Planos de execução

Sumário

- Introdução
- Planos de execução em SGBDs
- DBest
- Junções
  - Nested Loop
  - Hash Join
- Filtros
  - Sem uso de índice
  - Com uso de índice
  - Covering Index
  - Filtro disjuntivo
  - Filtro em junções

Introdução

Dataset utilizado

movie (movie_id, title, year)

person (person_id, name)

movie_cast (movie_id, person_id, cast_order, character_name)
- movie_id references movie
- person_id references person

movie_crew (movie_id, person_id, job)
- movie_id references movie
- person_id references person

Introdução

- Escrever uma consulta que retorne os títulos de filmes onde Brad Pitt trabalhou como ator

Introdução

- Escrever uma consulta que retorne os títulos de filmes onde Brad Pitt trabalhou como ator

resposta

SQL:

```
select distinct m.title
from movie m

join movie_cast mc on m.movie_id = mc.movie_id
join person p on mc.person_id = p.person_id

where p.person_name='Brad Pitt';
```

Introdução

- Escrever uma consulta que retorne os títulos de filmes onde Brad Pitt trabalhou como membro da equipe de produção

Introdução

- Escrever uma consulta que retorne os títulos de filmes onde Brad Pitt trabalhou como membro da equipe de produção

resposta

SQL:

```
select distinct m.title
from movie m

join movie_crew mc on m.movie_id = mc.movie_id
join person p on mc.person_id = p.person_id

where p.person_name='Brad Pitt';
```

Introdução

- Escrever uma consulta que retorne os títulos de filmes onde Brad Pitt trabalhou como ator ou como membro da equipe

Introdução

- Escrever uma consulta que retorne os títulos de filmes onde Brad Pitt trabalhou como ator ou como membro da equipe

resposta

SQL:

```
select distinct title
from movie  m

join movie_cast m_cast on m.movie_id = m_cast.movie_id
join person p_cast on m_cast.person_id = p_cast.person_id

join  movie_crew m_crew on m.movie_id = m_crew.movie_id
join  person p_crew on m_crew.person_id = p_crew.person_id

where (p_cast.person_name = 'Brad Pitt' OR p_crew.person_name = 'Brad Pitt')
order by 1;
```

Introdução

- Escrever uma consulta que retorne os títulos de filmes onde Brad Pitt trabalhou como ator ou como membro da equipe

resposta

SQL:

```
select distinct title
from movie  m

join movie_cast m_cast on m.movie_id = m_cast.movie_id
join person p_cast on m_cast.person_id = p_cast.person_id

join  movie_crew m_crew on m.movie_id = m_crew.movie_id
join  person p_crew on m_crew.person_id = p_crew.person_id

where (p_cast.person_name = 'Brad Pitt' OR p_crew.person_name = 'Brad Pitt')
order by 1;
```

Consulta levemente demorada.

Qual o motivo da demora?

Introdução

- Os planos de execução de consulta ajudam a entender porque uma consulta é demorada

- Aspectos presentes em um plano
  - Algoritmos usados
  - A ordem das operações
  - O uso de índices
  - Indicadores de custo

Introdução

- Mas como descobrir se um plano é adequado?
- O que deve ser feito para que o plano seja alterado?
- Simplesmente olhar para o plano não é bastante
  - É necessário interpretá-lo.

- Nessa aula veremos questões básicas referentes à relação entre consultas SQL e planos de execução
  - Essa compreensão é importante para escrever consultas melhores

Sumário

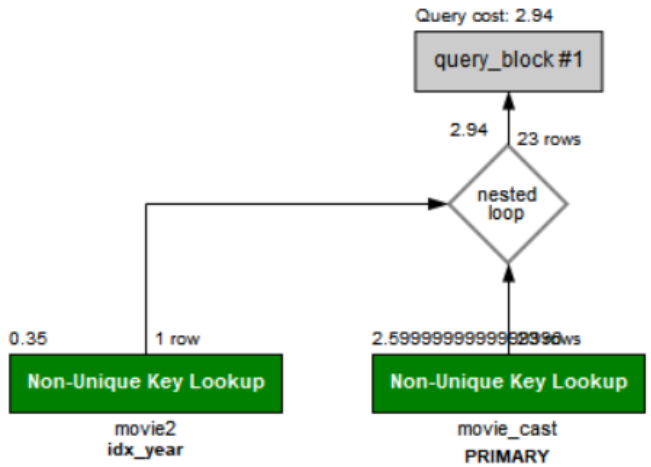
- Introdução
- Planos de execução em SGBDs
- Dbest
- Junções
  - Nested Loop
  - Hash Join
- Filtros
  - Sem uso de índice
  - Com uso de índice
  - Covering Index
  - Filtro disjuntivo
  - Filtro em junções

Exemplos em MySQL e PostgreSQL

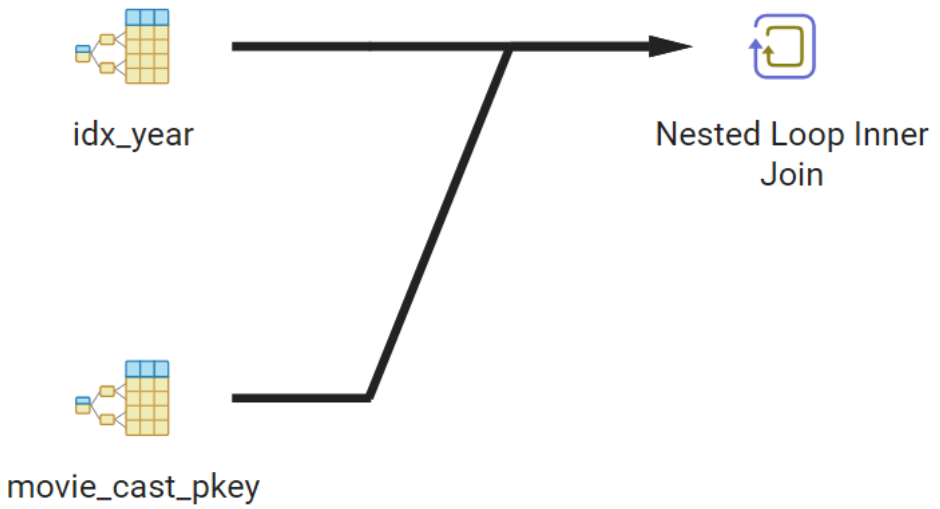
- SGBDs criam e apresentam planos de execução de formas diferentes

```
SELECT * FROM movie2 JOIN movie_cast USING(movie_id)
WHERE release_year = 1950;
```

MySQL



PostgreSQL



Plano de execução escrito

- Além da versão visual, também é possível acessar a versão escrita de um plano de execução por meio do comando EXPLAIN ANALYZE

```
EXPLAIN ANALYZE SELECT * FROM movie2
JOIN movie_cast USING(movie_id)
WHERE release_year = 1950;
```

Exemplo em MySQL:

-> Nested loop inner join (cost=2.94 rows=23.4) (actual time=0.946..0.957 rows=6 loops=1)
    -> Index lookup on movie2 using idx_year (release_year=1950) (cost=0.35 rows=1) (actual time=0.225..0.226 rows=1 loops=1)
    -> Index lookup on movie_cast using PRIMARY (movie_id=movie2.movie_id) (cost=2.59 rows=23.4) (actual time=0.717..0.726 rows=6 loops=1)

Plano de execução escrito

```
EXPLAIN ANALYZE SELECT * FROM movie2
JOIN movie_cast USING(movie_id)
WHERE release_year = 1950;
```

Exemplo em MySQL:

-> Nested loop inner join (cost=2.94 rows=23.4) (actual time=0.946..0.957 rows=6 loops=1)
  -> Index lookup on movie2 using idx_year (release_year=1950) (cost=0.35 rows=1) (actual time=0.225..0.226 rows=1 loops=1)
  -> Index lookup on movie_cast using PRIMARY (movie_id=movie2.movie_id) (cost=2.59 rows=23.4) (actual time=0.717..0.726 rows=6 loops=1)

O plano escrito descreve que operadores foram utilizados e quais são os seus parâmetros

Plano de execução escrito

```
EXPLAIN ANALYZE SELECT * FROM movie2
JOIN movie_cast USING(movie_id)
WHERE release_year = 1950;
```

Exemplo em MySQL:

```
-> Nested loop inner join  (cost=2.94 rows=23.4) (actual time=0.946..0.957 rows=6 loops=1)
    -> Index lookup on movie2 using idx_year (release_year=1950)  (cost=0.35 rows=1) (actual time=0.225..0.226 rows=1 loops=1)
    -> Index lookup on movie_cast using PRIMARY (movie_id=movie2.movie_id)  (cost=2.59 rows=23.4) (actual time=0.717..0.726 rows=6 loops=1)
```

O plano de execução é uma árvore.
A tabulação das setas indica a hierarquia dos operadores que compõe a árvore.

