Junções

Junções

- Cruzamento entre duas tabelas que recupera associações entre os registros com base em um predicado de junção

- Em bancos relacionais, onde as tabelas são conectadas por relacionamentos de chave primária/chave estrangeira, as junções são operações fundamentais

Junções

- Dois tipos
  - Equi-Join (caso mais comum)
    - O predicado de junção contém termos na forma (col1 = col2)
    - As colunas normalmente indicam conexões entre chave primária e chave estrangeira

```
SELECT p_name
FROM person p JOIN movie_cast mc ON p.person_id = mc.person_id
```

  - Non-equi-Join
    - O operador de comparação dos termos pode ser diferente da igualdade

```
SELECT p_name
FROM movie m JOIN person ON m.budget < p.salary
```

Junções

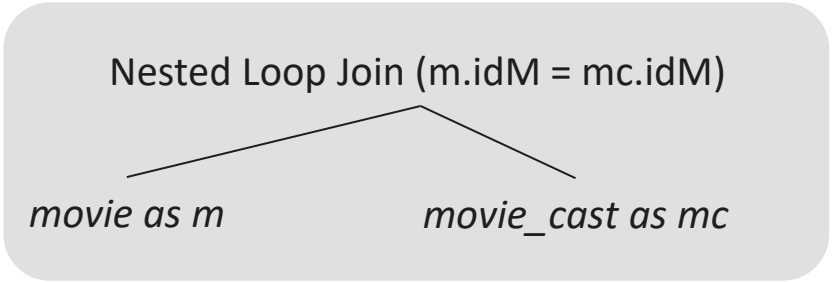
- Diversos algoritmos para realizar equi-joins
  - Nested Loop Join
  - Indexed Nested Loop Join
  - Hash Join
  - Merge Join
  - ...
- Além das versões clássicas, cada algoritmo possui variações
  - Grace Hash Join
  - Hash Right Anti Join
  - Merge Left Semi Join
  - ...

Sumário

- Nested Loop Join
  - Colunas de Correlação
  - Memoize
- Hash Join
  - Indicações de uso
    - Ausência de índices
    - Diminuir buscas aleatórias
  - Grace Join

Nested Loop Join

- Nested Loop Join é uma operação binária
- Lado da esquerda: lado externo
- Lado da direita: lado interno
- Funcionamento
  - Para cada registro do lado externo
    - Buscar as correspondências no lado interno
    - O formato da busca depende das características do lado interno



Nested Loop Join

- EXEMPLO 1: títulos de filmes que tiveram artistas registrados

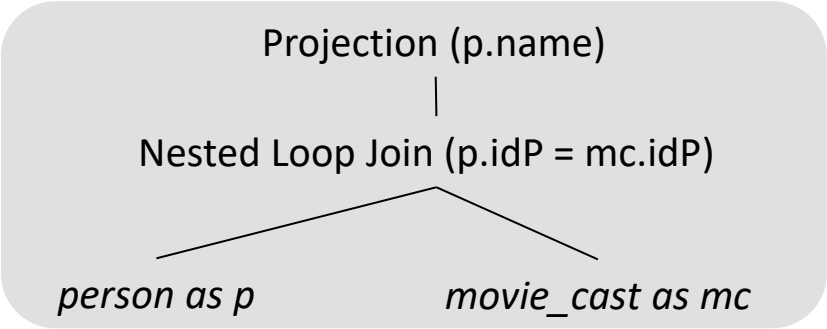
```
SELECT person_name
FROM person p
JOIN movie_cast mc ON p.idP = mc.idP
```

- A coluna idP
  - é um índice primário da tabela person (idP)
  - Faz parte do índice primário da tabela movie_cast (idM, idP)

- Veremos como esse índice pode ajudar a realizar a junção

Nested Loop Join

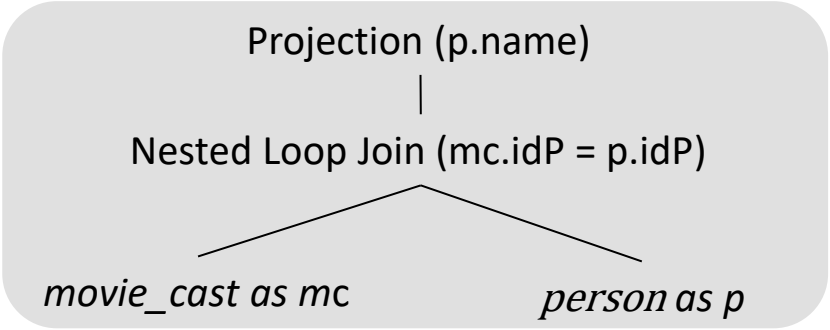
- Plano A
  - O lado interno é o índice primário de movie_cast, cuja chave é (idM, idP)
  - Como idP não faz parte do prefixo da chave de busca
    - Deve ser realizado um scan para localizar a correspondência
  - Ou seja, para cada person, um scan é realizado em movie_cast



Plano A

Nested Loop Join

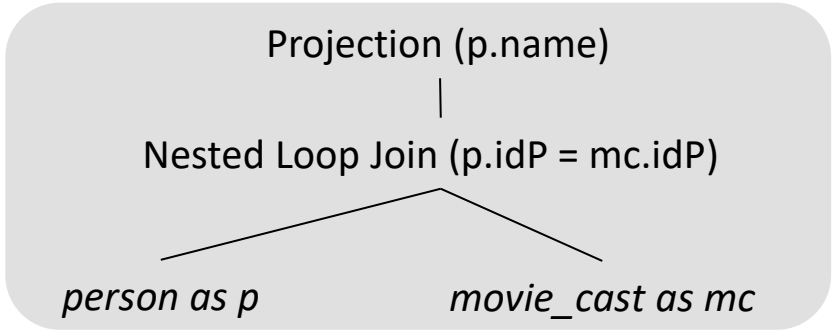
- Plano B
  - O lado interno é o índice primário de person , cuja chave é (idP)
  - Como idP é a chave de busca
    - Será realizado um seek para localizar a correspondência
  - Ou seja, para cada movie_cast, um seek é realizado em person



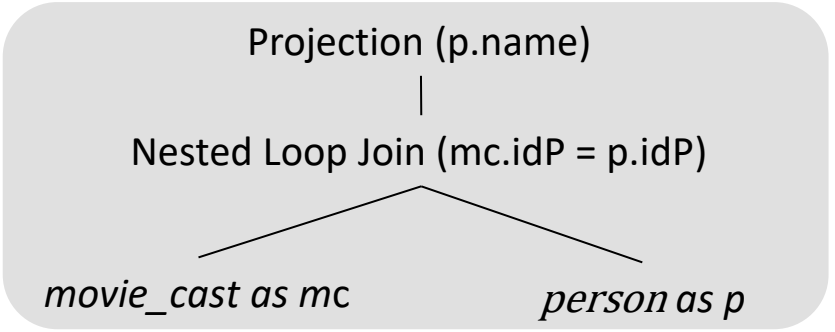
Plano B

Nested Loop Join

- Qual é melhor?



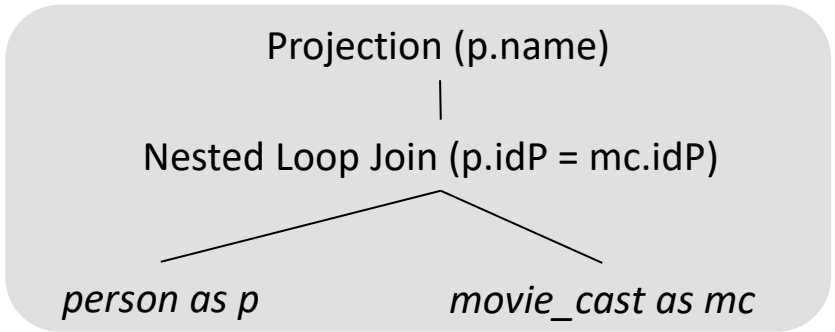
Plano A



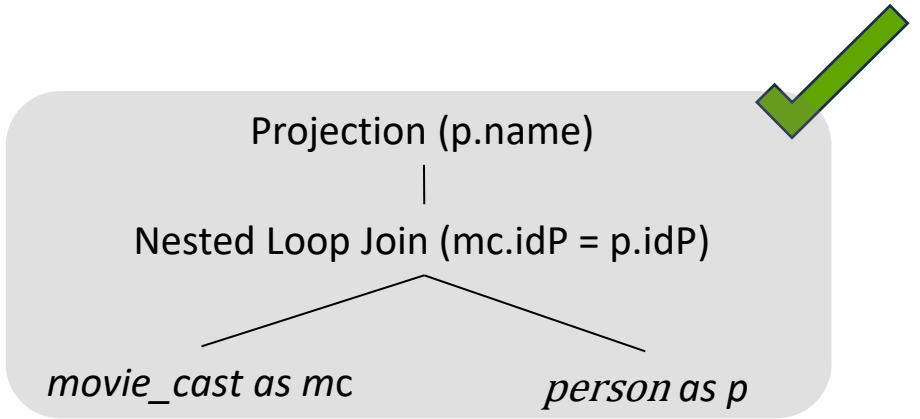
Plano B

Nested Loop Join

- O plano B é melhor
  - Utiliza um índice para recuperar correspondências por meio de um seek, em vez de percorrer todos os registros com um scan



