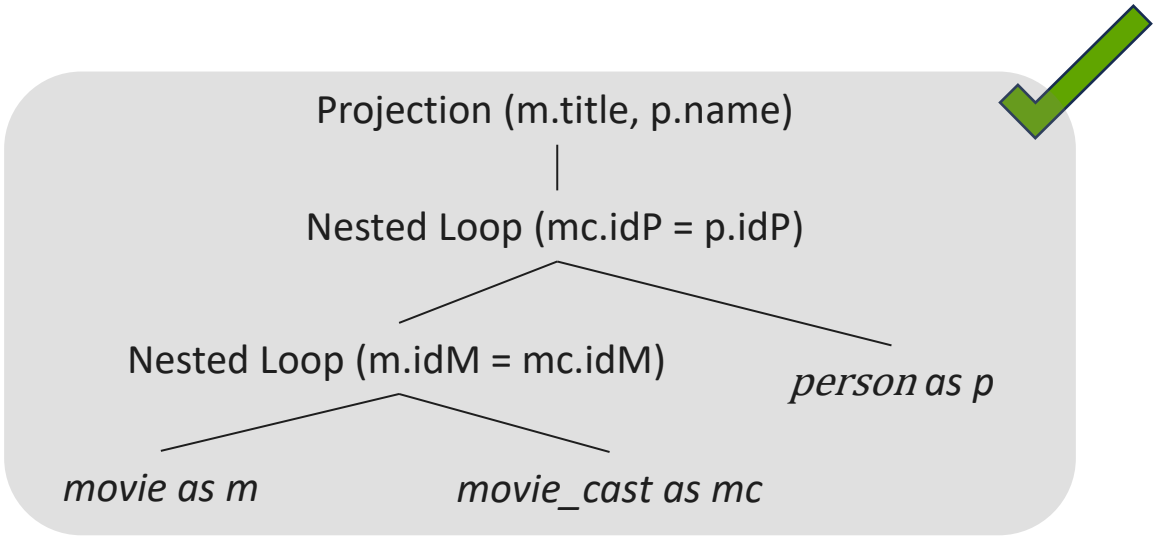
Plano A



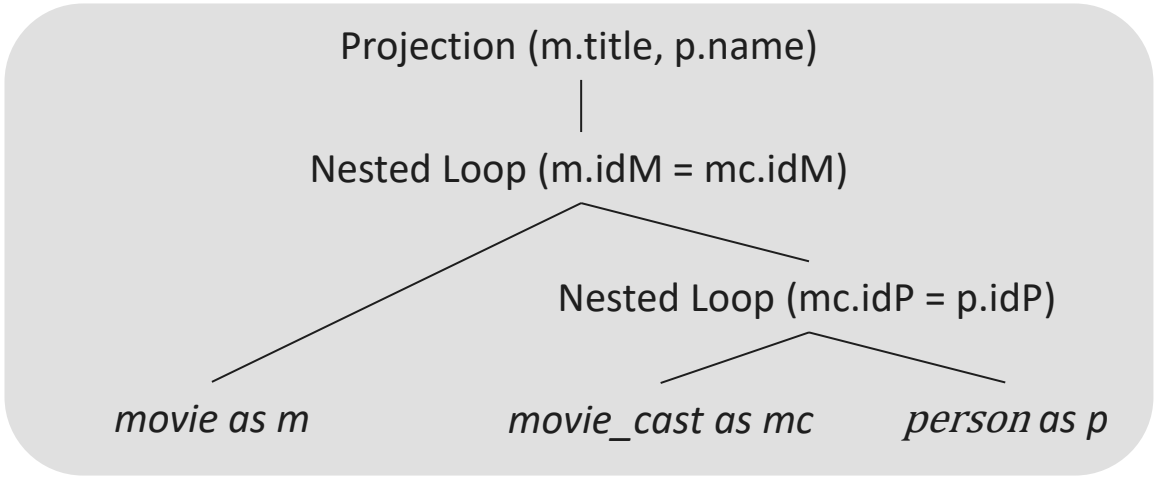
Plano B

Nested Loop

- O plano A é melhor
  - O filtro da junção não exige reprocessamento de uma parte interna da árvore
  - Os dois Nested Loops são indexados



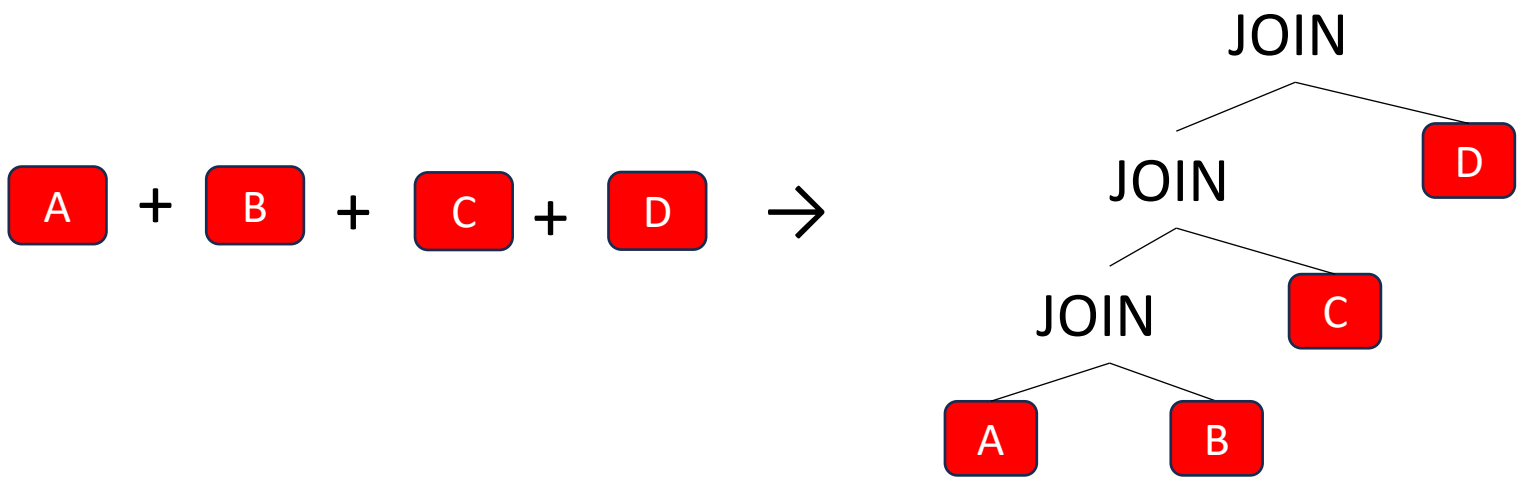
Plano A



Plano B

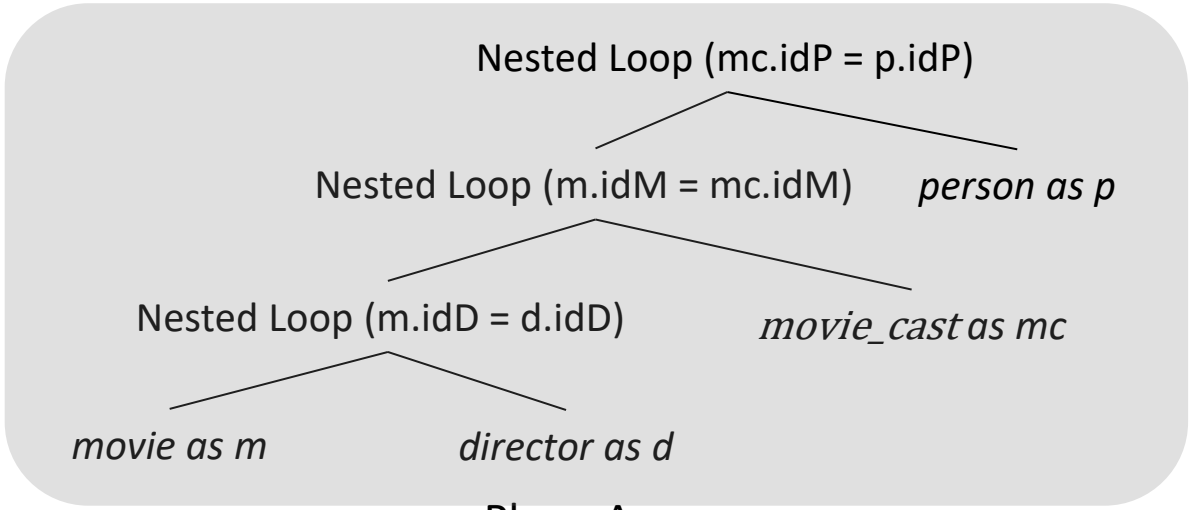
Nested Loop

- Quando o lado interno contém tabelas, o plano é inclinado para a esquerda
  - Nesses casos, a construção de um plano de execução é simplificada
    - O objetivo do otimizador é decidir qual é a próxima tabela a incluir no plano
  - O MySQL gera planos dessa maneira (na maioria das vezes)