Plano de execução escrito

```
EXPLAIN ANALYZE SELECT * FROM movie2
JOIN movie_cast USING(movie_id)
WHERE release_year = 1950;
```

Exemplo em MySQL:

-> Nested loop inner join (cost=2.94 rows=23.4) (actual time=0.946..0.957 rows=6 loops=1)
    -> Index lookup on movie2 using idx_year (release_year=1950) (cost=0.35 rows=1) (actual time=0.225..0.226 rows=1 loops=1)
    -> Index lookup on movie_cast using PRIMARY (movie_id=movie2.movie_id) (cost=2.59 rows=23.4) (actual time=0.717..0.726 rows=6 loops=1)

Custo estimado de cada operador: Quanto maior, mais custoso é

Plano de execução escrito

```
EXPLAIN ANALYZE SELECT * FROM movie2
JOIN movie_cast USING(movie_id)
WHERE release_year = 1950;
```

Exemplo em MySQL:

-> Nested loop inner join (cost=2.94 rows=23.4) (actual time=0.946..0.957 rows=6 loops=1)
   -> Index lookup on movie2 using idx_year (release_year=1950) (cost=0.35 rows=1) (actual time=0.225..0.226 rows=1 loops=1)
   -> Index lookup on movie_cast using PRIMARY (movie_id=movie2.movie_id) (cost=2.59 rows=23.4) (actual time=0.717..0.726 rows=6 loops=1)

Quantidade estimada de registros recuperados por cada operação, antes da operação ser executada

Plano de execução escrito

```
EXPLAIN ANALYZE SELECT * FROM movie2
JOIN movie_cast USING(movie_id)
WHERE release_year = 1950;
```

Exemplo em MySQL:

-> Nested loop inner join (cost=2.94 rows=23.4) (actual time=0.946..0.957 rows=6 loops=1)
  -> Index lookup on movie2 using idx_year (release_year=1950) (cost=0.35 rows=1) (actual time=0.225..0.226 rows=1 loops=1)
  -> Index lookup on movie_cast using PRIMARY (movie_id=movie2.movie_id) (cost=2.59 rows=23.4) (actual time=0.717..0.726 rows=6 loops=1)

Tempo efetivamente gasto com cada operação, em milisegundos
O primeiro valor é o tempo para obter o primeiro registro.
O segundo valor é o tempo para obter todos os registros

Plano de execução escrito

```
EXPLAIN ANALYZE SELECT * FROM movie2
JOIN movie_cast USING(movie_id)
WHERE release_year = 1950;
```

Exemplo em MySQL:

-> Nested loop inner join (cost=2.94 rows=23.4) (actual time=0.946..0.957 rows=6 loops=1)
   -> Index lookup on movie2 using idx_year (release_year=1950) (cost=0.35 rows=1) (actual time=0.225..0.226 rows=1 loops=1)
   -> Index lookup on movie_cast using PRIMARY (movie_id=movie2.movie_id) (cost=2.59 rows=23.4) (actual time=0.717..0.726 rows=6 loops=1)

Quantidade de registro efetivamente recuperados por cada operação

Plano de execução escrito

```
EXPLAIN ANALYZE SELECT * FROM movie2
JOIN movie_cast USING(movie_id)
WHERE release_year = 1950;
```

Exemplo em MySQL:

-> Nested loop inner join (cost=2.94 rows=23.4) (actual time=0.946..0.957 rows=6 loops=1)
  -> Index lookup on movie2 using idx_year (release_year=1950) (cost=0.35 rows=1) (actual time=0.225..0.226 rows=1 loops=1)
  -> Index lookup on movie_cast using PRIMARY (movie_id=movie2.movie_id) (cost=2.59 rows=23.4) (actual time=0.717..0.726 rows=6 loops=1)

Na versão textual aparece a etapa em que o filtro foi realizado.

O plano visual esconde essa informação

Sumário

- Introdução
- Planos de execução em SGBDs
- DBest
- Junções
  - Nested Loop
  - Hash Join
- Filtros
  - Sem uso de índice
  - Com uso de índice
  - Covering Index
  - Filtro disjuntivo
  - Filtro em junções

DBest

- Cada SGBD possui uma terminologia e um formato de visualização próprios

- Nesta aula
  - usaremos o formato e terminologia usados na ferramenta DBest
  - E faremos comparações com os planos de execução gerados pelo MySQL e PostgreSQL

DBest

- DBest é uma ferramenta que permite ao usuário
  - montar planos de execução de consulta por meio de uma interface visual
  - testar os planos e analisar as diferenças entre eles

- A ferramenta disponibiliza muitas operações
  - No decorrer das aulas veremos algumas possibilidades

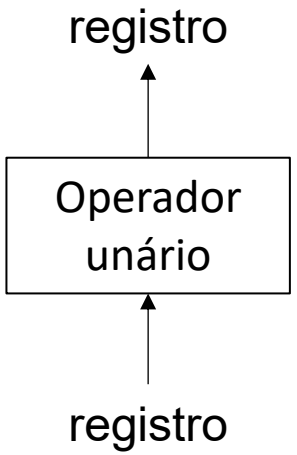
DBest

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

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

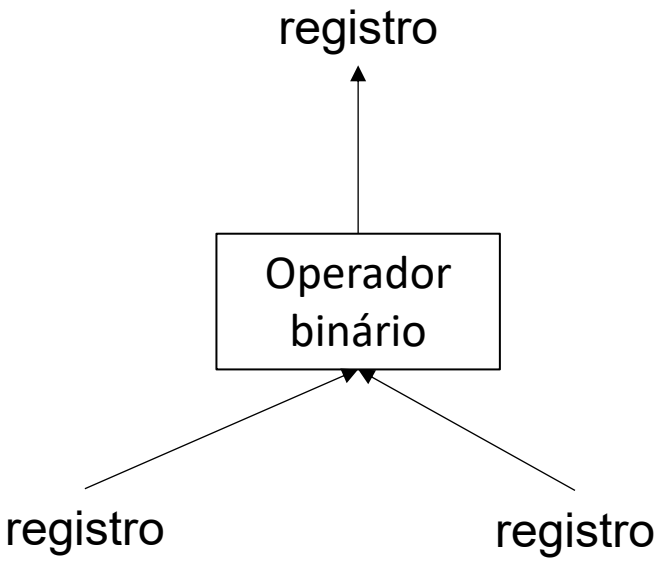
DBest

- Operadores unários
  - Recebem um registro na entrada e devolvem um registro na saída



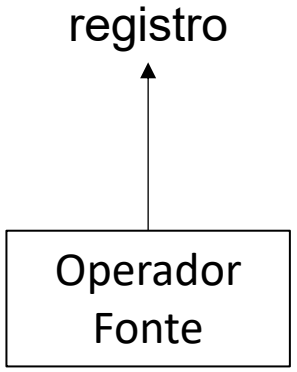
DBest

- Operadores binários
  - Recebem dois registros na entrada e devolvem um registro na saída