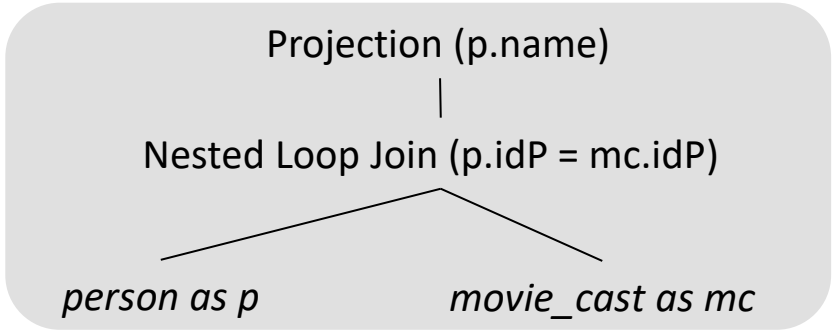
Plano A



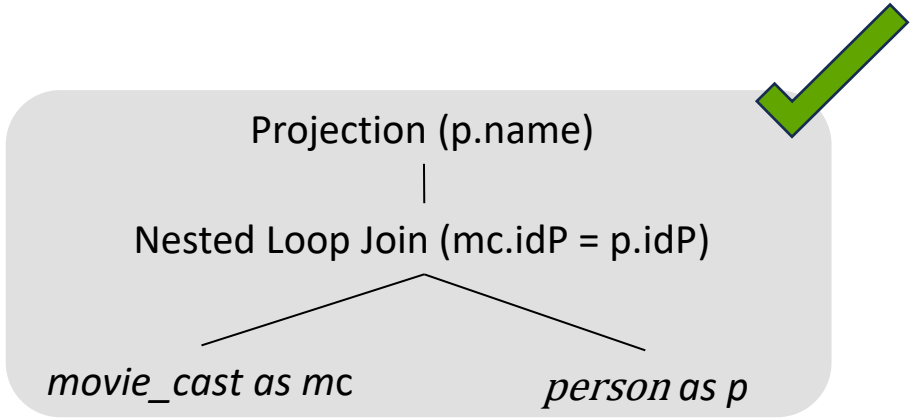
Plano B

Nested Loop Join

- Quando o Nested Loop Join consegue utilizar algum índice para fazer um seek, pode-se chamá-lo de Indexed Nested Loop Join



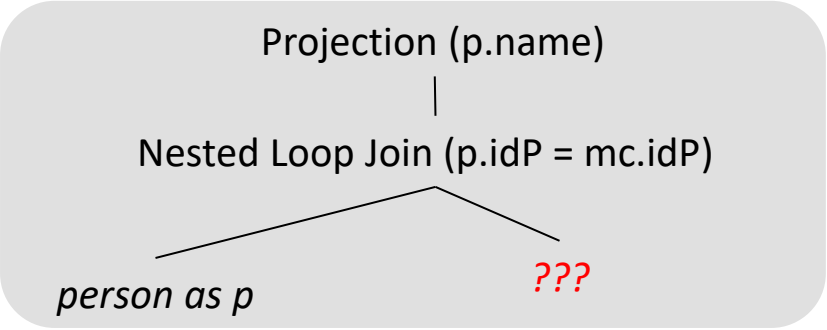
Plano A



Plano B

Nested Loop Join

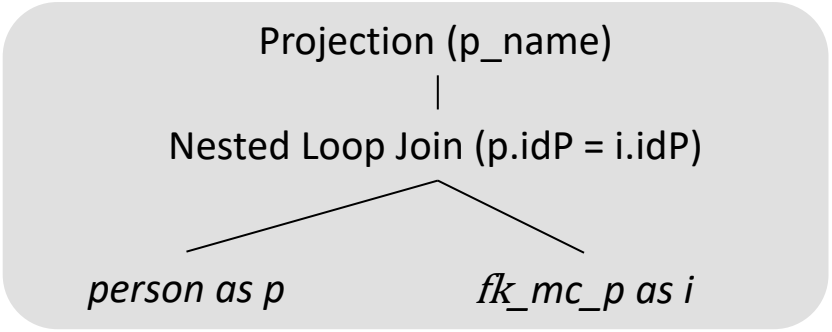
- E se fosse necessário manter person do lado externo?
- Nesse caso, seria importante usar uma estrutura no lado interno que permitisse a realização de seeks
- Solução: usar um índice secundário sobre a chave estrangeira idP



Nested Loop Join

- A estrutura fk_mc_p é um índice secundário criado sobre a chave estrangeira movie_cast.idP

- Ao usar esse índice no lado interno da junção, a busca por correspondências ocorre de forma eficiente



Plano C

Estruturas de Dados

nome	tipo	chave	valor
person	B+tree	idP	idP, p_name
movie_cast	B+tree	idM, idP	idM, idP, c_name
fk_mc_p	B+tree	idP	idM, idP

Nested Loop Join

- O exemplo anterior mostrou que índices sobre chaves estrangeiras podem ser usados pelo algoritmo Nested Loop Join

- MySQL
  - indexa automaticamente as chaves estrangeiras
- PostgreSQL
  - cabe ao DBA decidir se vale a pena indexá-las

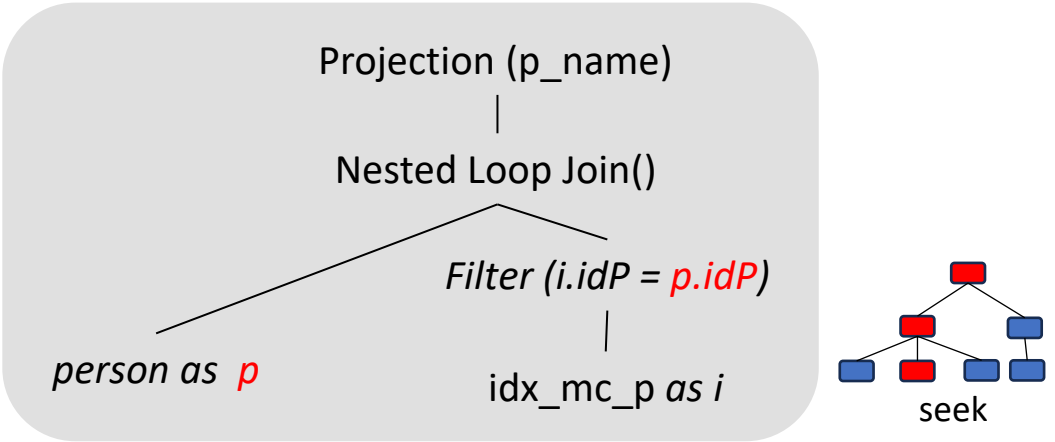
- Obs.
  - O PostgreSQL por vezes ignora índices sobre chaves estrangeiras
  - Dando preferência para outro algoritmo de junção (como veremos mais adiante)

Sumário

- Nested Loop Join
  - Colunas de Correlação
  - Memoize
- Hash Join
  - Indicações de uso
    - Ausência de índices
    - Diminuir buscas aleatórias
  - Grace Join

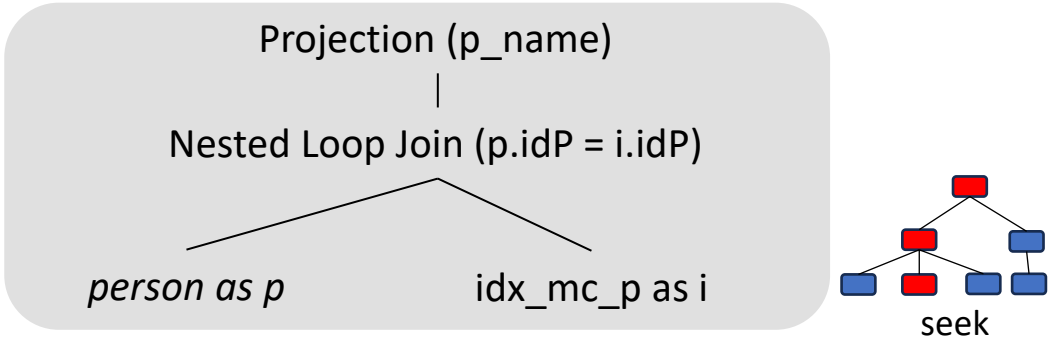
Colunas de correlação

- Uma coluna de correlação vem de uma parte mais externa da consulta
- Exemplo
  - O plano abaixo tem uma junção sem nenhum predicado
  - Já o filtro especifica o predicado que foi retirado da junção
    - Usando a coluna de correlação p.idP

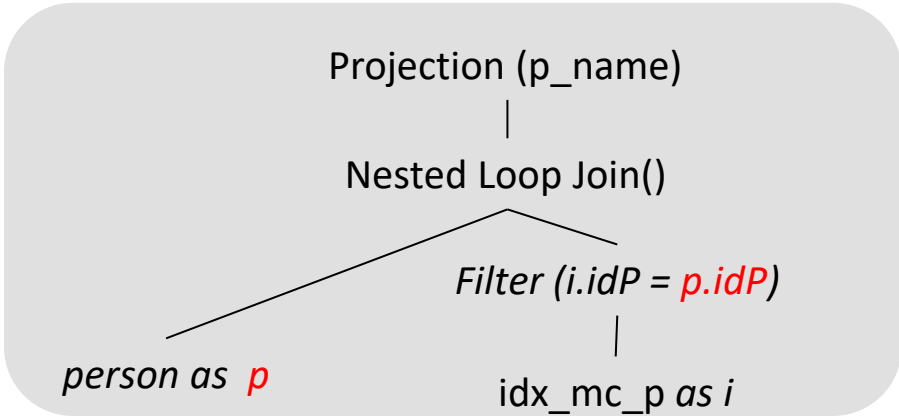


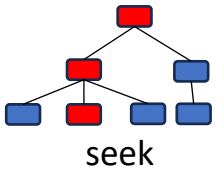
Colunas de correlação

- Qual dos planos é melhor?