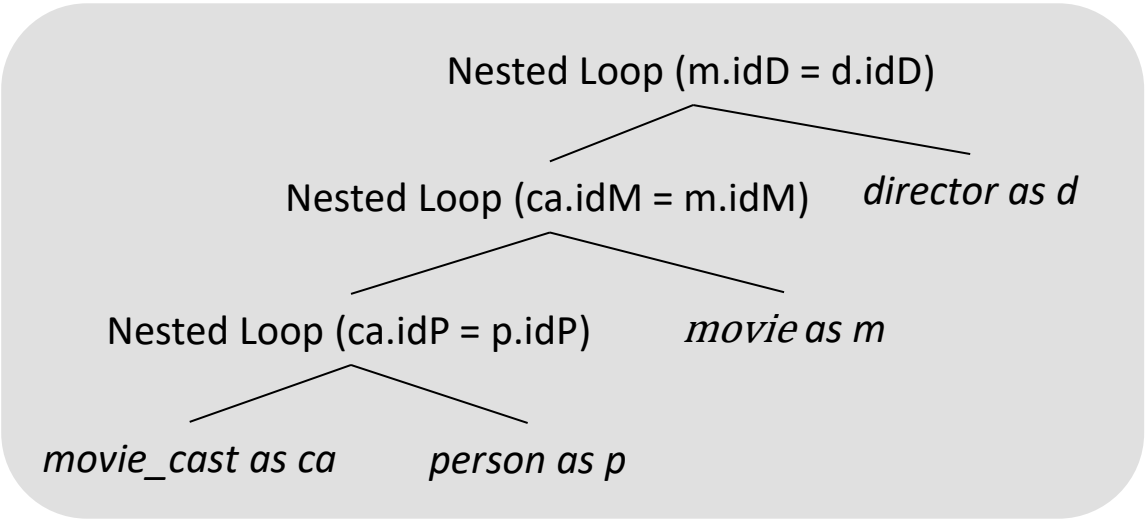


Nested Loop

- Exemplos de planos inclinados para a esquerda
- Cabe ao otimizador escolher qual é o melhor
- Regra geral: reduzir quantidade de registros do lado esquerdo



Plano A



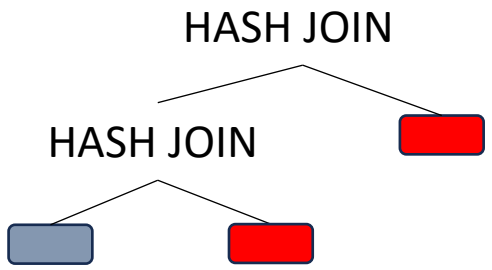
Plano B

Sumário

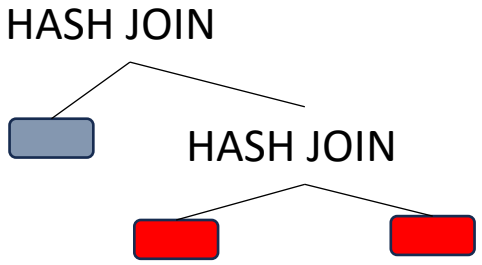
- Critérios usados
  - correspondência direta
  - reduzir transferências de páginas/consumo de memória
- Mais do que duas tabelas
  - Nested Loop
  - Hash Join
- Influência de filtros em junções
  - otimização push down
  - mudança do lado das tabelas
  - mudança do algoritmo de junção

Hash Join

- Com junções intermediárias no lado interno
  - O tamanho das tabelas hash é o tamanho das junções realizadas
- Ao manter uma tabela no lado interno
  - As tabelas hash ficam menores



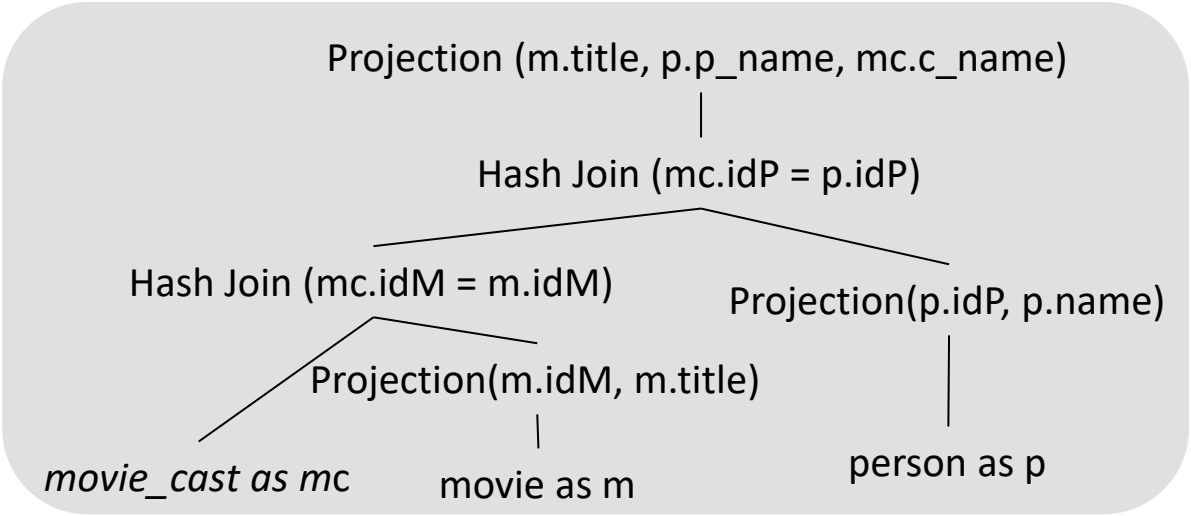
Uma tabela no lado interno



Uma junção intermediária no lado interno

Hash Join

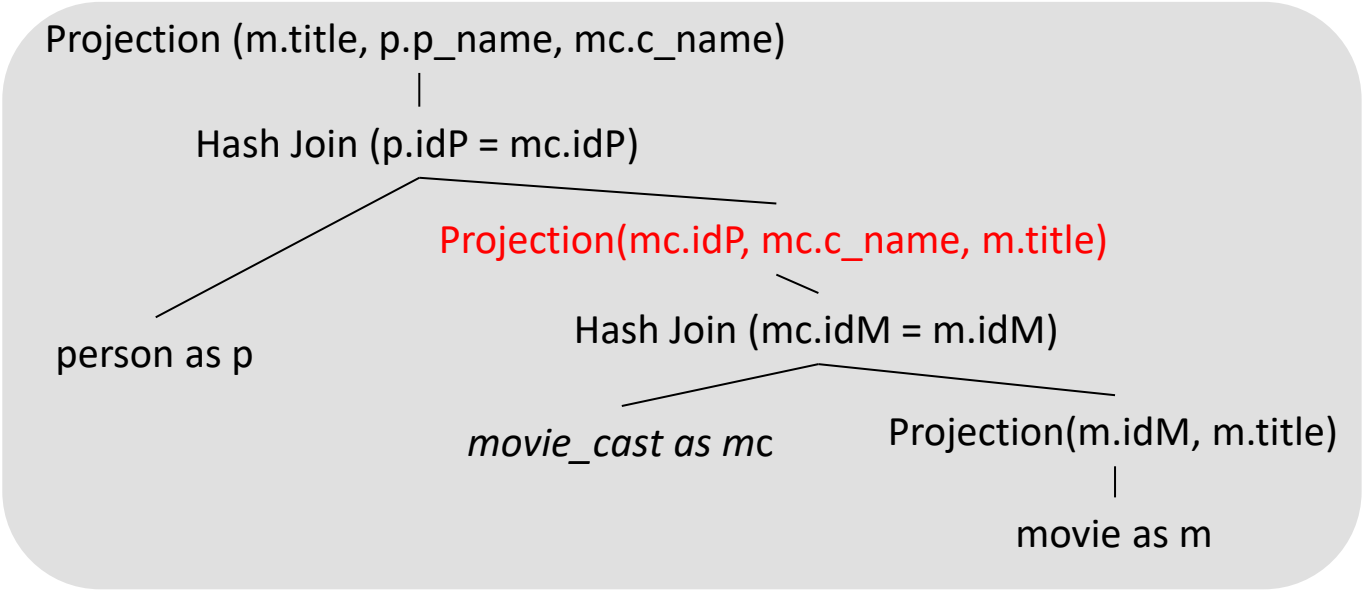
- No plano A, as tabelas hash
  - Foram criadas sobre tabelas (movie), (person)
  - Possuem duas colunas cada (m.idM, m.title), (p.idP, p.p_name)
  - Têm chaves que não se repetem (idM e idP são únicos)