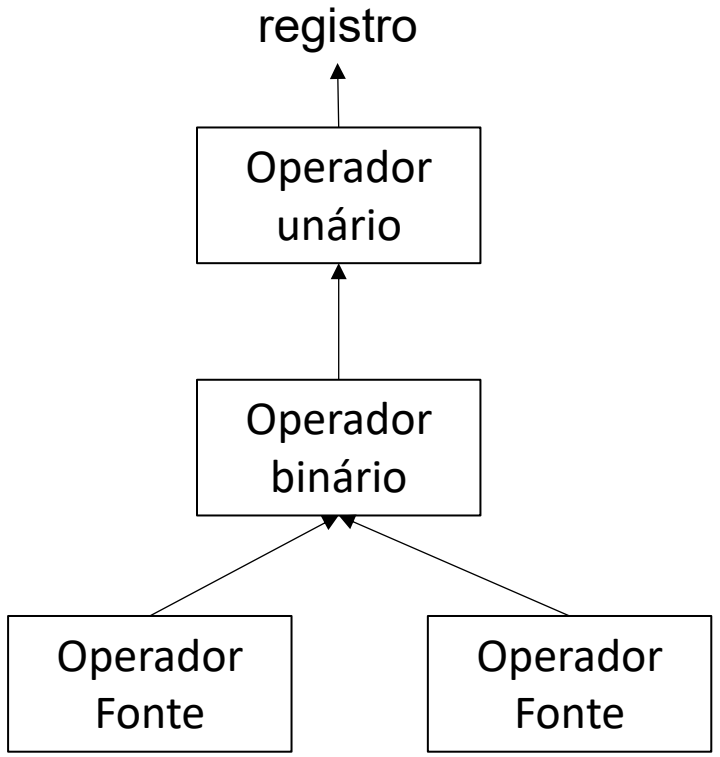
DBest

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



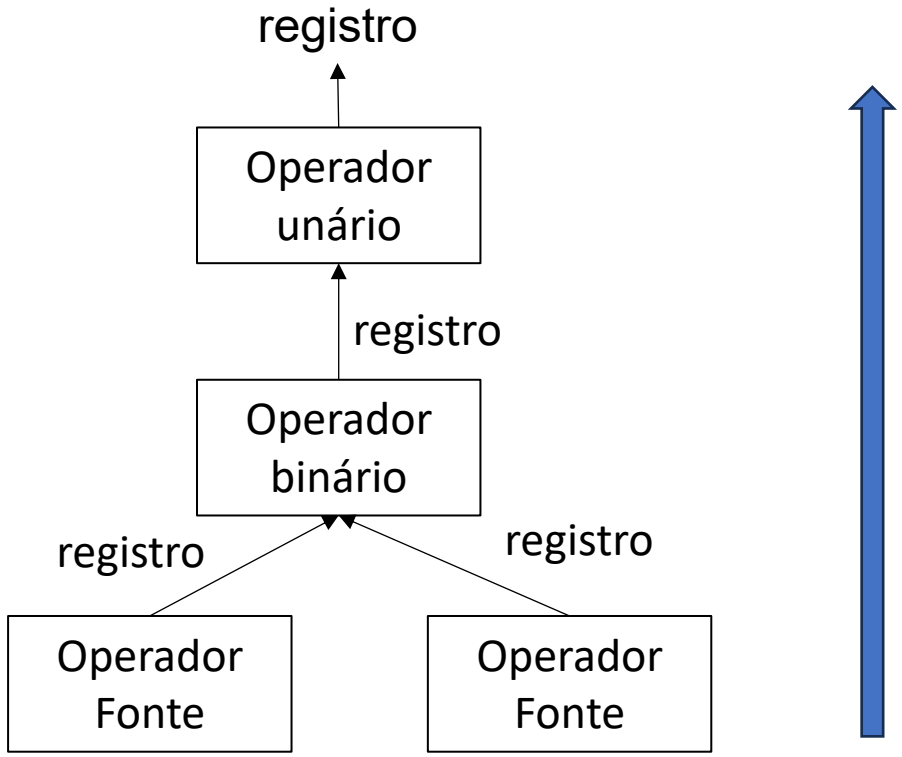
DBest

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



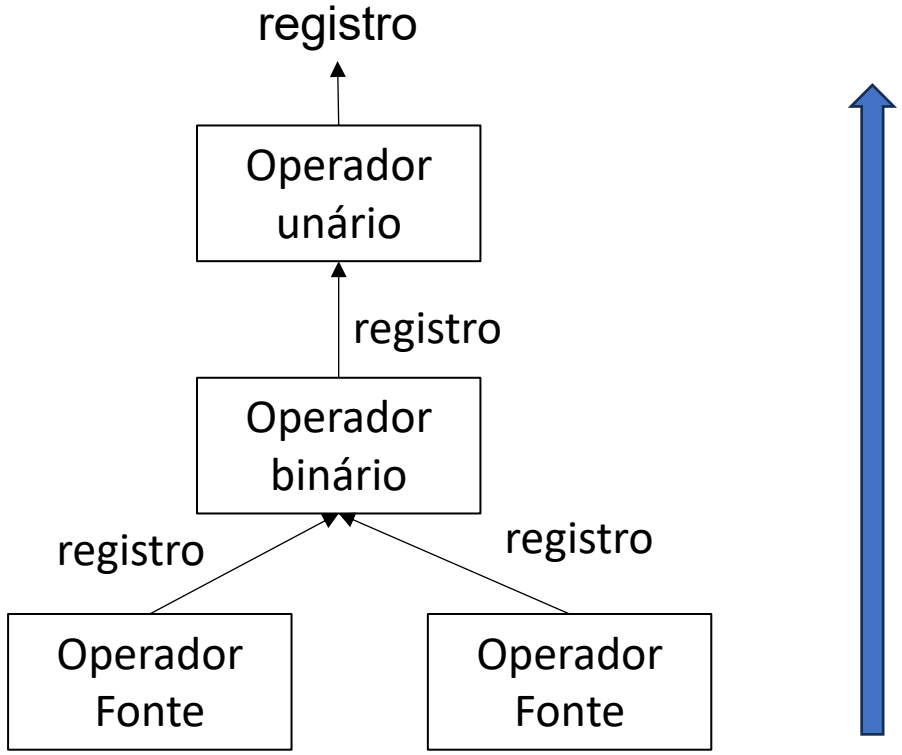
DBest

- Os registros fluem dos operadores fonte em direção aos operadores a eles conectados
- No caso da figura ao lado, os registros estão fluindo de baixo para cima
- Essa arquitetura de propagação dos registros é chamada de Volcano



DBest

- Os registros são gerados e combinados um de cada vez, a medida que forem acessados dos operadores fonte
  - Fluxo de execução por pipeline
- Operadores também podem ser do tipo materializados
  - São chamados assim porque mantém uma cópia de todos os registros
    - Essa cópia acarreta em consumo de memória
- Operadores materializados não seguem o fluxo de pipeline
  - Todos os registros são lidos primeiro
  - Só então a cópia é usada para responder requisições

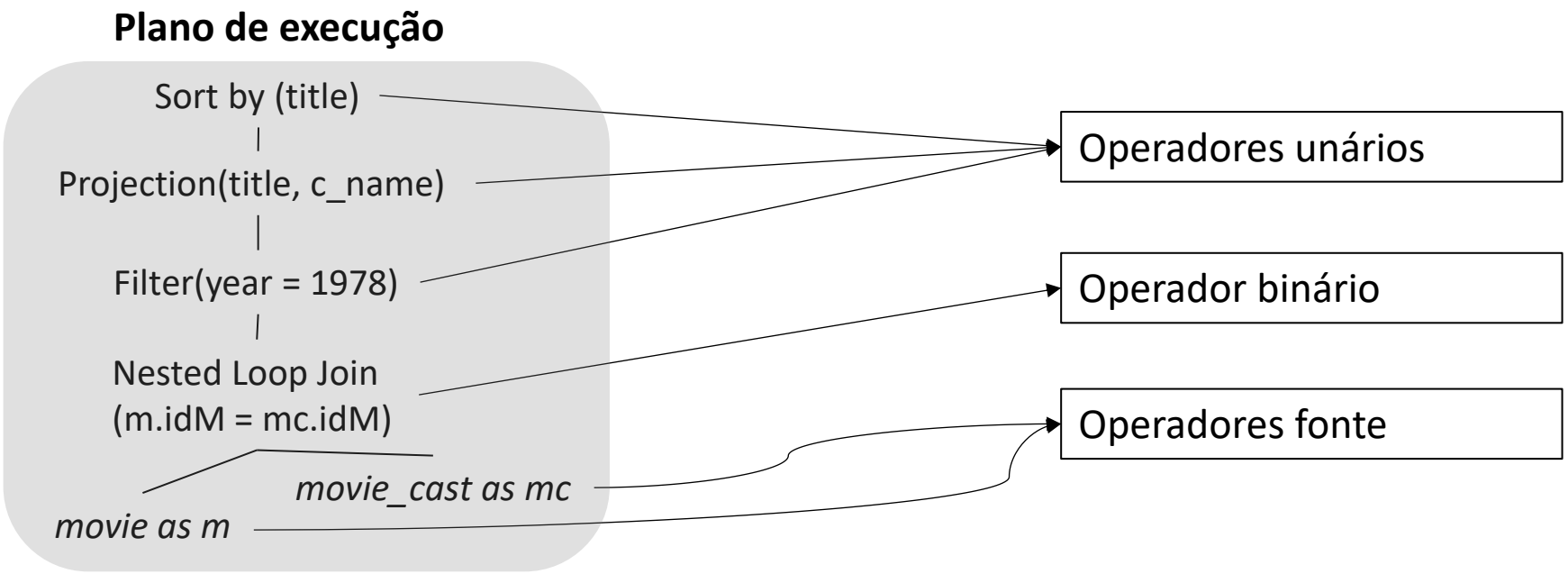


DBest

- Operadores típicos em um plano de execução
  - Projection: indica quais colunas devem ser retornadas
  - Filter (Selection): indica quais registros devem ser retornados
  - Join: indica que registros relacionados de duas tabelas devem ser combinados
  - Aggregation: indica que registros devem ser agrupados e valores sintetizados devem ser calculados
  - …

DBest

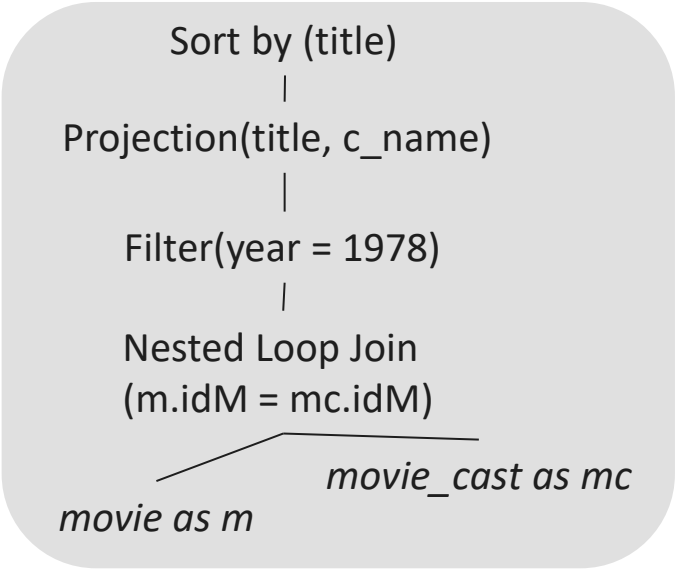
- O exemplo apresenta um plano de execução concreto usando operadores