Plano A

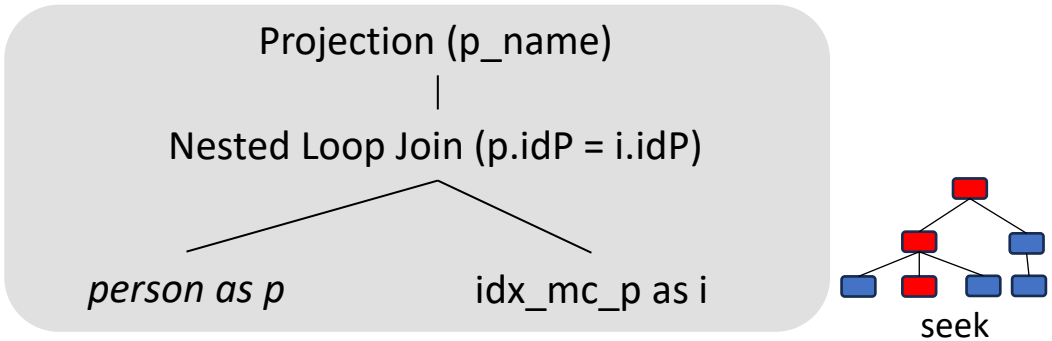




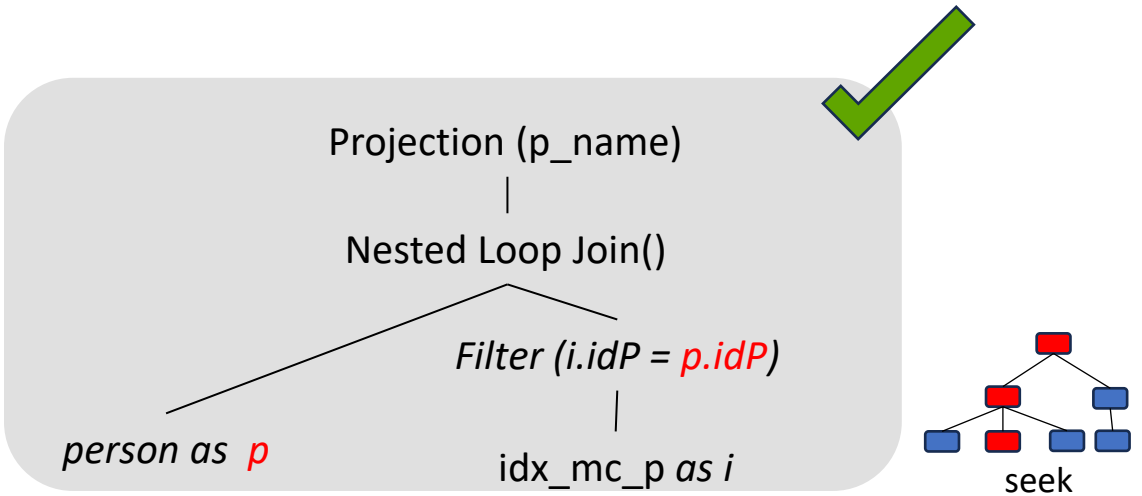
Plano B

Colunas de correlação

- Qual dos planos é melhor?
  - Em termos de flexibilidade
    - O plano B é melhor
    - Permite especificar junções diferentes de equi-joins
      - Ex. (i.dP > p.idP)



Plano A



Plano B

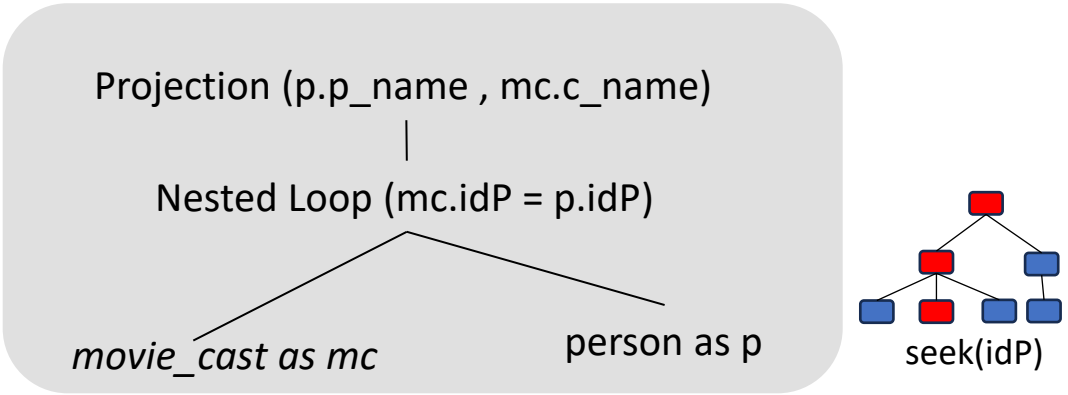
Sumário

- Nested Loop Join
  - Colunas de Correlação
  - Memoize
- Hash Join
  - Indicações de uso
    - Ausência de índices
    - Diminuir buscas aleatórias
  - Grace Join

Nested Loop & Memoize

- O plano abaixo recupera nomes de pessoas que atuaram em filmes

- Pode haver vários movie_casts para uma mesma pessoa
  - Os movie_casts de uma mesma pessoa realizarão o seek usando o mesmo idP

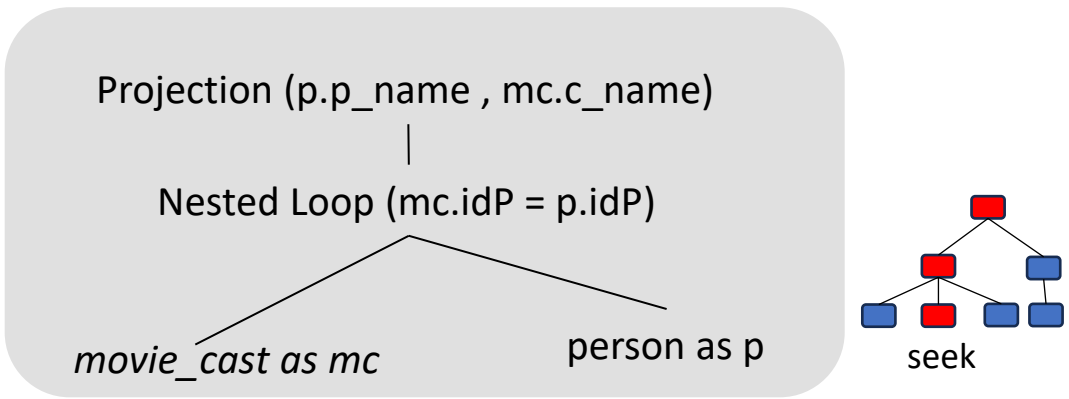


Nested Loop & Memoize

- Problema
  - A tabela movie_cast não está ordenada por idP
  - Ou seja, os seeks para um mesmo idP serão aleatórios
    - estarão distantes um do outro
  - Com isso, reduz as chances de que o registro buscado permaneça na memória
    - O que leva à necessidade de ler o registro do disco mais de uma vez

movie_cast

idM	idP	...
5	12	...
5	33	...
12	87	...
16	12	...

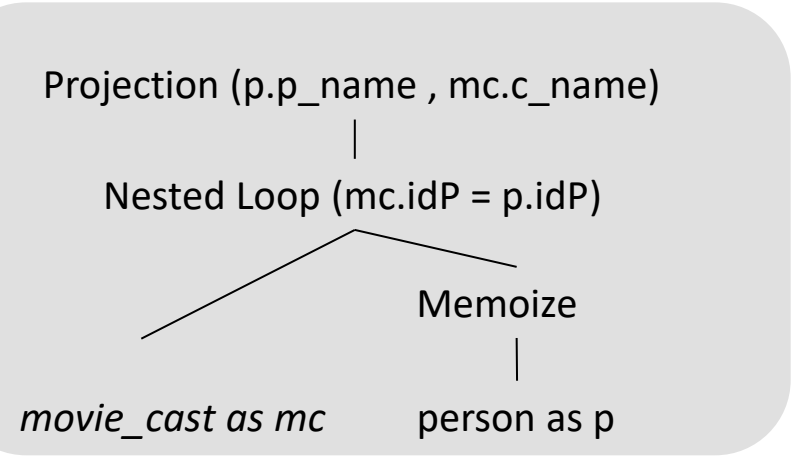


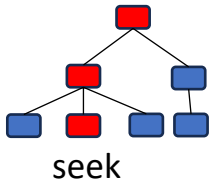
Nested Loop & Memoize

- No DBest
  - o Memoize é um operador materializado que intercepta as buscas do Nested Loop
- Na primeira vez que chega um idP
  - A busca é propagada para person
  - O resultado é armazenado em memória
- Na segunda fez que chega um idP
  - Não será necessária uma nova busca em person

movie_cast

idM	idP	...
5	12	...
5	33	...
12	87	...
16	12	...