Plano A

Hash Join

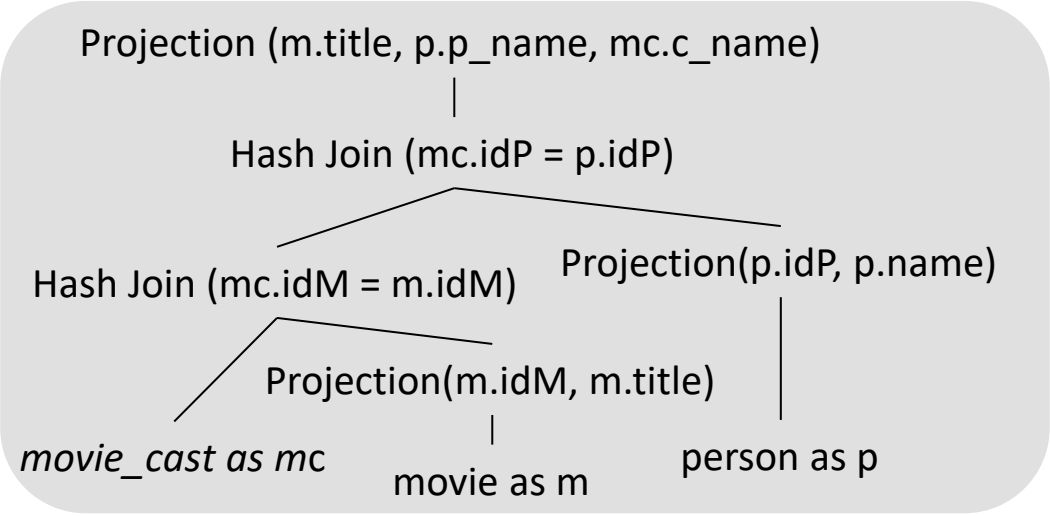
- No plano B, uma das tabelas hash foi criada sobre o resultado de uma junção (movie_cast, movie)
  - Possui três colunas (idP, c_name, title)
  - Tem chaves que se repetem (idP tem várias ocorrências iguais)



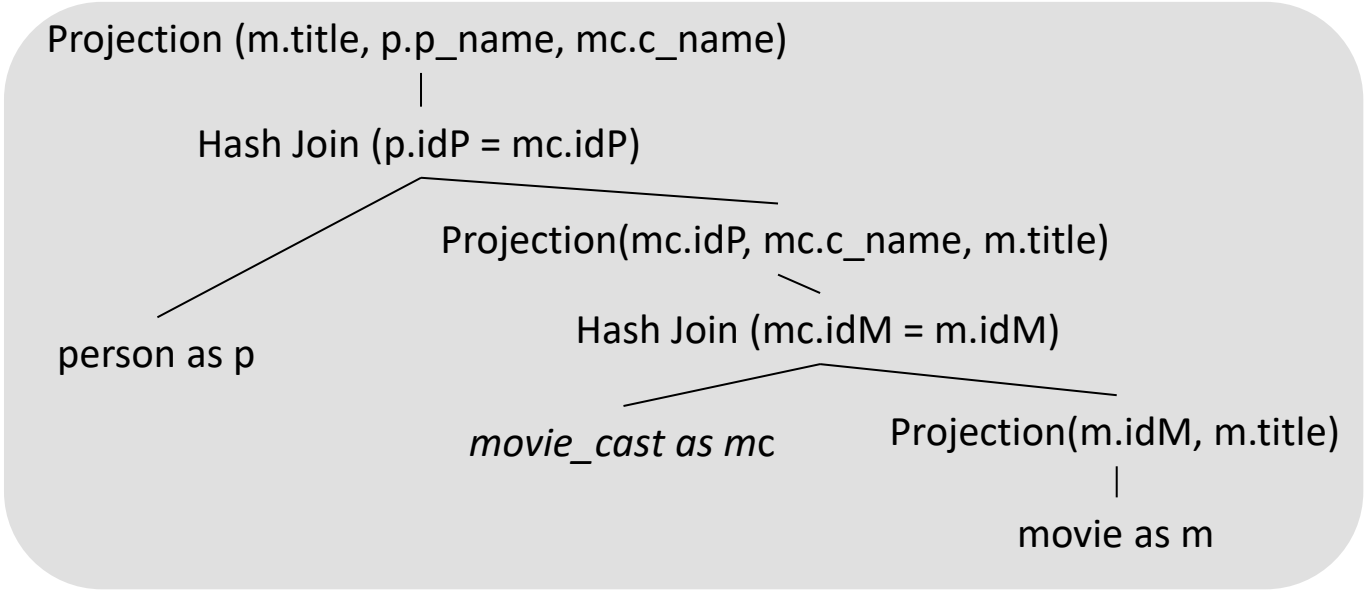
Plano B

Hash Join

- Qual é melhor?