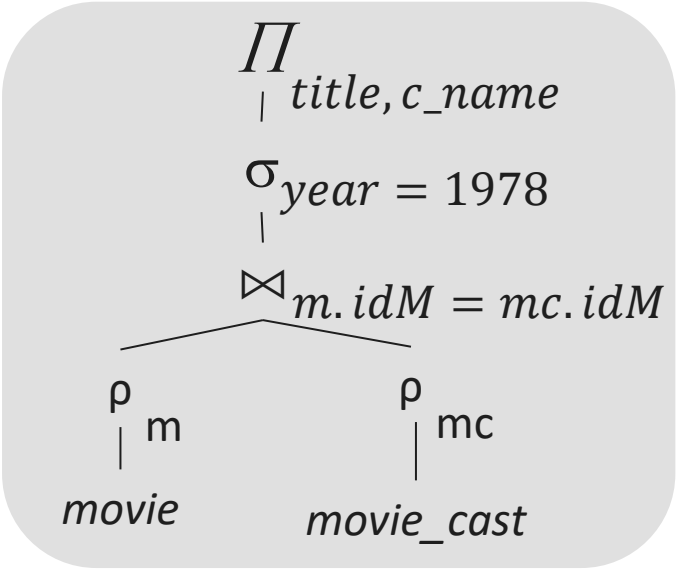
DBest

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

Plano de execução



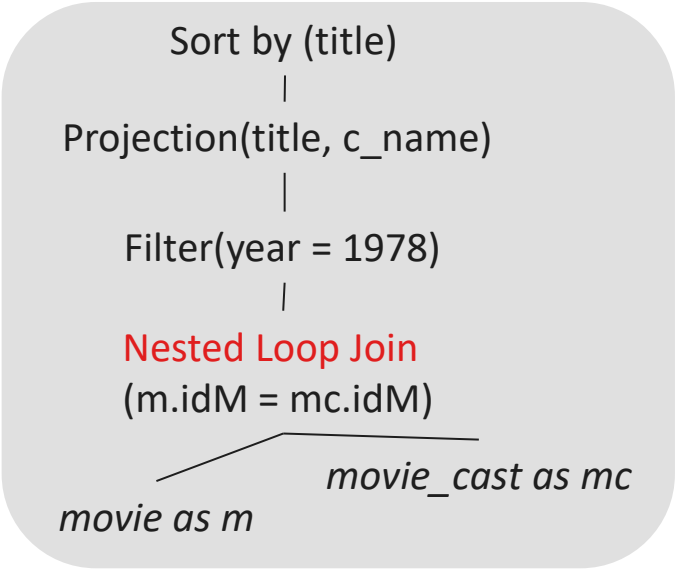
Expressão em álgebra relacional



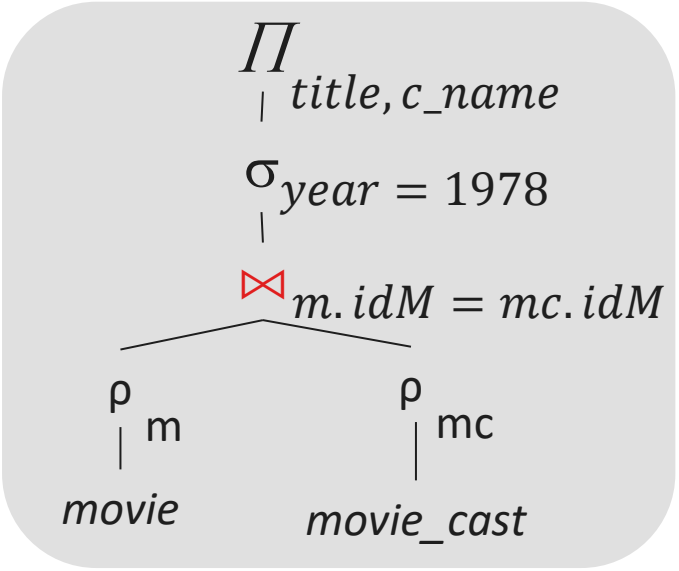
DBest

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

Plano de execução



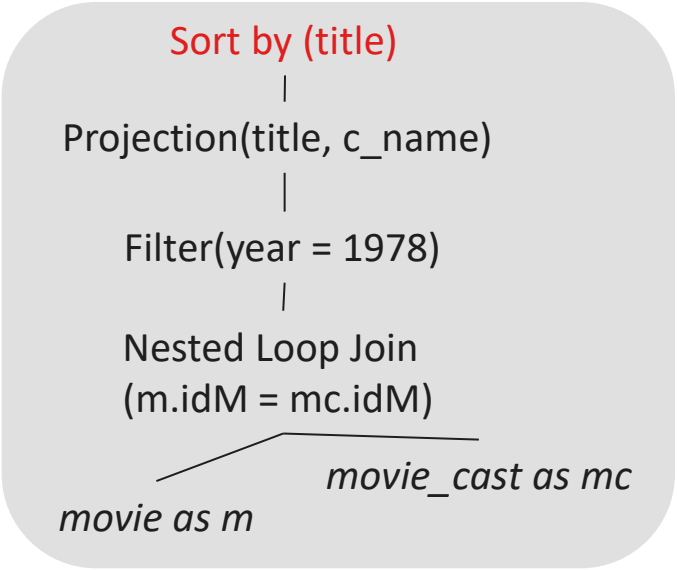
Expressão em álgebra relacional



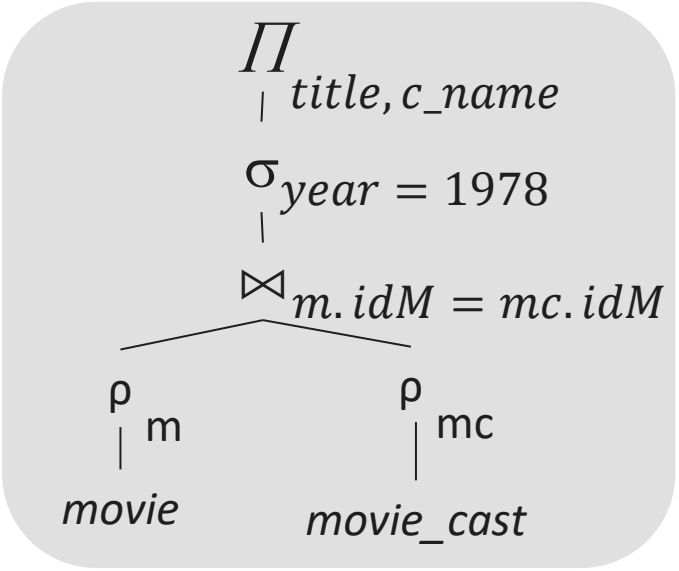
DBest

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

Plano de execução

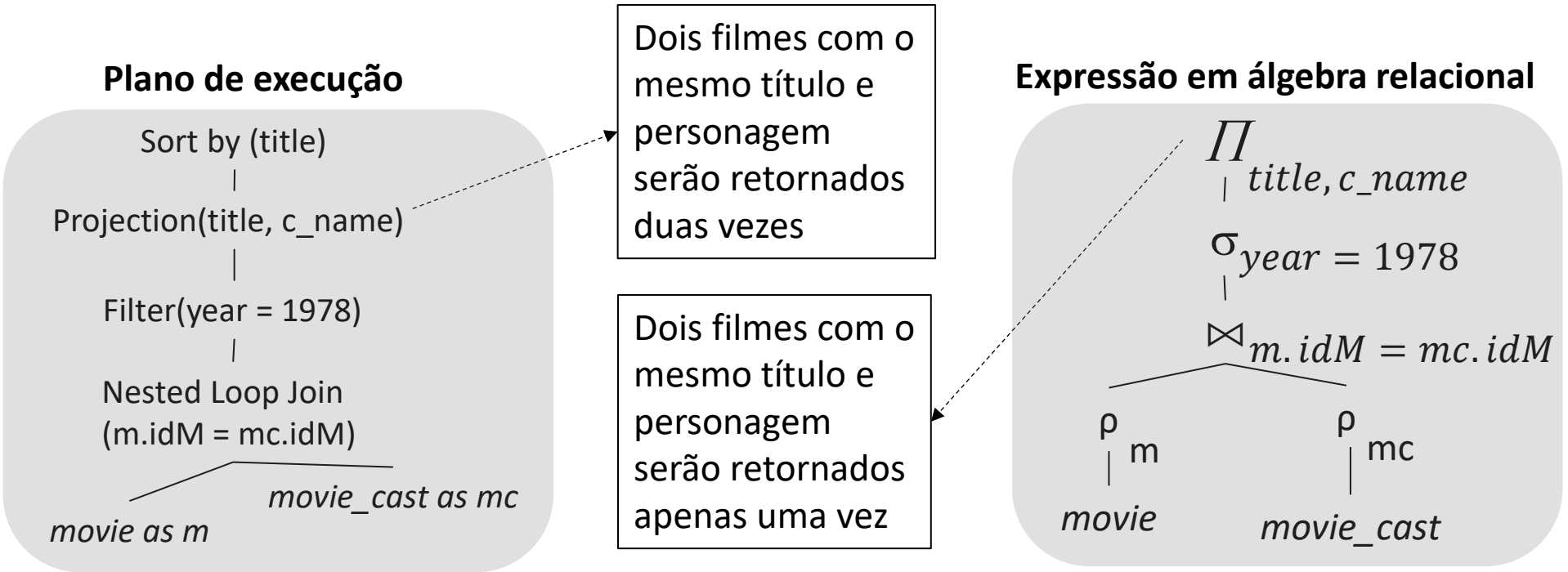


Expressão em álgebra relacional



DBest

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



DBest

- Nos planos de execução, os nós fonte são onde os dados estão armazenados
- A estrutura de armazenamento usada depende do SGBD usado
- Ex.
  - MySQL (versão InnoDB) usa árvores B+ para os registros e para os índices
  - PostgreSQL usa heap para os registros e árvores B+ para os índices

DBest

- No DBest, todos os dados são armazenados em árvores B+
  - Semelhante ao InnoDB, do MySQL
- Algumas árvores são as tabelas
  - onde os valores armazenados são os registros
- Outras árvores são índices
  - onde os valores armazenado são as chaves primárias dos registros

Tabelas: (árvores clusterizadas)
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

Índices (árvores não clusterizadas)
Idx_year(year, idM) -- índice para a coluna year de movie
Idx_order (order, idM, idP) -- índice para a coluna order de movie_cast

Os valores sublinhados são as chaves de busca de cada árvore

Sumário

- Introdução
- Planos de execução em SGBDs
- DBest
- Junções
  - Nested Loop
  - Hash Join
- Filtros
  - Sem uso de índice
  - Com uso de índice
  - Covering Index
  - Filtro disjuntivo
  - Filtro em junções