Sumário

- Nested Loop Join
  - Colunas de Correlação
  - Memoize
- Hash Join
  - Indicações de uso
    - Ausência de índices
    - Diminuir buscas aleatórias
  - Grace Join

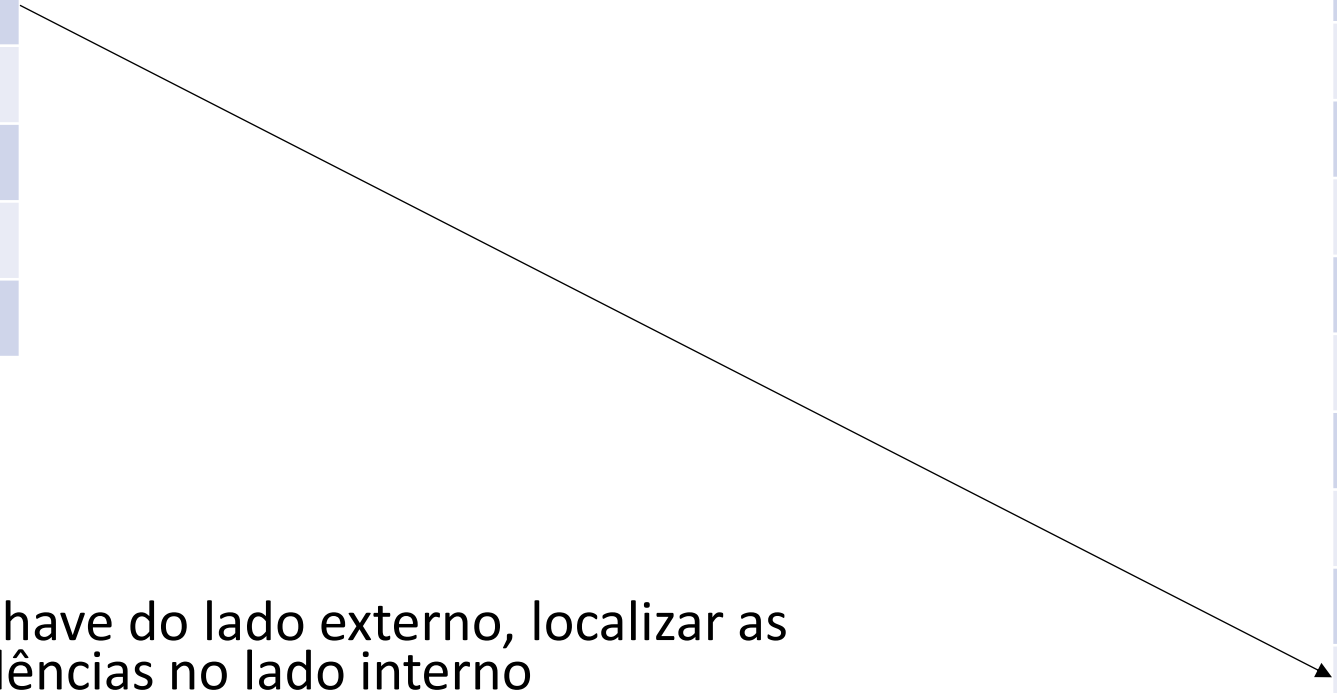
Hash Join

Lado externo

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

Lado interno

movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>



- Objetivo:
  - Para cada chave do lado externo, localizar as correspondências no lado interno
    - Sem precisar acessar os dados no meio físico

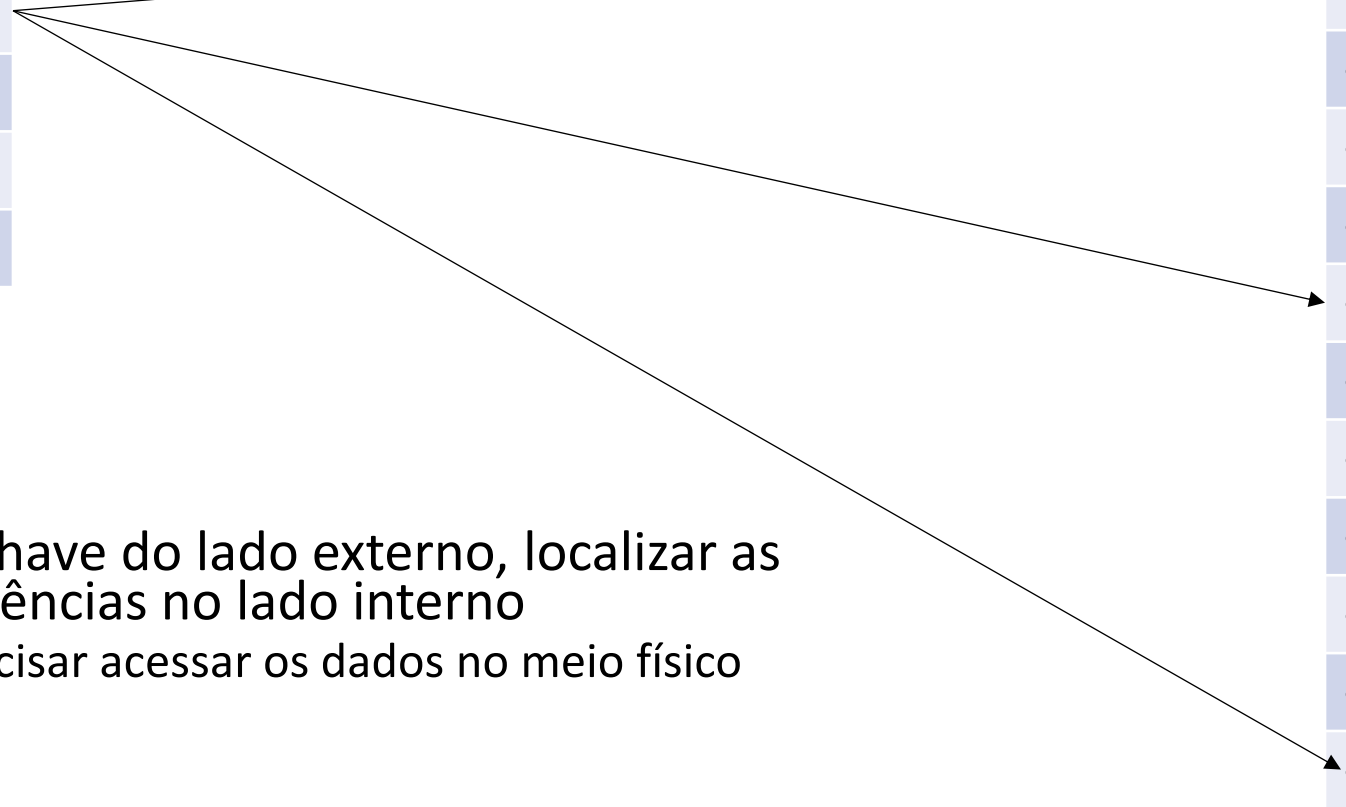
Hash Join

Lado externo

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

Lado interno

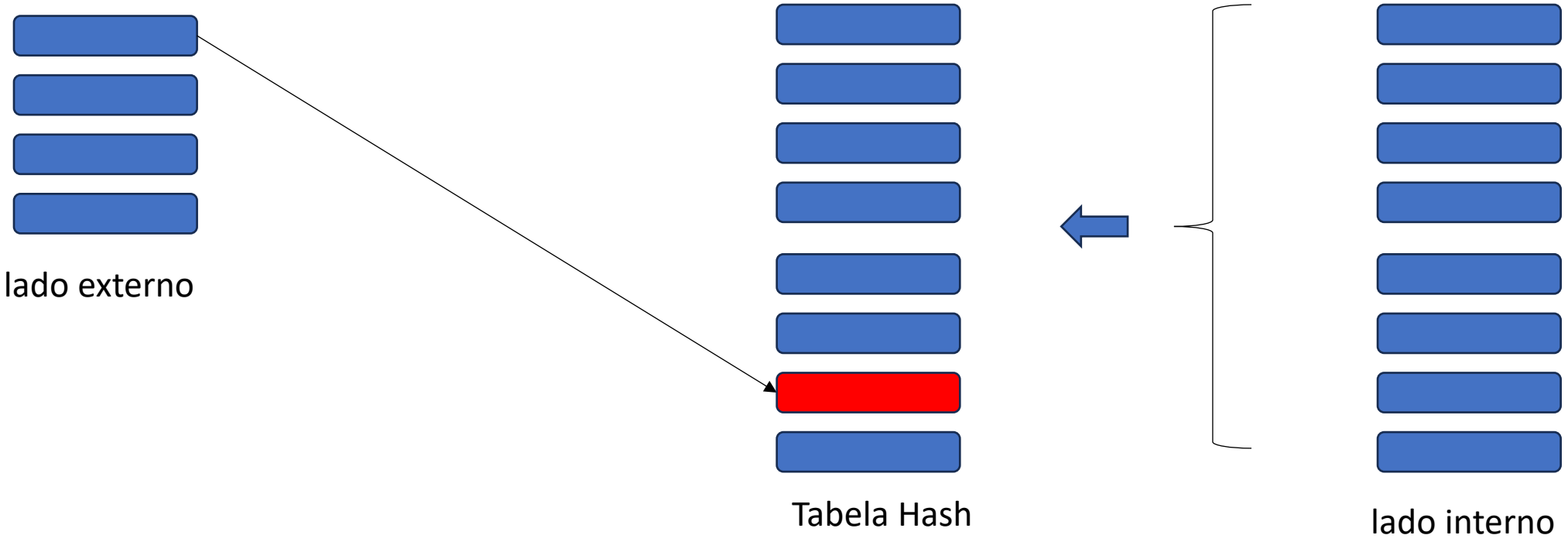
movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>