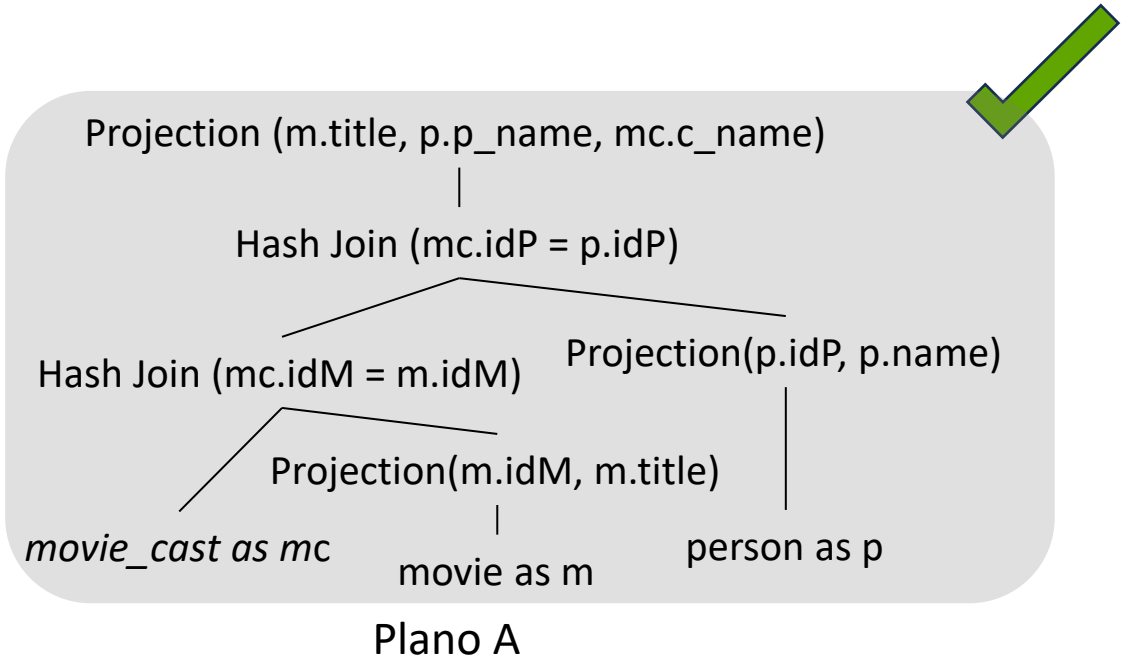
Plano A



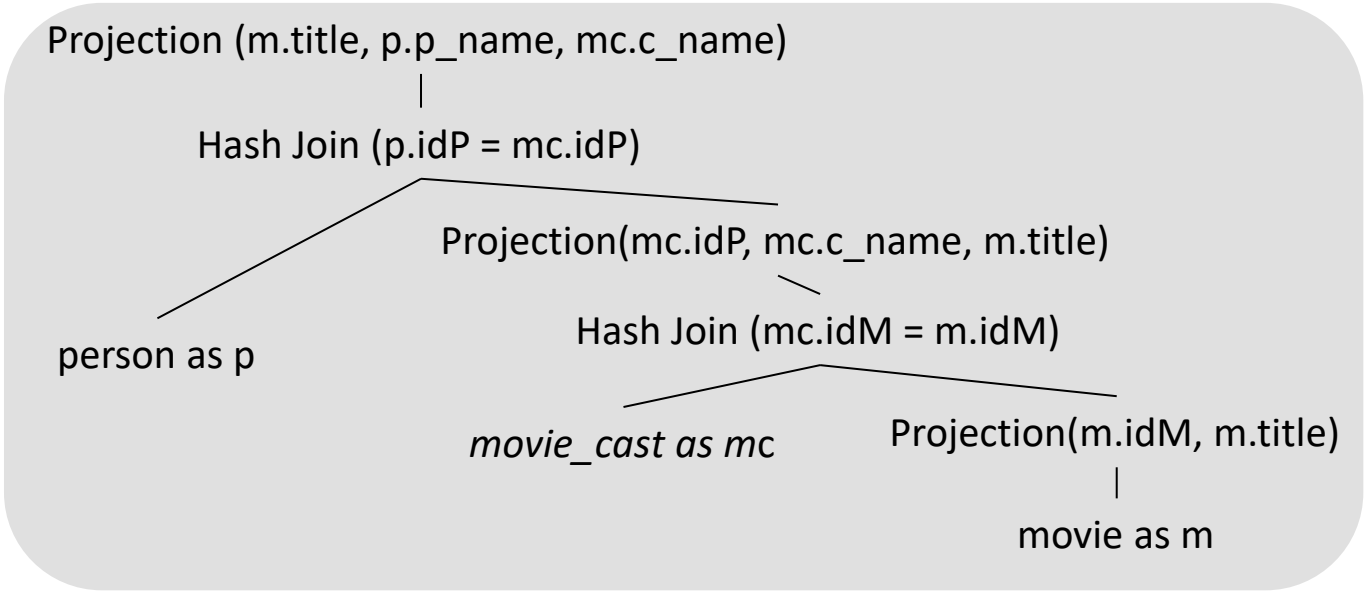
Plano B

Hash Join

- Plano A é melhor
  - Mantem tabelas no lado interno
    - reduz o consumo de memória, sem penalizar a busca



Plano A



Plano B

Sumário

- Critérios usados
  - correspondência direta
  - reduzir transferências de páginas/consumo de memória
- Mais do que duas tabelas
  - Nested Loop
  - Hash Join
- Influência de filtros em junções
  - otimização push down
  - mudança do lado das tabelas
  - mudança do algoritmo de junção

Influência de filtros em junções

- De modo geral
  - A definição de um plano de execução é um processo complexo, afetado por diversas variáveis
    - Unicidade da chave de junção
    - Tamanho das tabelas
    - Presença de filtros seletivos
    - Necessidade de complementação
    - ...
- A seletividade dos filtros tem um papel muito importante na tomada de decisão, referente à
  - Formato da árvore
  - Ordem das junções
  - Escolha do algoritmo de junção
- A seguir, veremos como a presença de filtros pode afetar a composição de um plano de execução, no tocante à junções

Sumário

- Critérios usados
  - correspondência direta
  - reduzir transferências de páginas/consumo de memória
- Mais do que duas tabelas
  - Nested Loop
  - Hash Join
- Influência de filtros em junções
  - otimização push down
  - mudança do lado das tabelas
  - mudança do algoritmo de junção

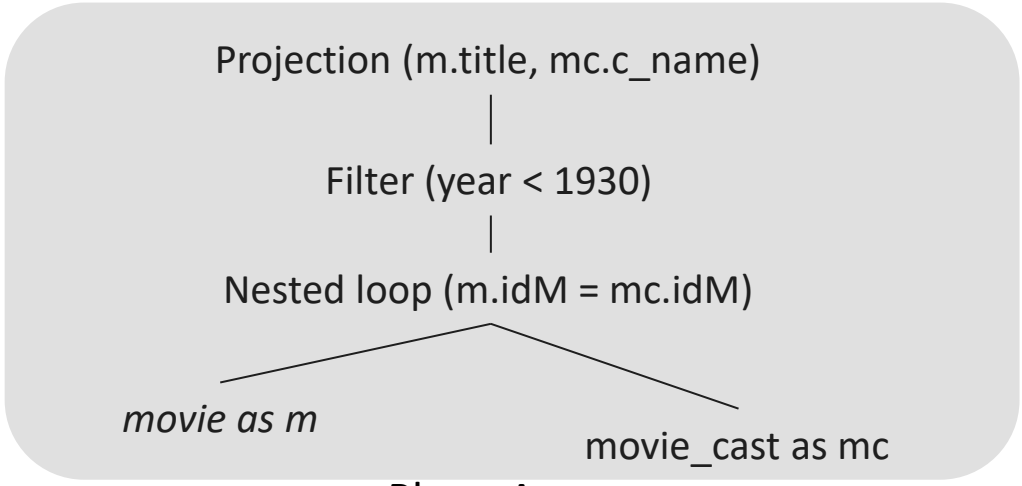
Otimização push-down

- De modo geral
  - Filtros são empurrados para baixo, ficando o mais próximo possível das tabelas a qual são aplicáveis
  - Otimização chamada de push-down

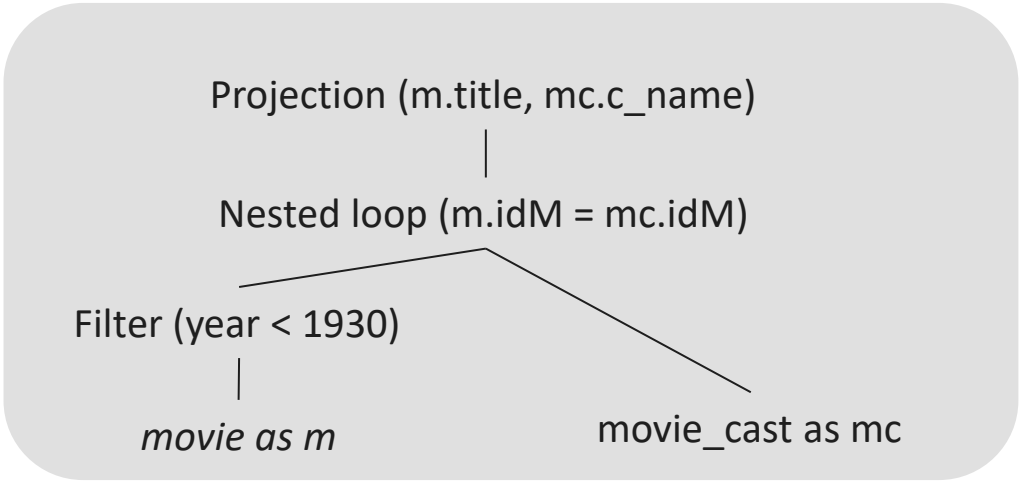
- Isso reduz o esforço e a quantidade de memória necessária durante a execução de uma consulta

Otimização push-down

- Plano A: filtro feito depois da junção
- Plano B: filtro feito antes da junção
- Qual é melhor?