Junções

- Uma das principais operações realizadas por um plano de execução é a junção

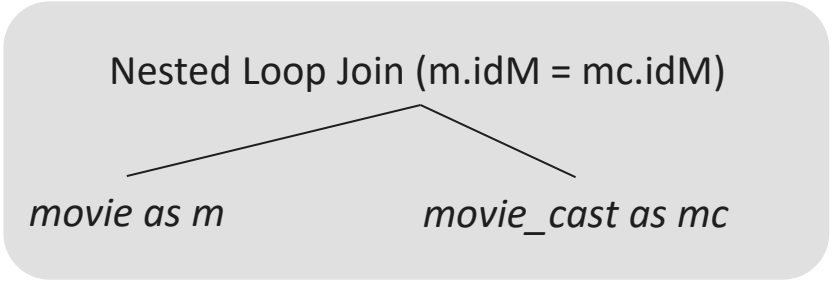
- Existem diversos algoritmos para junção
  - Os principais são
    - Nested Loop Join
    - Hash Join

Sumário

- Introdução
- Planos de execução em SGBDs
- Dbest
- Junções
  - Nested Loop
  - Hash Join
- Filtros
  - Sem uso de índice
  - Com uso de índice
  - Covering Index
  - Filtro disjuntivo
  - Filtro em junções

Nested Loop Join

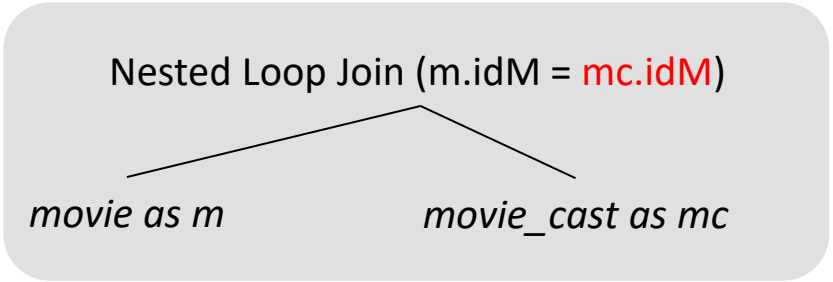
- Nested Loop Join é uma operação binária
  - Lado da esquerda: lado externo
  - Lado da direita: lado interno
- Funcionamento
  - Para cada registro do lado externo (movie)
    - Buscar as correspondências no lado interno (movie_cast)



Esquema:
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

Nested Loop Join

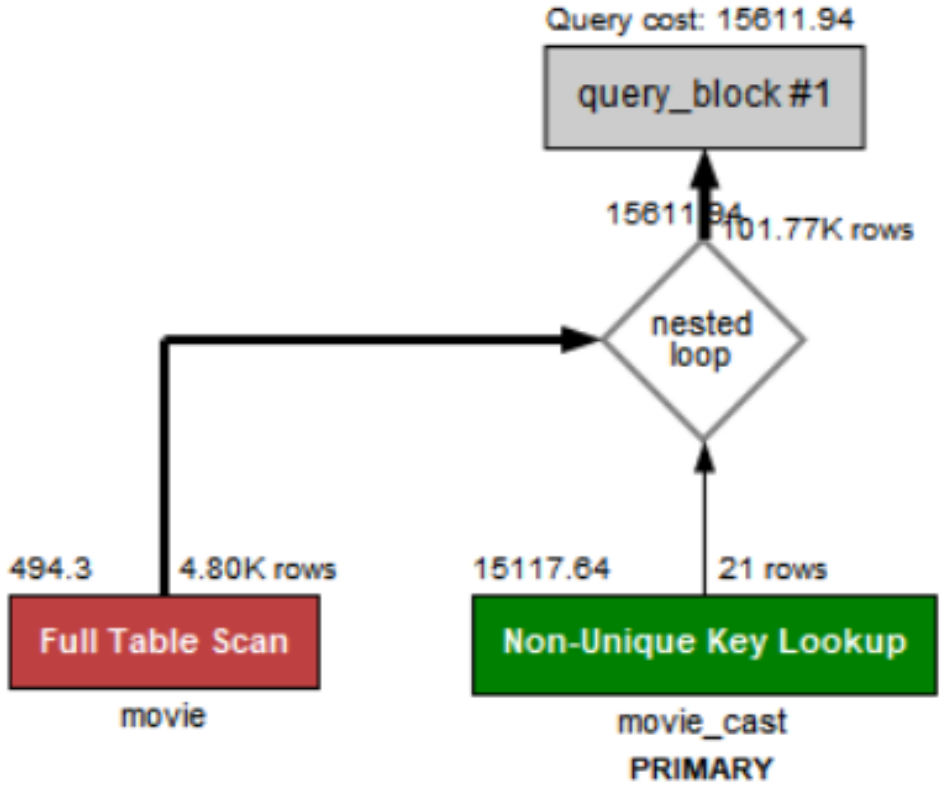
- No exemplo abaixo, a busca no lado interno é feito sobre a coluna mc.idM
- Essa coluna é um prefixo da chave de busca de movie_cast
  - Por isso, a busca é indexada (eficiente)



Esquema:
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

Nested Loop Join

- No MySQL
  - **Operador Full Table Scan**: todos os registros de movie são processados
  - **Operador Nested Loop**: O algoritmo de junção, que busca correspondências para cada movie
  - **Operador Key Lookup**: a busca é realizada por meio do acesso a um índice
    - **Unique key lookup**: apenas um registro pode ser retornado
    - **Non-unique key lookup**: mais de um registro pode ser retornado



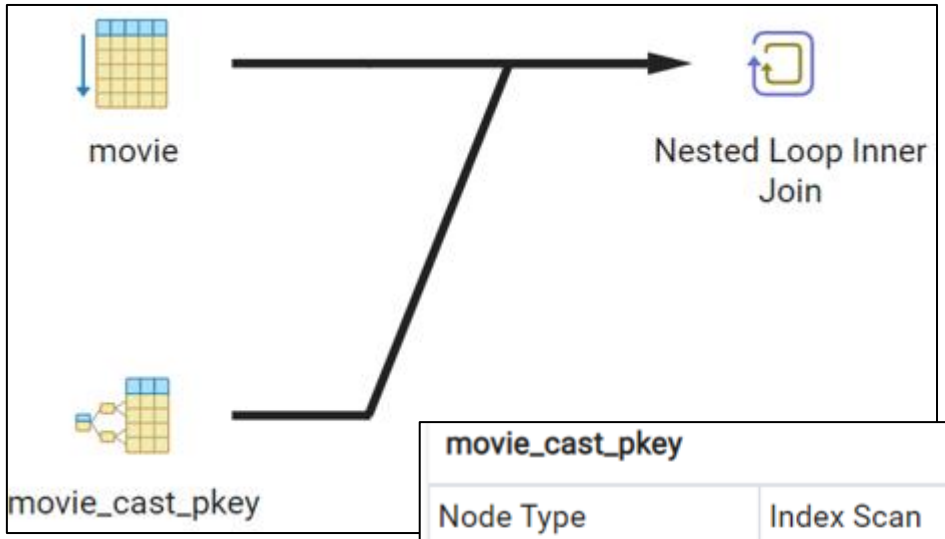
Nested Loop Join

- No PosgtreSQL

Operador Seq Scan: acessa todos os movies sequencialmente

Operador Nested Loop Inner Join: algoritmo de junção, que busca correspondências para cada movie

Operador Index Scan: a busca é realizada por meio do acesso a um índice



movie_cast_pkey

Node Type	Index Scan
Parent Relationship	Inner
Parallel Aware	
Async Capable	
Scan Direction	Forward
Index Name	movie_cast_pkey
Relation Name	movie_cast
Alias	movie_cast
Actual Rows	0
Actual Loops	0
Index Cond	(movie_id = movie.movie_id)
Rows Removed by Ind	0

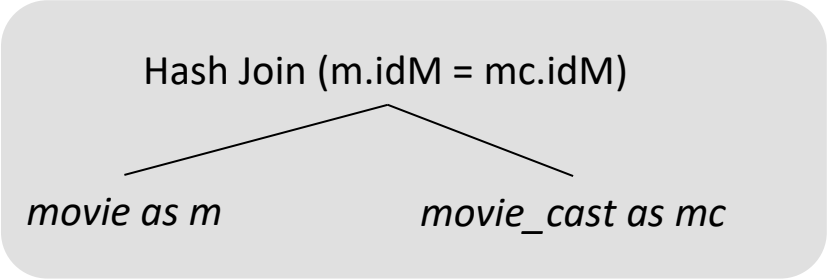
Sumário

- Introdução
- Planos de execução em SGBDs
- Dbest
- Junções
  - Nested Loop
  - Hash Join
- Filtros
  - Sem uso de índice
  - Com uso de índice
  - Covering Index
  - Filtro disjuntivo
  - Filtro em junções

Hash Join

- Hash Join é uma operação binária
  - Lado da esquerda: lado externo
  - Lado da direita: lado interno

- Funcionamento
  - Constrói uma tabela hash temporária sobre o lado interno, indexada pelo atributo de junção
    - Ou seja, é uma operação materializada (consome memória)
  - Para cada registro do lado externo
    - Busca as correspondências dentro da tabela Hash criada



Hash Join

- Uma das principais utilidades do Hash Join é em situações em que não existe índice sobre a coluna alvo de uma junção