- Objetivo:
  - Para cada chave do lado externo, localizar as correspondências no lado interno
    - Sem precisar acessar os dados no meio físico

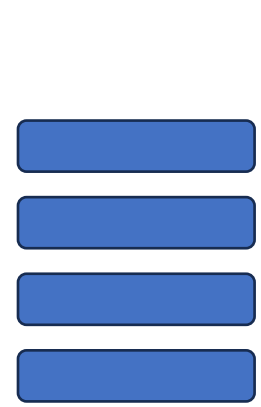
Hash Join

- Cria uma tabela hash em memória com todos os registros do lado interno
  - Para cada registro externo, realiza uma busca sobre a tabela hash
  - Inconveniente: acarreta em consumo de memória

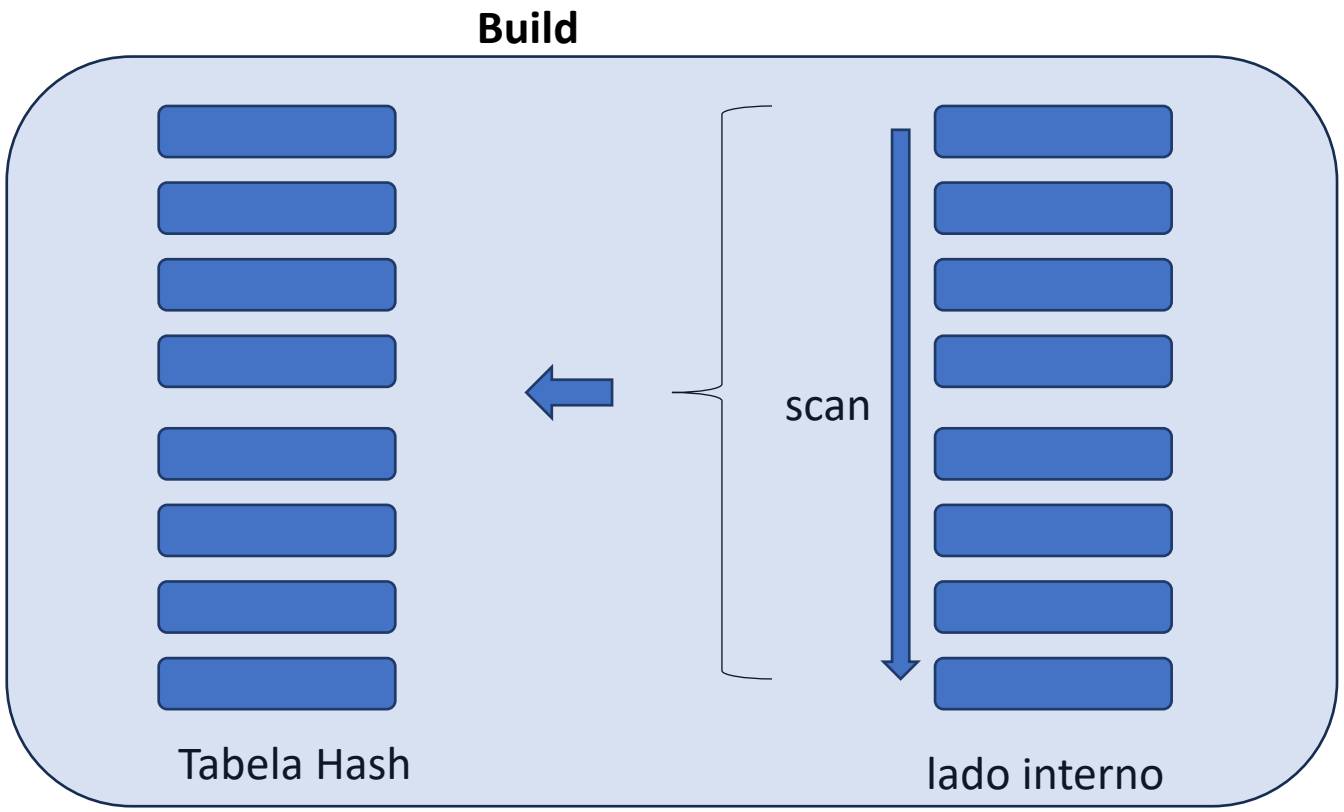


Hash Join

- Etapas
  - Build: montagem da tabela hash
    - O scan no lado interno ocorre uma única vez

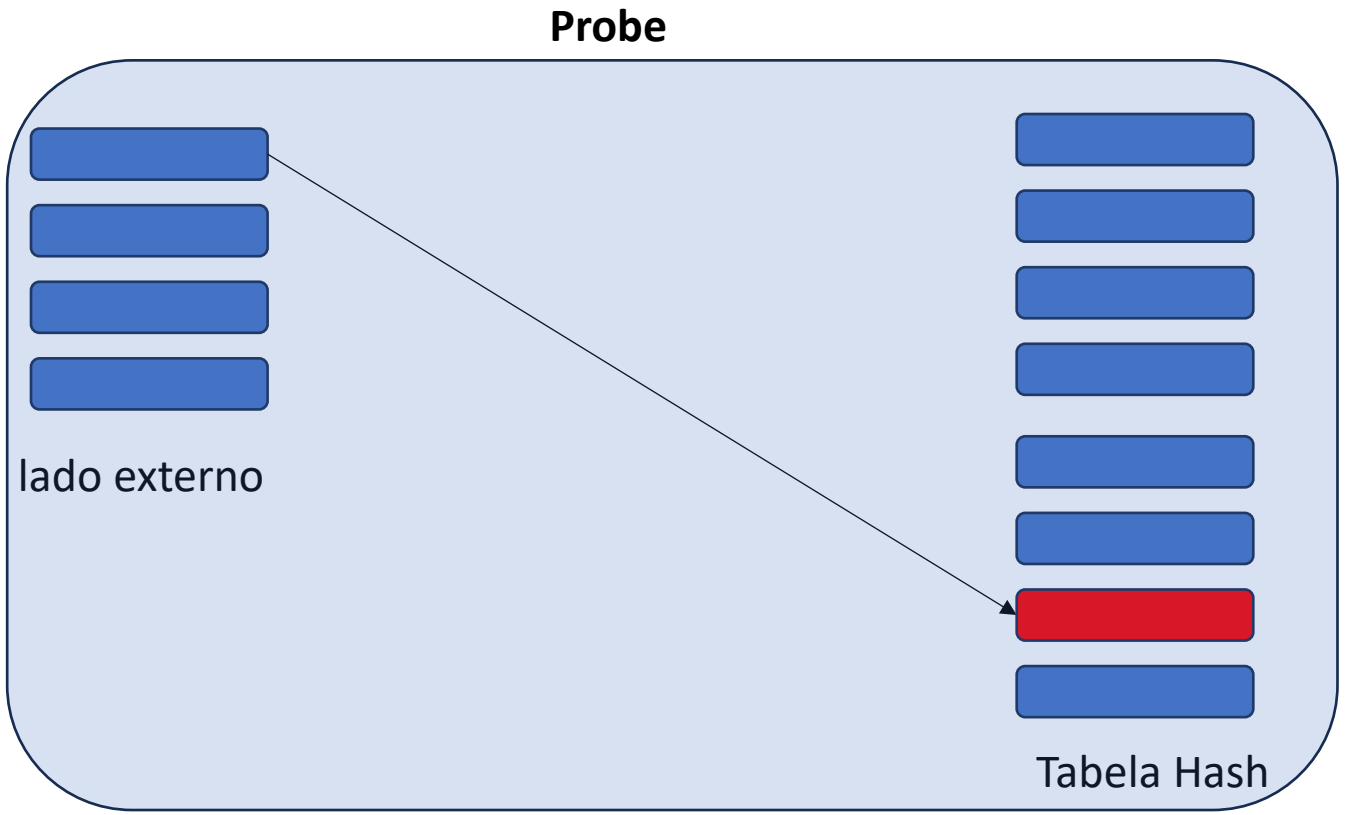


lado externo



Hash Join

- Etapas
  - Probe: busca sobre a tabela hash



Hash Join

Lado externo

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

Tabela hash

Endereço do bucket	registros
0	4, 4
1	1, 1, 5, 5, 1, 1
2	2, 2, 6, 2
3	

Função Hash(idM) = idM % 4

Lado interno

movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>

- **Fase build**: Lado interno mapeado para buckets usando uma função de hash

Hash Join

Lado externo

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

Tabela hash

Endereço do bucket	registros
0	4, 4
1	1, 1, 5, 5, 1, 1
2	2, 2, 6, 2
3	

Função Hash(idM) = idM % 4

Lado interno

movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>

- **Fase probe**:
  - Dado um registro do lado externo
    - Aplicar a função hash sobre a chave para localizar o bucket
    - comparar a chave com cada chave mapeada no bucket