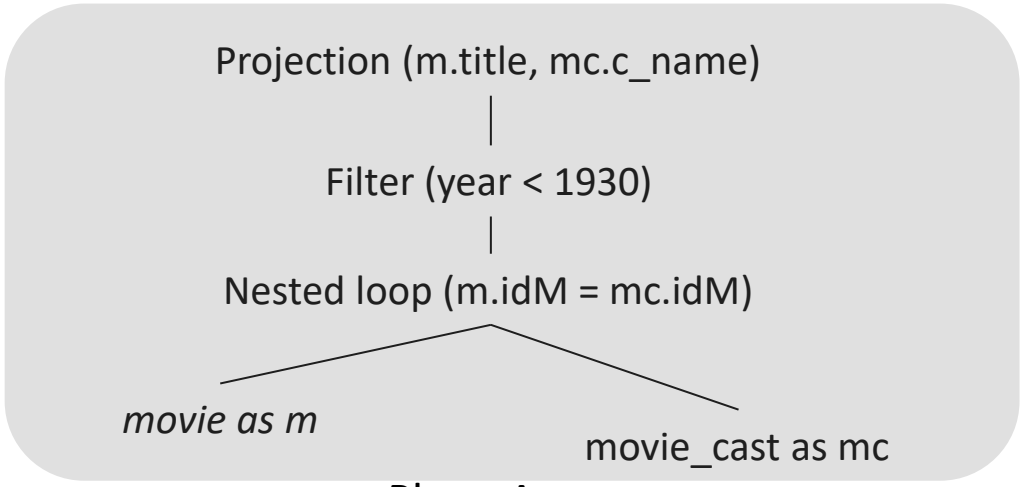
Plano A



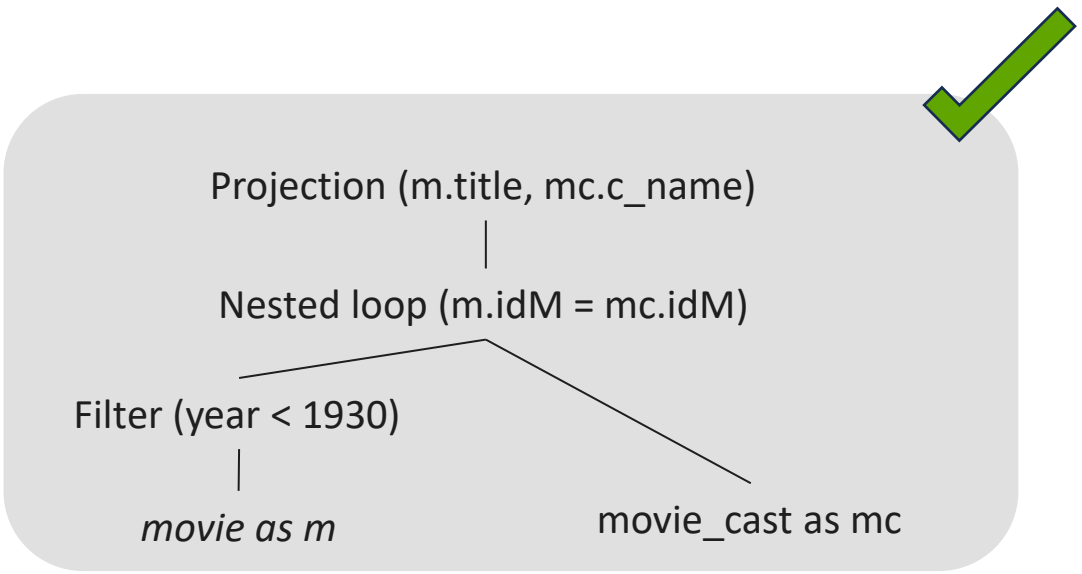
Plano B

Otimização push-down

- Plano B é melhor
  - Plano A: faz a junção para todos os filmes
  - Plano B: faz a junção apenas para os filmes necessários



Plano A



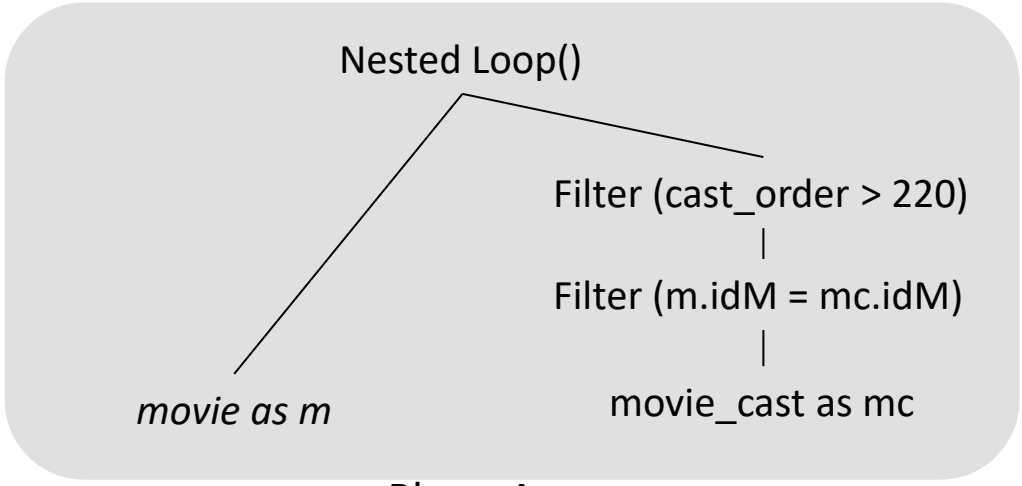
Plano B

Sumário

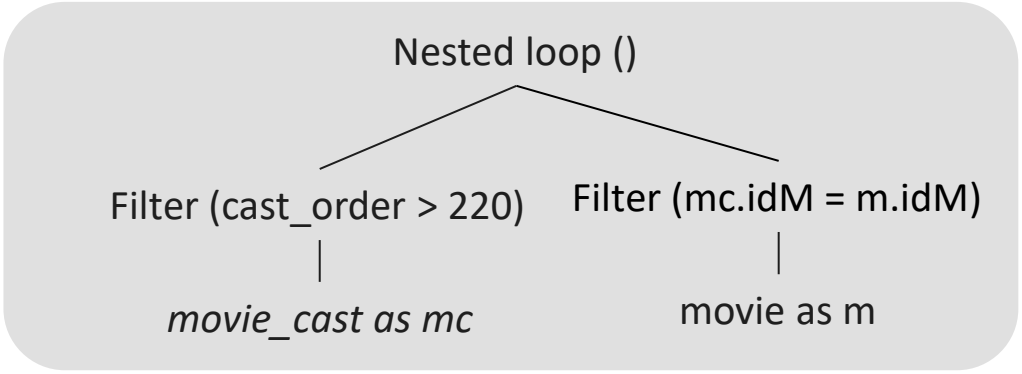
- Critérios usados
  - correspondência direta
  - reduzir transferências de páginas/consumo de memória
- Mais do que duas tabelas
  - Nested Loop
  - Hash Join
- Influência de filtros em junções
  - otimização push down
  - mudança do lado das tabelas
  - mudança do algoritmo de junção

Mudança do lado de tabelas

- Exemplo 1:
  - Plano A: filtro (cast_order > 220) do lado interno da junção
  - Plano B: filtro (cast_order > 220) do lado externo da junção



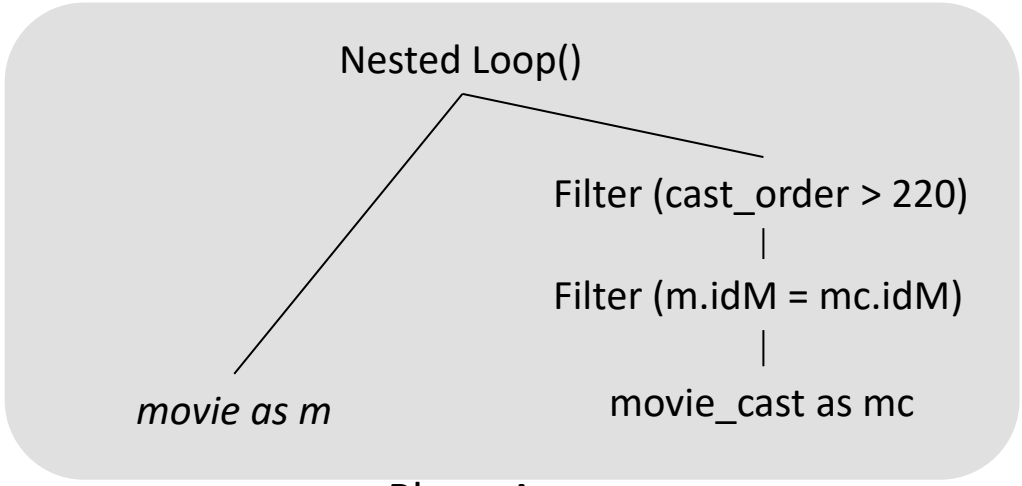
Plano A



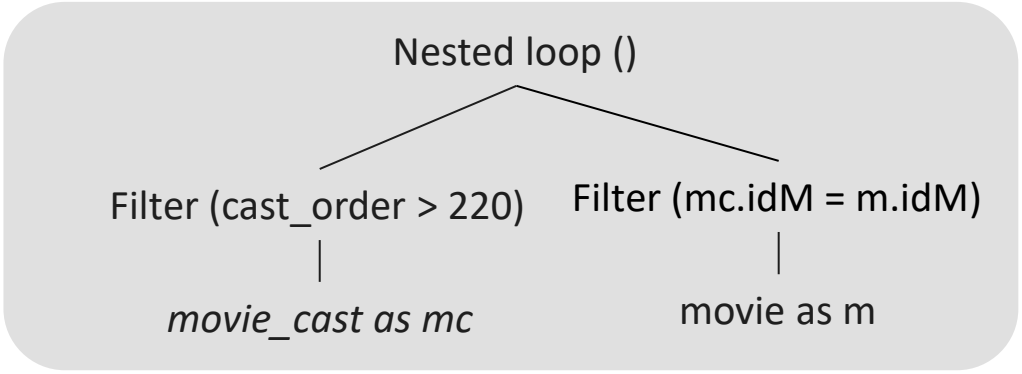
Plano B

Mudança do lado de tabelas

- Exemplo 1:
  - Qual é melhor?