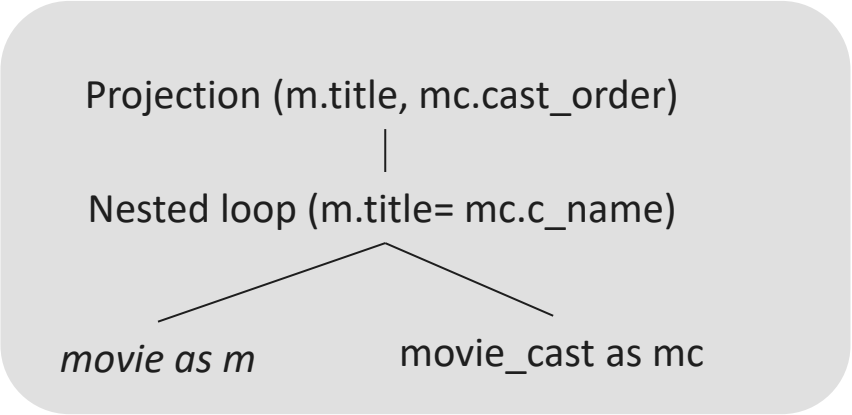
- Por exemplo, a consulta abaixo busca por filmes cujo título seja igual ao nome de algum personagem de filme

```
SELECT m.title, mc.c_name
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

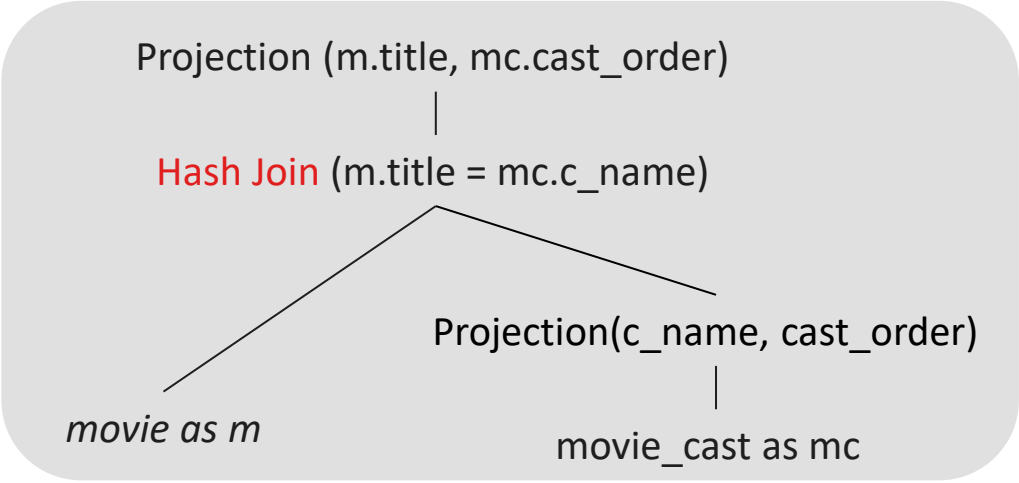
- O plano abaixo usa Nested Loop Join
- Como não existe índice a acessar
  - Para cada filme, é necessário varrer toda a tabela de movie_cast para encontrar correspondências



Esquema:
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

Hash Join

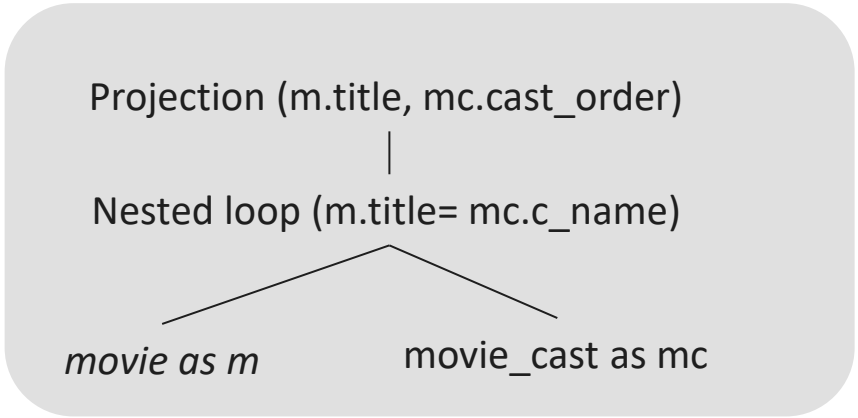
- O plano abaixo usa Hash Join
- O algoritmo
  - constrói uma tabela hash indexada por c_name
    - A tabela inclui
      - todos os registros de movie_cast
      - Mas apenas as colunas de interesse (cast_name, cast_order)
  - Realiza as buscas sobre essa tabela

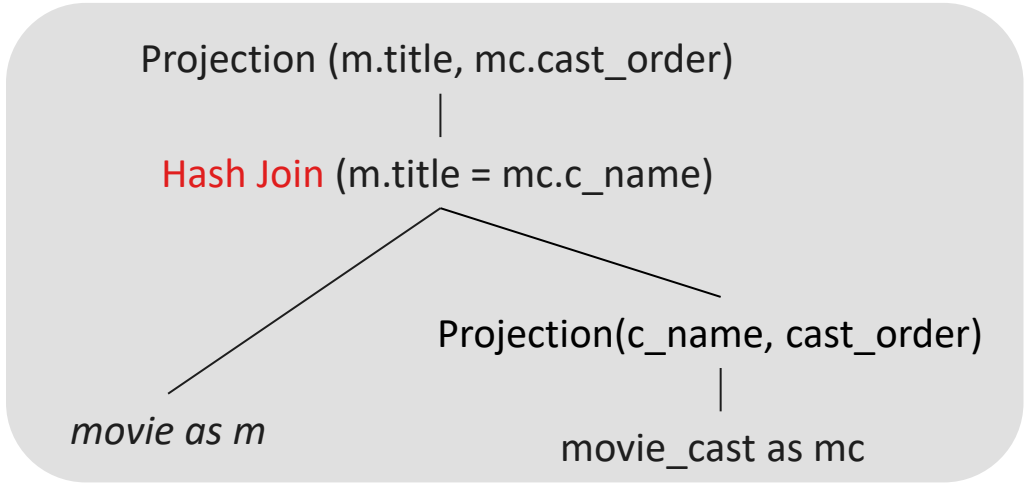


Esquema:
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

Hash Join

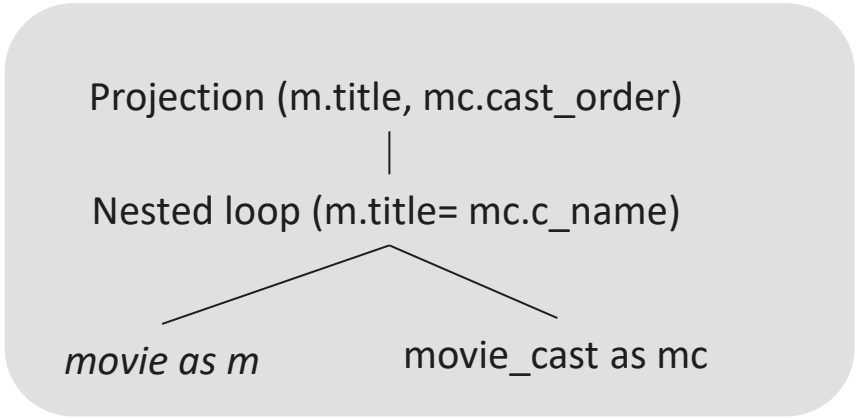
- Qual é melhor?

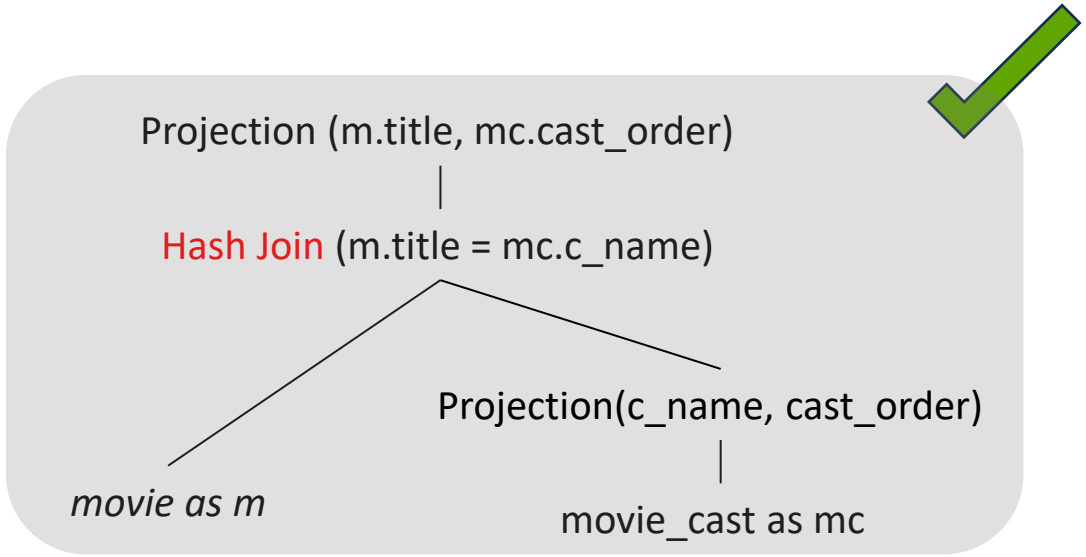




Hash Join

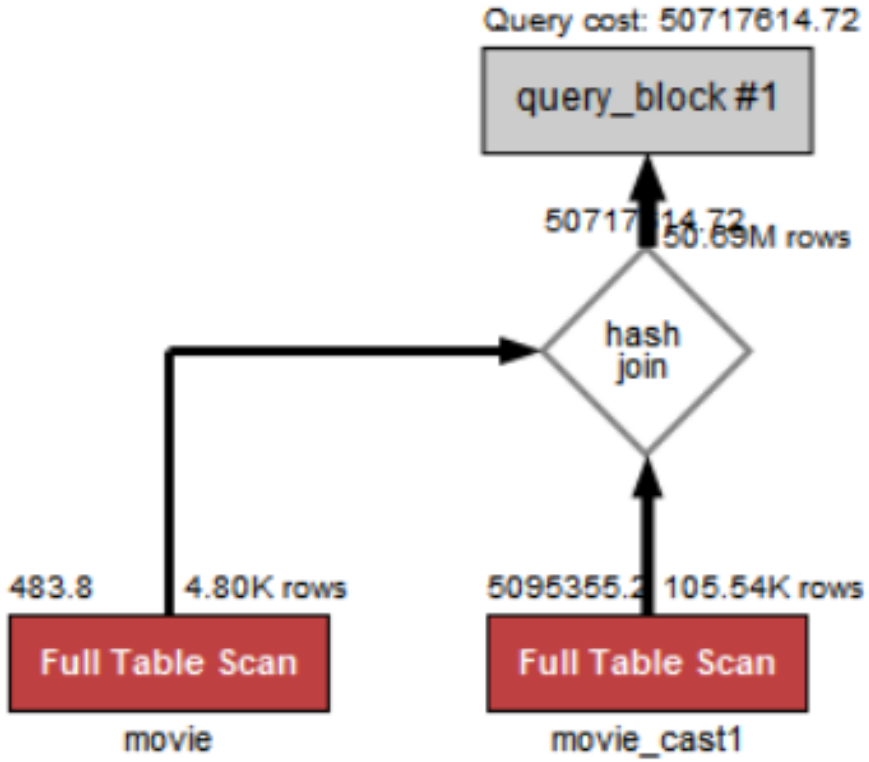
- O Hash Join é mais eficiente
  - Faz o table scan uma única vez, e não uma vez para cada movie
- Desvantagem
  - precisa carregar todos os registros de movie_cast para a memória





Hash Join

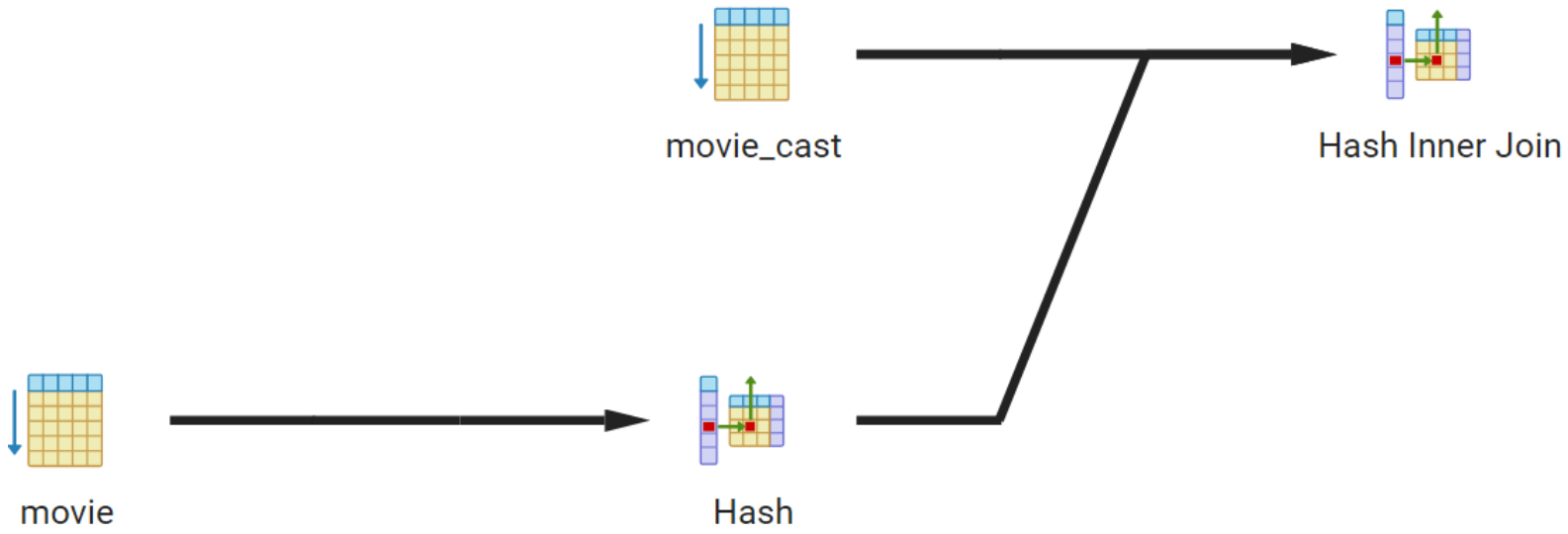
- Em MySQL
  - O losango mostra que o plano usa Hash Join
  - Dois table scan são necessários
    - Um para percorrer os filmes
    - Outro para alimentar a tabela hash usada pelo Hash Join



Hash Join

- Em PostgreSQL
  - O operador Hash Inner Join indica o uso do Hash Join
  - O operador Hash cria a tabela Hash que é usada pelo algoritmo de junção

  - Dois table scan são necessários
    - Um para percorrer os filmes
    - Outro para alimentar a tabela hash



Nested Loop Join vs Hash Join

- Quando cada algoritmo de junção é usado

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

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

Sumário

- Introdução
- Planos de execução em SGBDs
- Dbest
- Junções
  - Nested Loop
  - Hash Join
- Filtros
  - Sem uso de índice
  - Com uso de índice
  - Covering Index
  - Filtro disjuntivo
  - Filtro em junções

Índices e Filtros

- Como vimos em outra aula, índices podem ser (e geralmente são) usados para resolver filtros em consultas

- Os próximos slides mostram exemplos onde índices são usados, destacando os operadores usados nos planos de execução

Sumário

- Introdução
- Planos de execução em SGBDs
- DBest
- Junções
  - Nested Loop
  - Hash Join
- Filtros
  - Sem uso de índice
  - Com uso de índice
  - Covering Index
  - Filtro disjuntivo
  - Filtro em junções

Filtros sem uso de índice

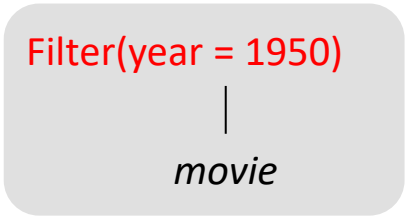
- Exemplo: retornar filmes cujo ano seja 1950