Hash Join

Lado externo

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

Tabela hash

Endereço do bucket	registros
0	4, 4
1	1, 1, 5, 5, 1, 1
2	2, 2, 6, 2
3	

Função Hash(idM) = idM % 4

Lado interno

movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>

- **Fase probe**:
  - Quanto mais entradas houver em um bucket, mais custosa será essa comparação de chaves

Hash Join

Lado externo

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

Tabela hash

Endereço do bucket	registros
0	6
1	1, 1, 1, 1
2	2, 2, 2
3	
4	4, 4
5	5, 5

Função Hash(idM) = idM % 6

Lado interno

movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>

- Para reduzir o custo do probe, pode-se aumentar o número de buckets
  - Ex. hash(idM) = idM % 6

Hash Join

Lado externo

movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>

Tabela hash

Endereço do bucket	registros
0	6
1	1
2	2
3	
4	4
5	5

Função Hash(idM) = idM % 6

Lado interno

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

- Também pode-se manter no lado interno a tabela cuja chave não se repita
  - Isso leva a uma distribuição mais uniforme

Sumário

- Nested Loop Join
  - Colunas de Correlação
  - Memoize
- Hash Join
  - Indicações de uso
    - Ausência de índices
    - Diminuir buscas aleatórias
  - Grace Join

Hash Join – Indicações de uso

- O Hash Join é útil
  - Quando não existe índice sobre a coluna alvo de uma junção
  - Para evitar buscas aleatórios durante uma junção

Sumário

- Nested Loop Join
  - Colunas de Correlação
  - Memoize
- Hash Join
  - Indicações de uso
    - Ausência de índices
    - Diminuir buscas aleatórias
  - Grace Join

Hash Join – Ausência de Índices

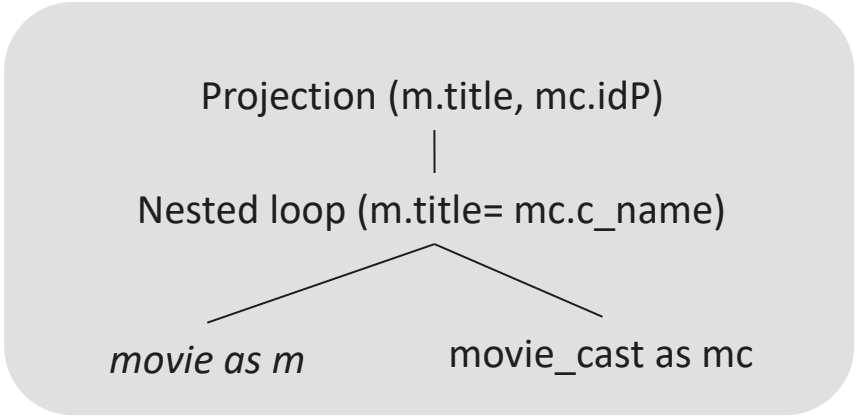
- Exemplo 1: encontrar filmes cujo título seja igual ao nome de algum personagem (mesmo que não seja do mesmo filme). Retornar o título do filme e o id da pessoa que atuou como o personagem.

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

- O critério de junção é sobre colunas não indexadas
  - title
  - c_name

Hash Join – Ausência de Índices

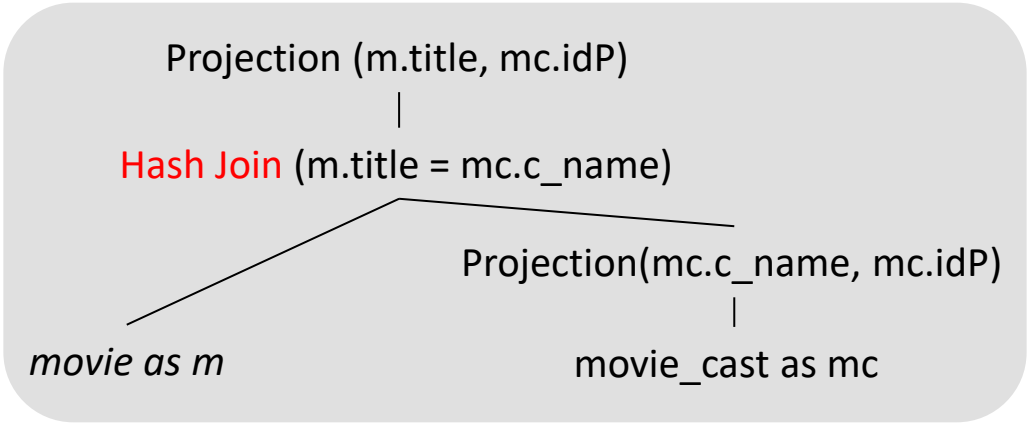
- O plano abaixo usa Nested Loop Join
- Como não existe índice a acessar
  - Para cada filme, é necessário varrer toda a tabela de movie_cast para encontrar correspondências



Plano A

Hash Join – Ausência de Índices

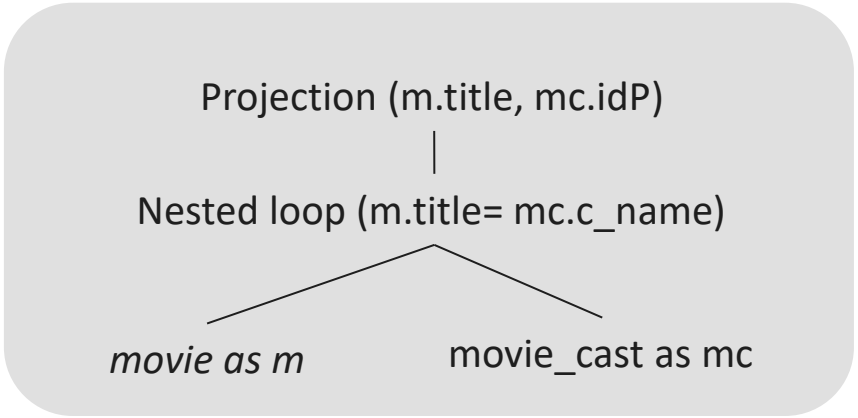
- O plano abaixo usa Hash Join
- O algoritmo
  - constrói uma tabela hash indexada por c_name
    - A tabela inclui todos os registros de movie_cast
      - Mas apenas as colunas de interesse (c_name, idP)
        - O Projection é usado para reduzir o consumo de memória
  - Realiza as buscas sobre essa tabela



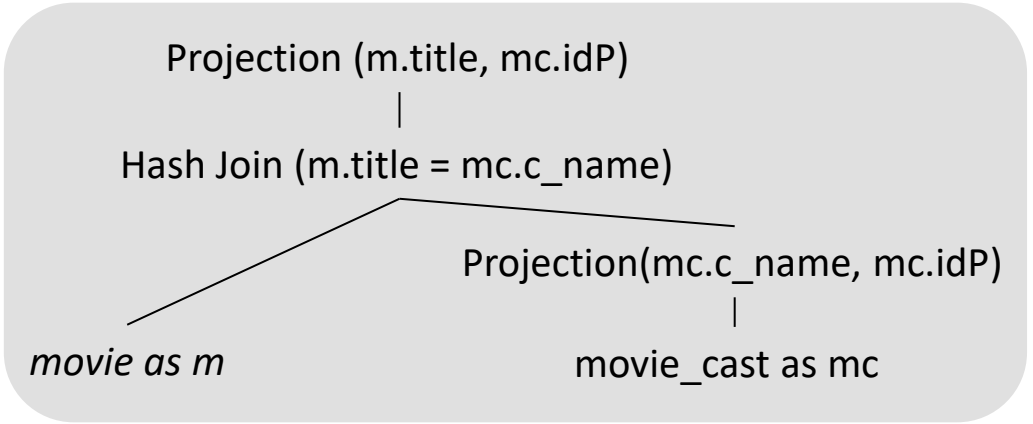
Plano B

Hash Join – Ausência de Índices

- Qual é melhor?