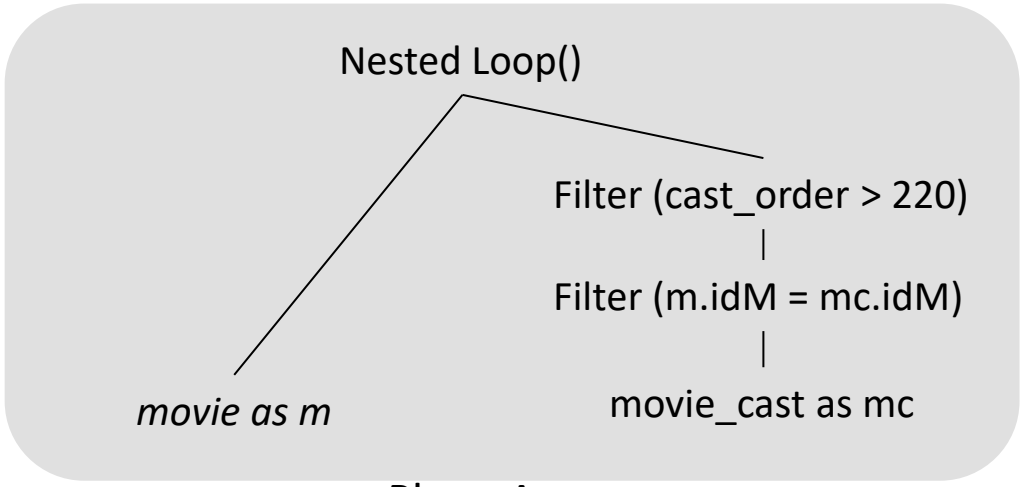
Plano A



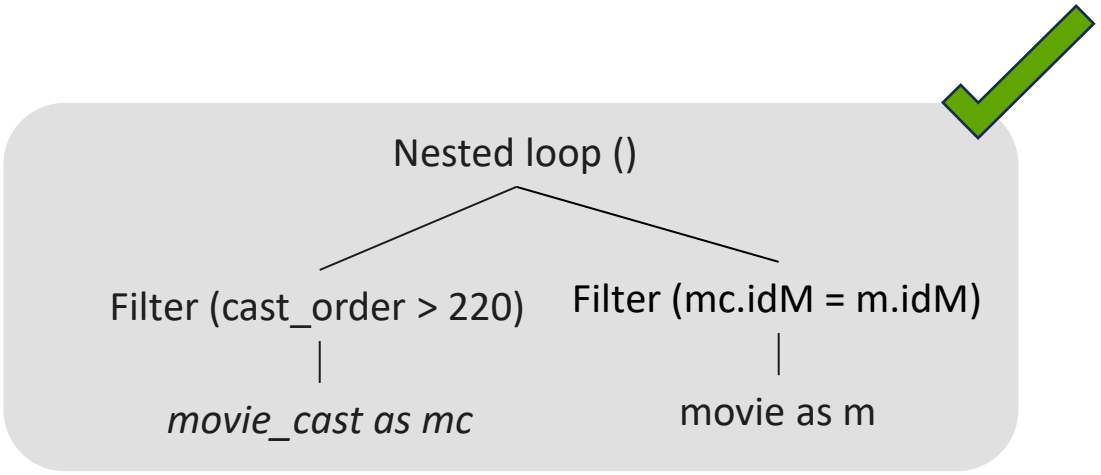
Plano B

Mudança do lado de tabelas

- Exemplo 1:
  - Plano A: O nested loop executa muitos seeks (um para cada movie)
  - Plano B: O nested loop Join executa poucos seeks
    - Apenas para os membros de elenco que chegaram até a junção
  - Plano B é melhor



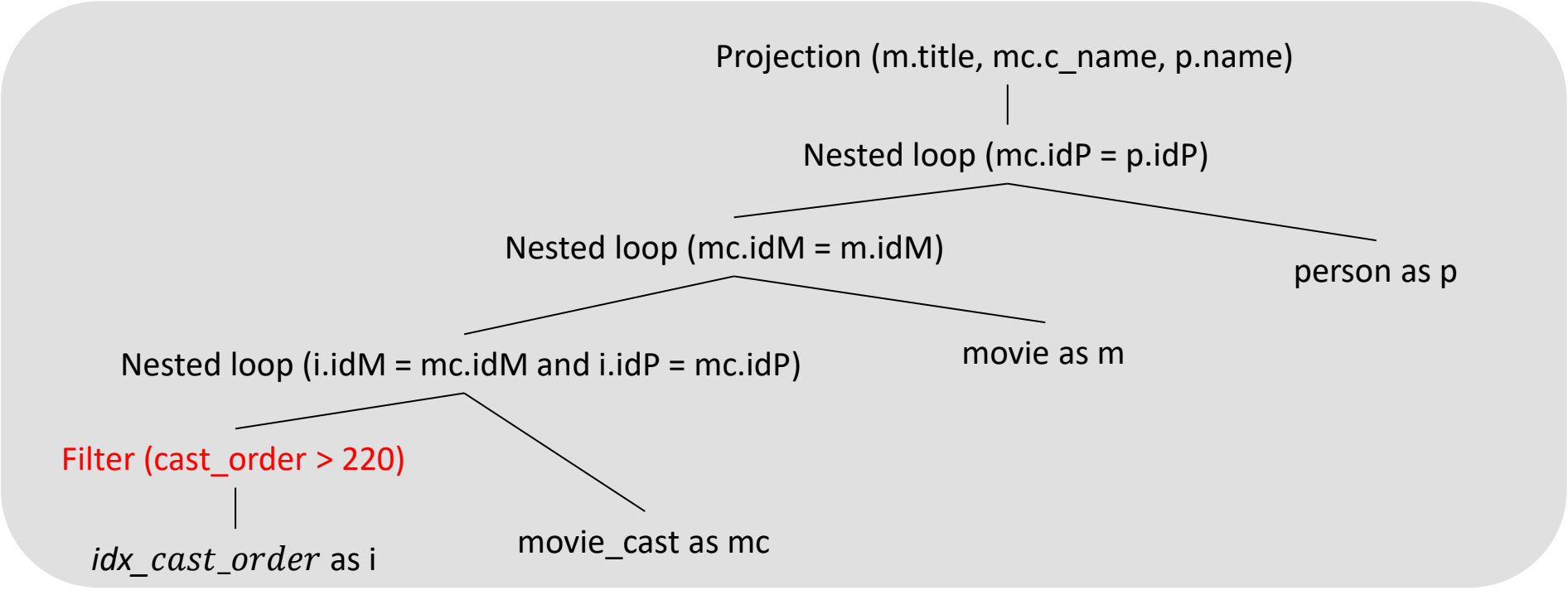
Plano A



Plano B

Mudança do lado de tabelas

- Exemplo 2: Se existir um índice (e o filtro for seletivo)
  - O índice pode determinar o ponto de partida do plano



Sumário

- Critérios usados
  - correspondência direta
  - reduzir transferências de páginas/consumo de memória
- Mais do que duas tabelas
  - Nested Loop
  - Hash Join
- Influência de filtros em junções
  - otimização push down
  - mudança do lado das tabelas
  - mudança do algoritmo de junção

Mudança do algoritmo de junção

- Exemplo: título de filmes e nomes de personagens cuja ordem de aparição for superior a 220

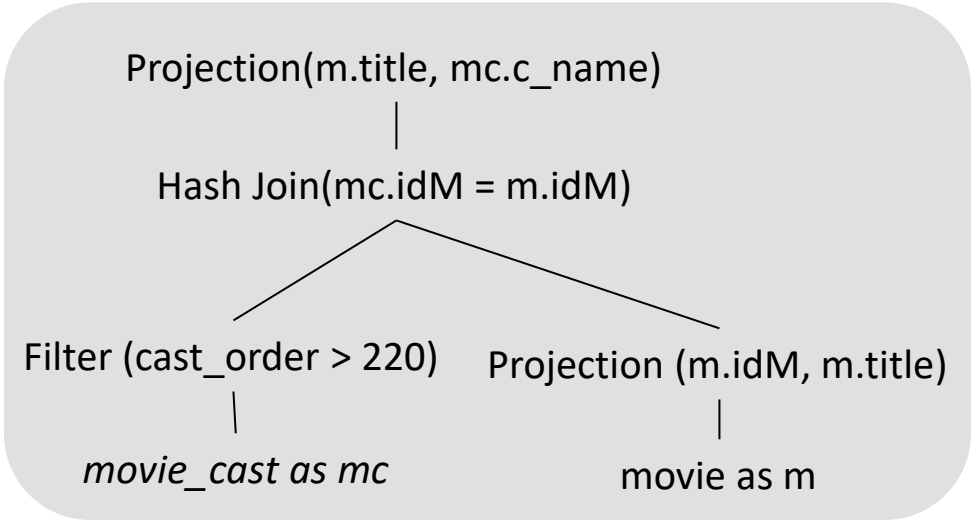
```
SELECT m.title, mc.c_name
FROM movie m JOIN movie_cast mc ON m.idM = mc.idM
Where mc.cast_order > 220
```