```
SELECT *
FROM movie
WHERE year = 1950
```

- Primeiramente, vamos supor que não exista índice que possa ser usado

Filtros sem uso de índice

- O plano abaixo executa o filtro diretamente sobre a tabela movie



Plano B

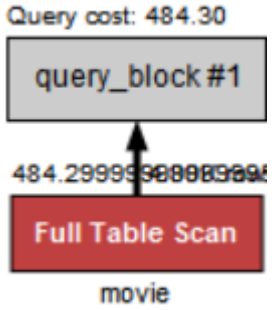
tabelas:
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

Filtros sem uso de índice

```
SELECT * FROM movie2
WHERE release_year = 1950;
```

Em MySQL, o Full Table Scan indica que a tabela movie é varrida.

O plano escrito mostra que o filtro é aplicado sobre cada registro lido



MySQL

-> Filter: (movie.release_year = 1950)

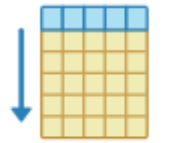
    -> Table scan on movie

Filtros sem uso de índice

```
SELECT * FROM movie2
WHERE release_year = 1950;
```

Em PostgreSQL, o plano visual indica que foi feita uma leitura sequencial.

O plano escrito mostra a aplicação do filtro sobre os registros lidos



movie

PostgreSQL

Seq Scan on movie

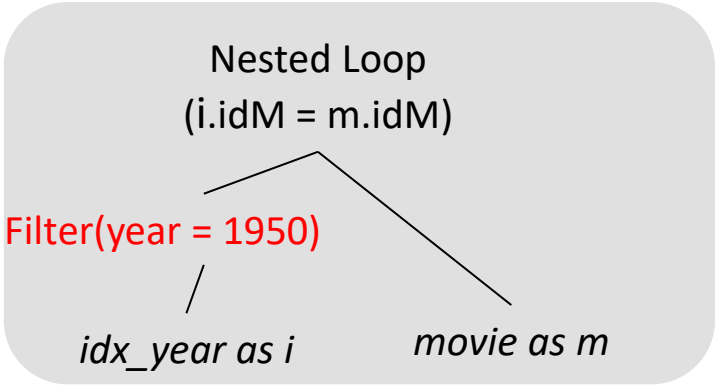
  Filter: (release_year = 1950)

Sumário

- Introdução
- Planos de execução em SGBDs
- DBest
- Junções
  - Nested Loop
  - Hash Join
- Filtros
  - Sem uso de índice
  - Com uso de índice
  - Covering Index
  - Filtro disjuntivo
  - Filtro em junções

Filtros com uso de índice

- Neste outro plano, o filtro acessa um índice sobre a coluna filtrada year



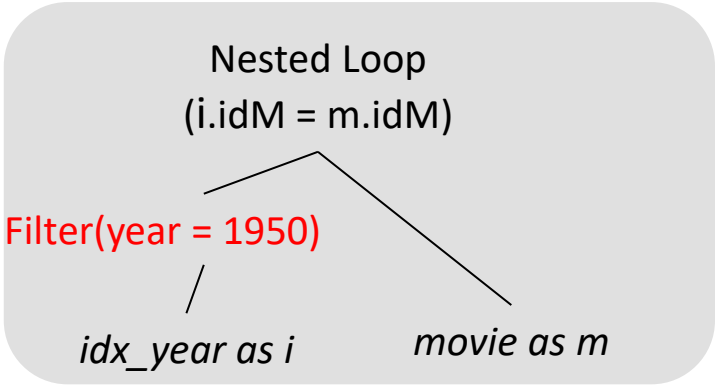
Plano B

tabelas:
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

Índices:
Idx_year(year, idM)

Filtros com uso de índice

- No DBest, para cada entrada do índice que satisfizer o filtro, é necessário fazer a complementação (buscar o registro completo de movie)
  - A complementação é feita por meio de um algoritmo de junção (ex. Nested Loop)



Plano B

tabelas:
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

Índices:
Idx_year(year, idM)

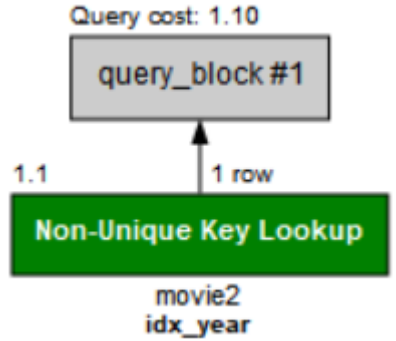
Filtros com uso de índice