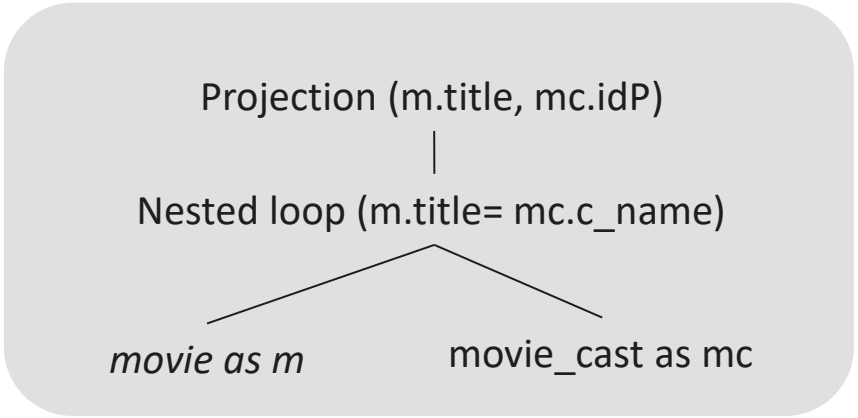
Plano A



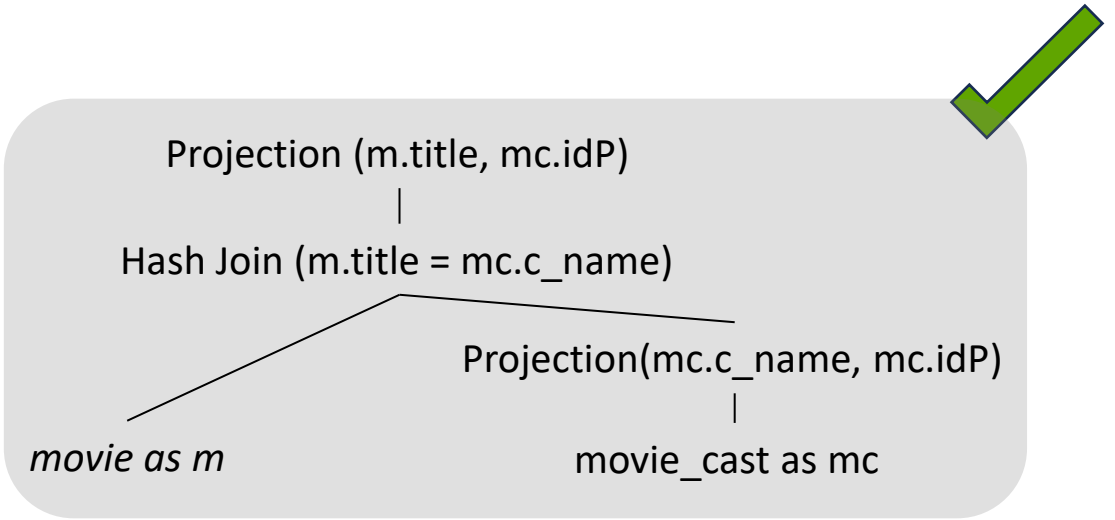
Plano B

Hash Join – Ausência de Índices

- O Hash Join é mais eficiente
  - Faz o table scan uma única vez, e não uma vez para cada movie
- Desvantagem
  - precisa carregar todos os registros de movie_cast para a memória



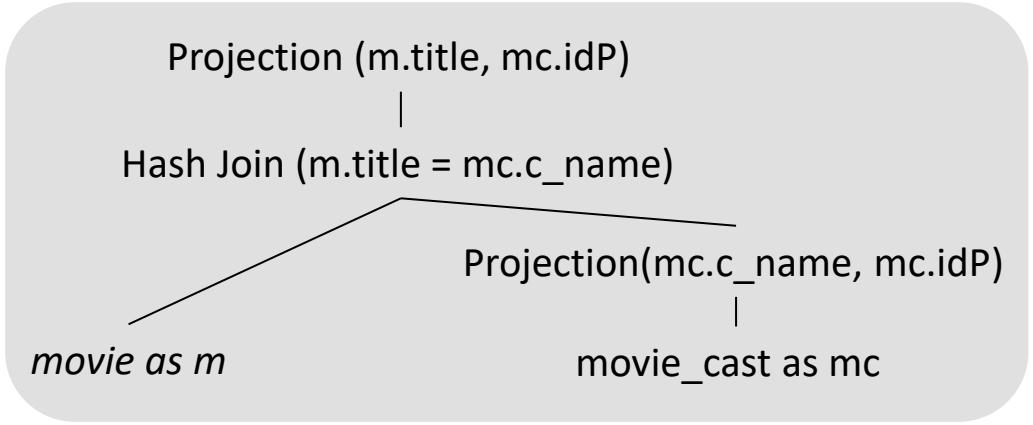
Plano A

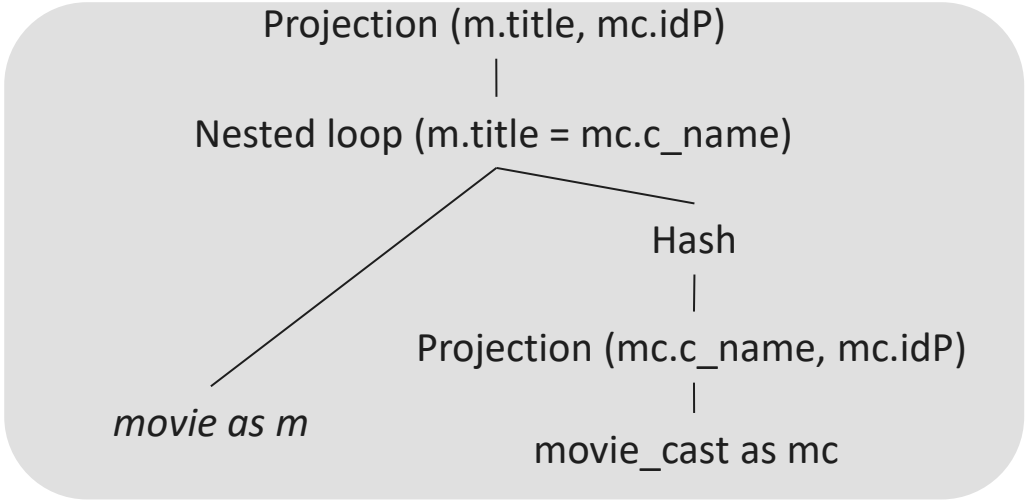


Plano B

Hash Join – Ausência de Índices

- Curiosidade
  - No DBest, é possível aplicar a mesma lógica do Hash Join usando uma combinação de dois operadores
    - Hash
      - Um operador que materializa registros em uma tabela hash
    - Nested Loop



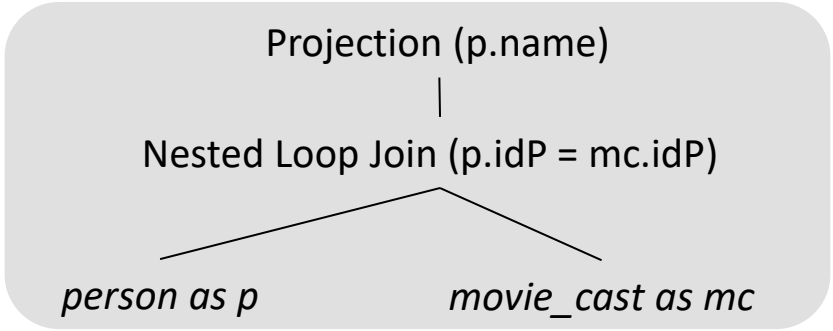


Sumário

- Nested Loop Join
  - Colunas de Correlação
  - Memoize
- Hash Join
  - Indicações de uso
    - Ausência de índices
    - Diminuir buscas aleatórias
  - Grace Join

Hash Join – Redução de Buscas aleatórias

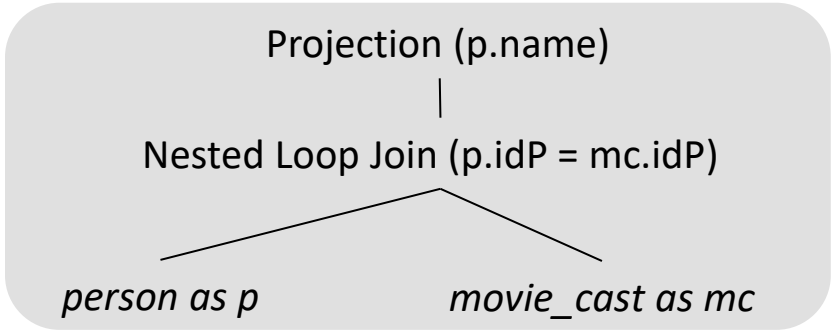
- Vimos que o plano abaixo é ruim porque a tabela movie_cast não tem idP como prefixo da chave de busca
- Se realmente for importante manter person do lado externo, é necessário pensar em alguma alternativa
  - Uma delas é o uso de índice sobre a chave estrangeira (como vimos)
  - Outra opção é o uso do algoritmo de Hash Join