- A seletividade do filtro é alta

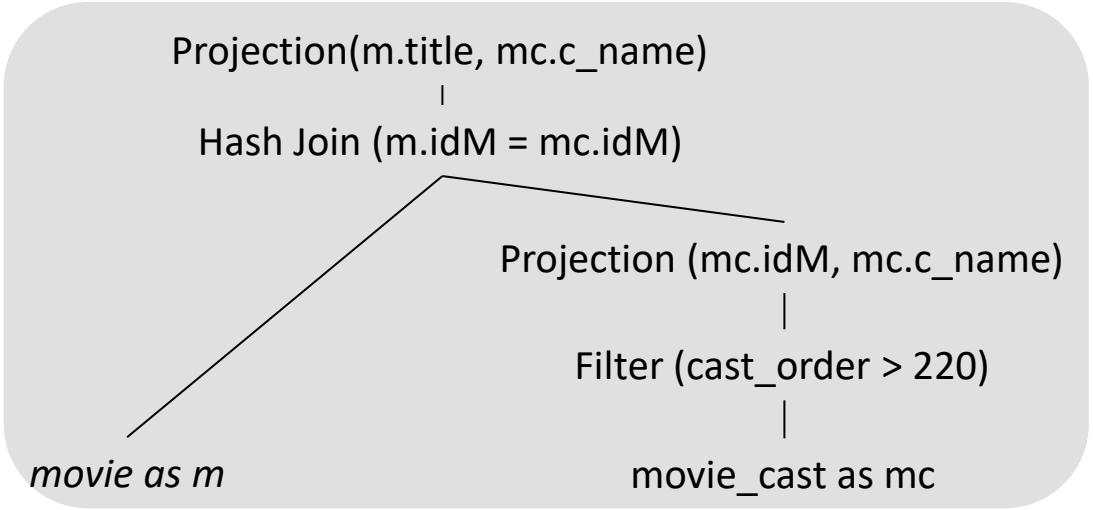
- Nesses casos, o Hash Join se torna uma opção menos atraente

Mudança do algoritmo de junção

- Plano A: filtro do lado externo
  - Vantagem: poucos seeks
  - Desvantagem: tabela hash muito grande
    - Em relação à quantidade de filmes que realmente interessam



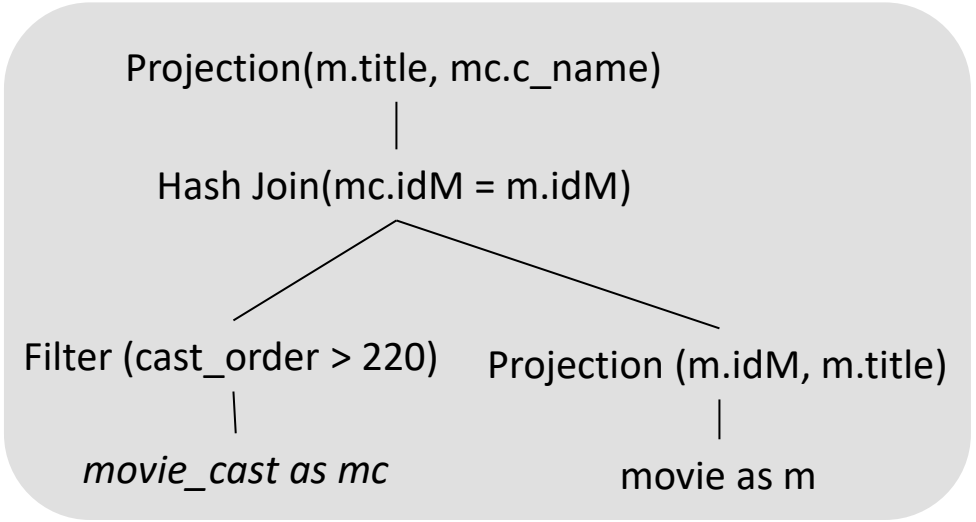
Plano A



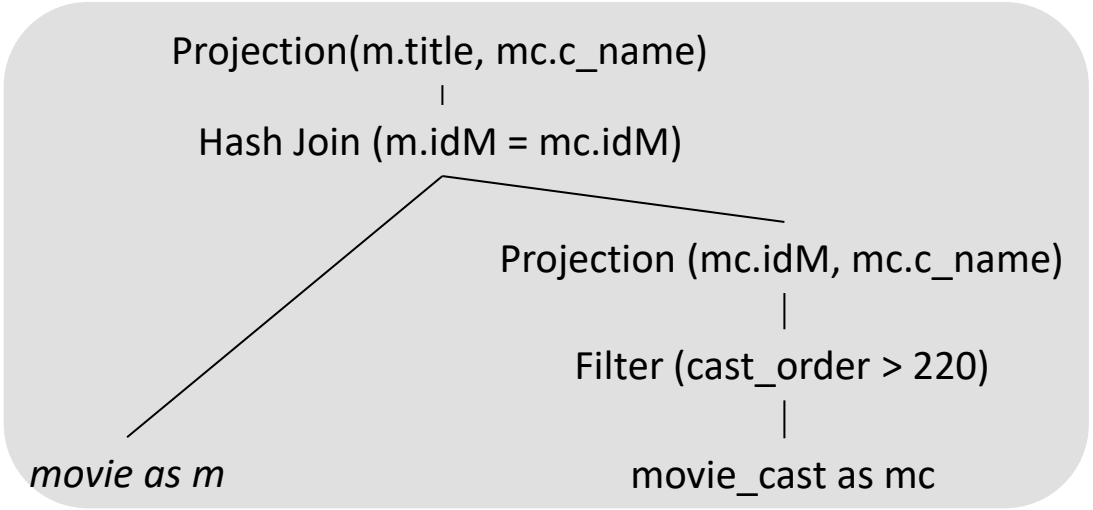
Plano B

Mudança do algoritmo de junção

- Plano B: filtro do lado interno
  - Vantagem: tabela hash pequena
  - Desvantagem: muitos seeks



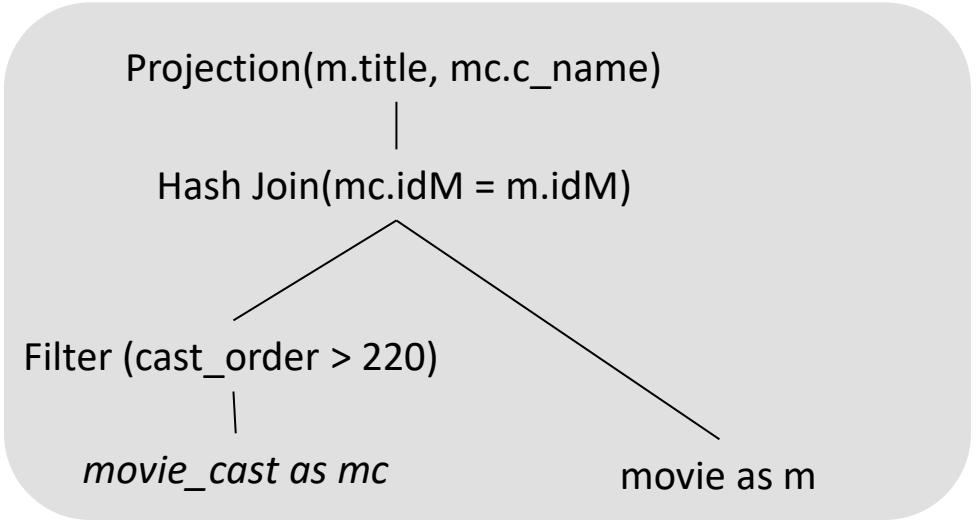
Plano A



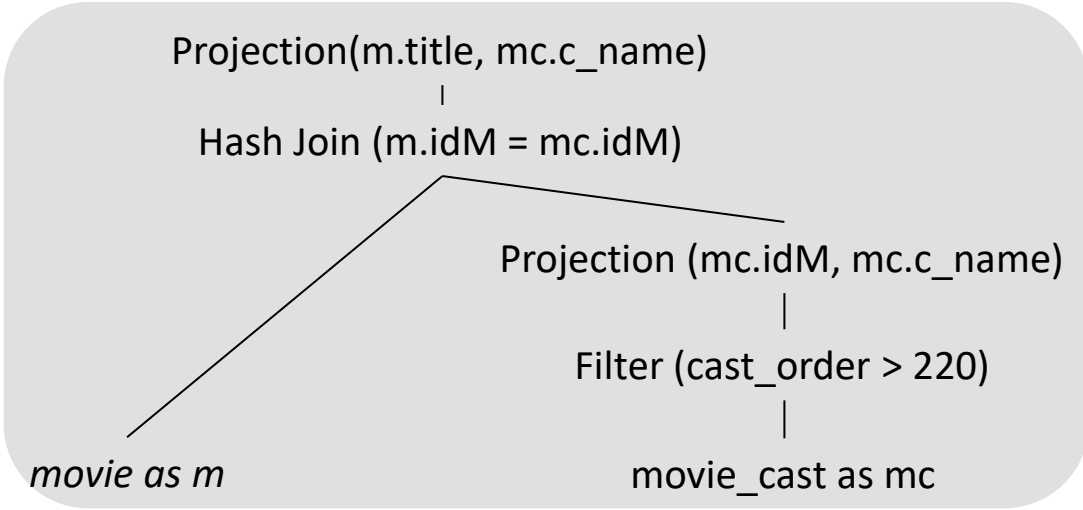
Plano B

Mudança do algoritmo de junção

- Tanto A quanto B possuem deficiências
- É possível que o otimizador use Nested Loop em vez de Hash Join
- Outra opção é acrescentar o operador Memoize



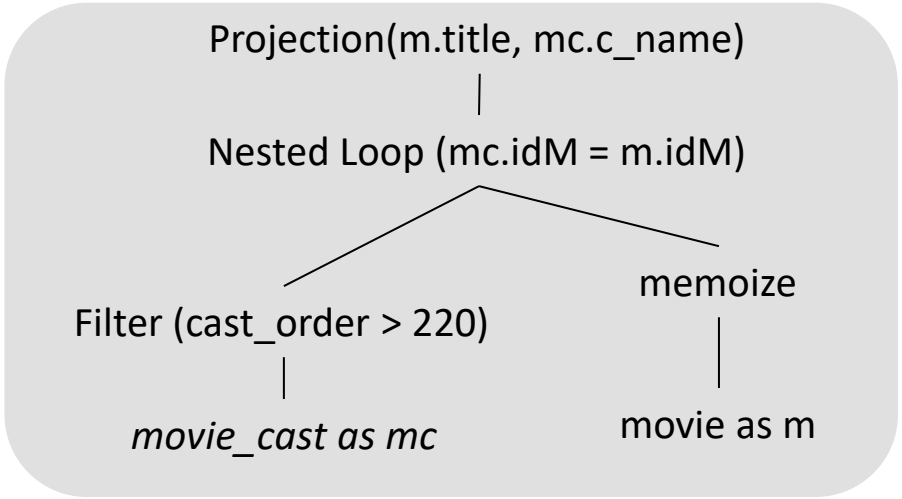
Plano A



Plano B

Mudança do algoritmo de junção

- Alternativa: Nested Loop com Memoize
  - Na primeira vez que for feito o seek sobre um idM
    - O registro correspondente é armazenado na memória (Memoize)
  - Se o mesmo idM aparecer novamente
  - Será possível obtê-lo a partir da memória



Plano C

Encerrando

- Fatores relevantes
  - Nested Loop: transferência de páginas
    - Prioriza manter tabela menor do lado externo
  - Hash Join: consumo de memória
    - Prioriza manter tabela menor do lado interno

- Índices sobre as colunas filtradas podem ajudar na definição do melhor plano de execução

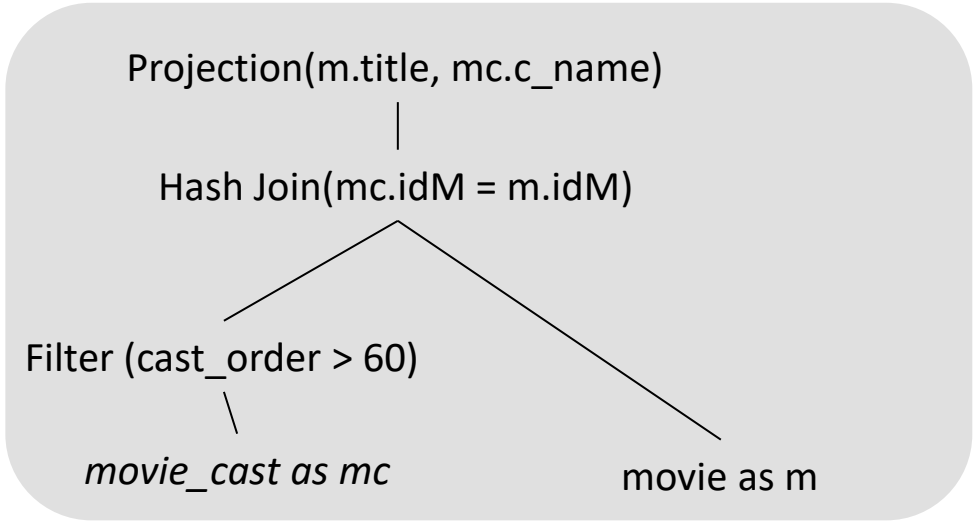
Atividade Individual

- Crie os três planos de execução discutidos na última parte da aula
  - Planos apresentados nos próximos slides

- Analise-os em função da quantidade de páginas lidas e consumo de memória

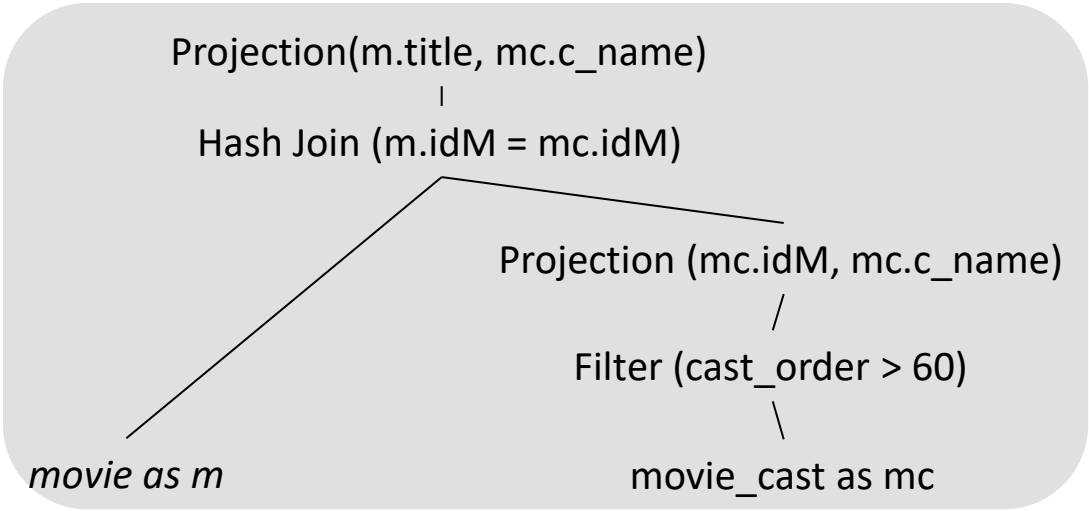
- Indique qual opção você escolheria, e justifique o motivo

Atividade Individual



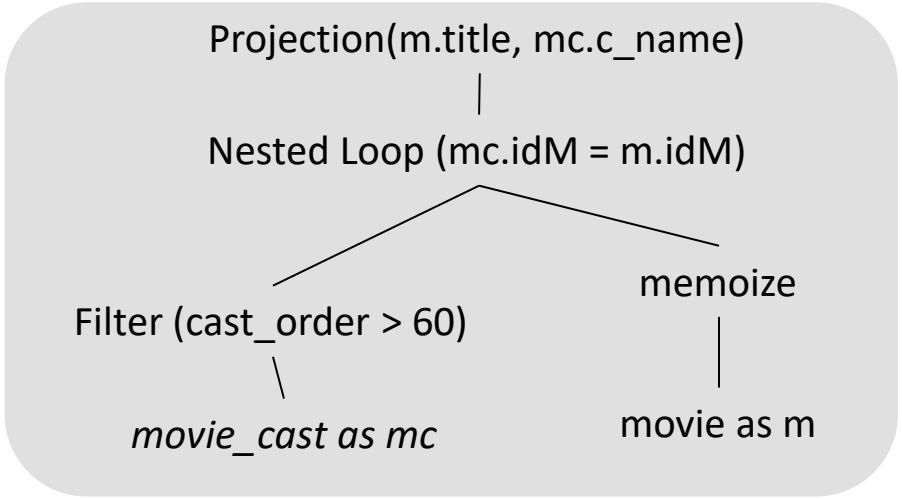
Plano A

Atividade Individual



Plano B

Atividade Individual



Plano C