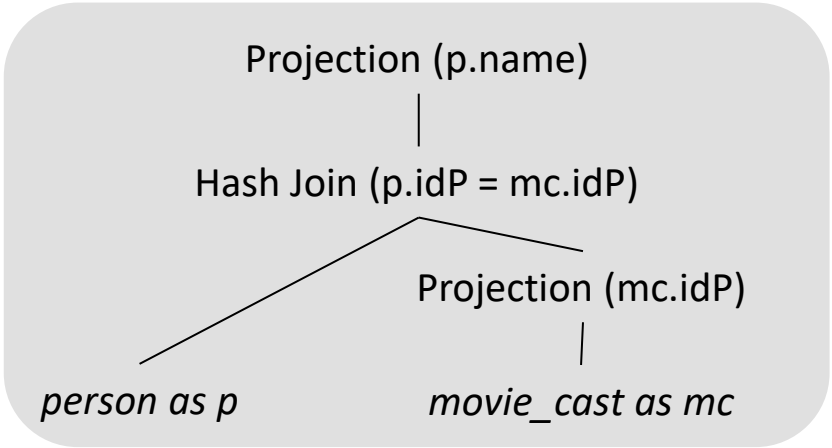
Plano A

Hash Join – Redução de Buscas aleatórias

- Qual é melhor?
  - Plano A: Usa Nested Loop Join, com movie_cast do lado interno
  - Plano B: Usa Hash Join, com movie_cast do lado interno



Plano A

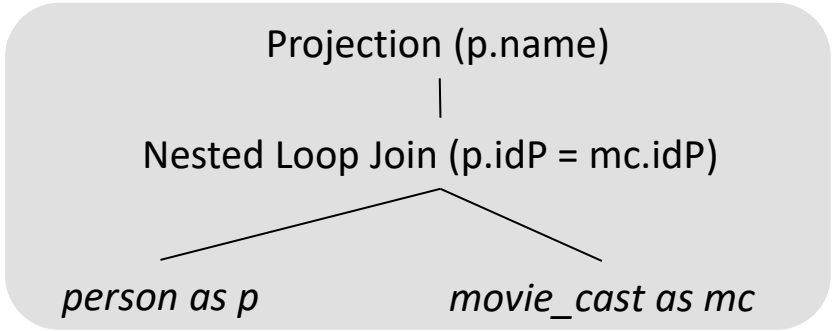


Plano B

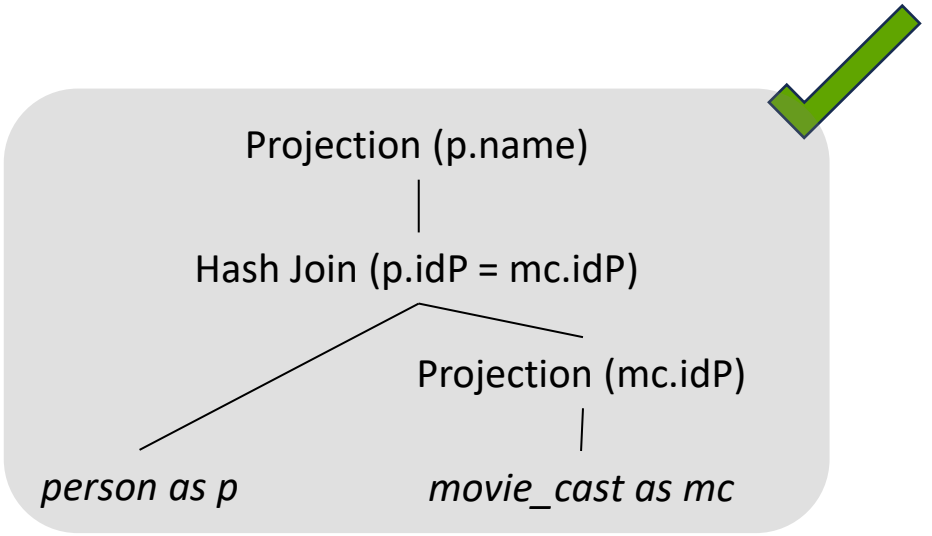
Hash Join – Redução de Buscas aleatórias

- Hash Join é melhor
  - Em vez de fazer um scan por person, é realizado um único scan
  - A partir de então, as buscas ocorrem sobre a tabela hash em memória

  - Desvantagem: Precisa materializar informações de movie_cast (idP)



Plano A



Plano B

Sumário

- Nested Loop Join
  - Colunas de Correlação
  - Memoize
- Hash Join
  - Indicações de uso
    - Ausência de índices
    - Diminuir buscas aleatórias
  - Grace Join

Grace Join

- Variação do Hash Join usada quando não há espaço para toda a relação interna R na memória M (ou seja, caso R>M)

- Fases do Grace Join
  - Build
    - Uma função hash divide a relação interna em buckets
    - **A mesma função hash divide a relação externa em buckets**
  - Probe
    - Processa um bucket de cada vez.

Grace Join

- Quantidade de buckets:
  - Um número que visa fazer cada bucket ser o maior possível sem provocar estouro de memória
  - Fórmula para a quantidade: $\left\lceil \frac{R}{M} * F \right\rceil$
    - Onde:
      - R = tamanho da tabela a ser particionada
      - M = tamanho da memória
      - F = fator de fudge
- O fator de fudge é uma margem de proteção para evitar estouros.
  - Exemplo de valor: 1,2
    - Significa que haverá 20% mais buckets do que o estimado

Grace Join

Função Hash(idM) = idM % 3

Lado externo

movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>

Lado interno

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

Grace Join

Função Hash(idM) = idM % 3

Lado externo

movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>

Hash

Bucket 1
<1>, <...>
<1>, <...>
<4>, <...>
<1>, <...>
<1>, <...>
<4>, <...>

Bucket 2
<2>, <...>
<5>, <...>
<2>, <...>
<5>, <...>
<2>, <...>

Bucket 3
<6>, <...>

Lado interno

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

Lado externo é dividido em buckets

Grace Join

Função Hash(idM) = idM % 3

Lado externo

movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>

Hash

Bucket 1
<1>, <...>
<1>, <...>
<4>, <...>
<1>, <...>
<1>, <...>
<4>, <...>

Bucket 2
<2>, <...>
<5>, <...>
<2>, <...>
<5>, <...>
<2>, <...>

Bucket 3
<6>, <...>

Lado interno

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

Hash

Bucket 1
<1>, <...>
<4>, <...>

Bucket 2
<2>, <...>
<5>, <...>

Bucket 3
<6>, <...>

Lado interno também é dividido em buckets, usando a mesma função de hash

Grace Join

Função Hash(idM) = idM % 3

Lado externo

movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>

Hash

Bucket 1
<1>, <...>
<1>, <...>
<4>, <...>
<1>, <...>
<1>, <...>
<4>, <...>

Bucket 2
<2>, <...>
<5>, <...>
<2>, <...>
<5>, <...>
<2>, <...>

Bucket 3
<6>, <...>

Lado interno

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

Hash

Bucket 1
<1>, <...>
<4>, <...>

Bucket 2
<2>, <...>
<5>, <...>

Bucket 3
<6>, <...>

Cada par de buckets é processado

Grace Join

Função Hash(idM) = idM % 3

Lado externo

movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>

Hash

Bucket 1
<1>, <...>
<1>, <...>
<4>, <...>
<1>, <...>
<1>, <...>
<4>, <...>

Bucket 2
<2>, <...>
<5>, <...>
<2>, <...>
<5>, <...>
<2>, <...>

Bucket 3
<6>, <...>

Lado interno

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

Hash

Bucket 1
<1>, <...>
<4>, <...>

Bucket 2
<2>, <...>
<5>, <...>

Bucket 3
<6>, <...>

Cada par de buckets é processado

Grace Join

Função Hash(idM) = idM % 3

Lado externo

movie_cast
<2>, <...>
<1>, <...>
<1>, <...>
<5>, <...>
<4>, <...>
<2>, <...>
<5>, <...>
<1>, <...>
<1>, <...>
<6>, <...>
<4>, <...>
<2>, <...>

Hash

Bucket 1
<1>, <...>
<1>, <...>
<4>, <...>
<1>, <...>
<1>, <...>
<4>, <...>

Bucket 2
<2>, <...>
<5>, <...>
<2>, <...>
<5>, <...>
<2>, <...>

Bucket 3
<6>, <...>

Lado interno

movie
<6>, <...>
<2>, <...>
<1>, <...>
<5>, <...>
<4>, <...>

Hash

Bucket 1
<1>, <...>
<4>, <...>

Bucket 2
<2>, <...>
<5>, <...>

Bucket 3
<6>, <...>

Cada par de buckets é processado

Encerrando

- O Hash Join agiliza os seeks
  - Contudo, traz como consequência o consumo de memória

- O otimizador de consultas decide se seu uso compensa
- Por exemplo
  - MySQL
    - Usa Hash Join se não existirem índices disponíveis e em outros casos isolados
  - PostgreSQL
    - Pode usar Hash Join mesmo havendo índices disponíveis
      - A depender da disponibilidade de memória

Encerrando

- A tabela abaixo mostra a escolha preferencial desses dois bancos de dados

Aspecto		MySQL	PostgreSQL
Existem índices para a junção	Consulta seletiva	Indexed Nested Loop Join	Indexed Nested Loop Join
	Consulta pouco seletiva	Indexed Nested Loop Join	Hash Join
Não existem índices para a junção		Hash Join	Hash Join

Atividade Individual

- Escreva dois planos de consulta alternativos que retornem os títulos dos filmes em que um ator interpretou a si mesmo. O resultado também deve incluir o nome do ator.

- Explique por que um plano é melhor do que o outro