```
SELECT * FROM movie2
WHERE release_year = 1950;
```

Em MySQL, o plano só indica que o índice é usado (idx_year) em uma operação de busca (index lookup)

No entanto, a etapa de complementação não aparece.

MySQL
-> Index lookup on movie2 using idx_year (release_year=1950)



Filtros com uso de índice

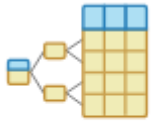
```
SELECT * FROM movie2
WHERE release_year = 1950;
```

O mesmo ocorre com o PostgreSQL

O plano visual indica que um índice está sendo usado para realizar uma busca (index scan)

PostgreSQL

Index Scan using idx_year on movie2
  Index Cond: (release_year = 1950)



idx_year

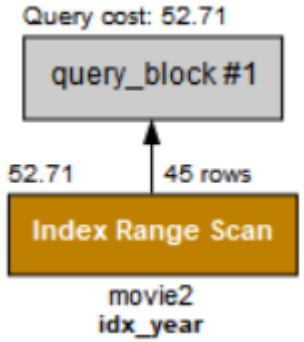
Filtros com uso de índice

```
SELECT * FROM movie2
WHERE release_year < 1950;
```

Com filtro por intervalo (year < 1950), o MySQL usa index range scan em vez de um key lookup

MySQL

-> Index range scan on movie2 using idx_year over (NULL < release_year < 1950),
with index condition: (movie2.release_year < 1950)



Filtros com uso de índice

```
SELECT * FROM movie2
WHERE release_year < 1950;
```

O PostgreSQL usa uma técnica baseada em bitmaps (bitmap heap scan)

Somente para as entradas relevantes do bitmap, os registros da tabela são lidos

Como os bitmaps podem trazer falsos positivos, é necessário verificar se os registros encontrados são efetivamente válidos (recheck cond)

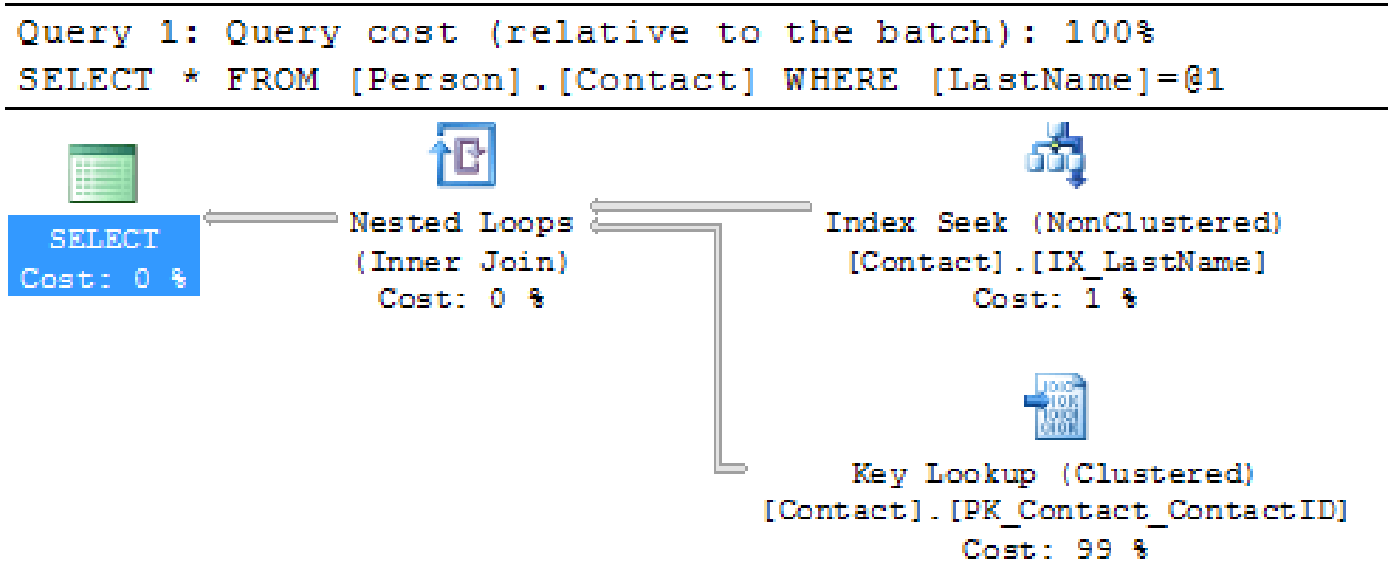
PostgreSQL

Bitmap Heap Scan on movie2
  Recheck Cond: (release_year < 1950)
  -> Bitmap Index Scan on idx_year
        Index Cond: (release_year < 1950)

Filtros com uso de índice

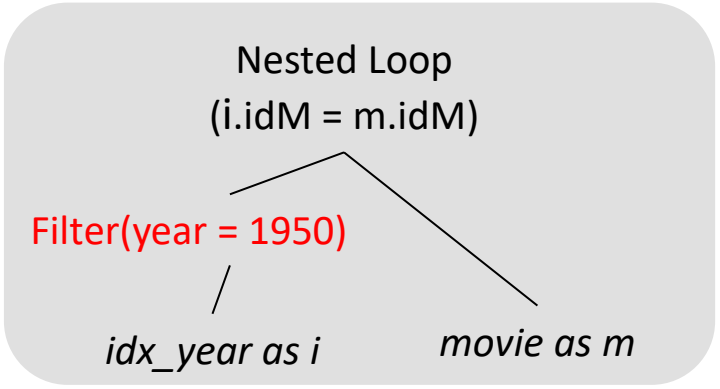
- Os exemplos anteriores mostraram que o nem MySQL nem PostgreSQL exibem a complementação quando um índice é usado

- Já o SQL Server transparece essa informação
  - Assim como no DBest, o Nested Loop Join é usado para fazer a complementação

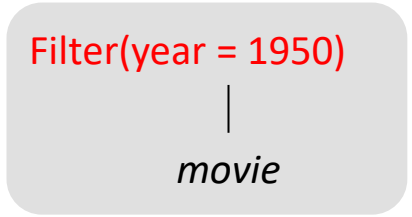


Filtros com uso de índice

- Qual plano é melhor?



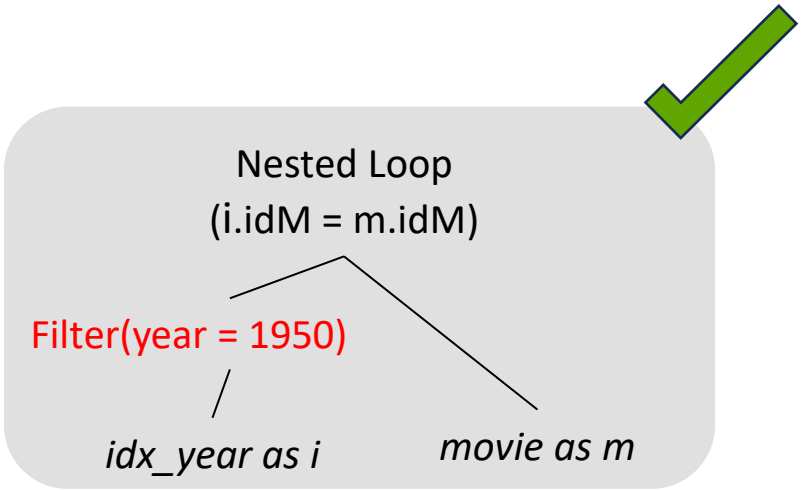
Plano A



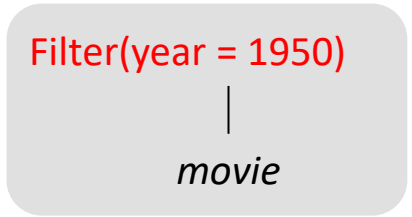
Plano B

Filtros com uso de índice

- O plano A é melhor
  - O filtro é seletivo
  - O índice leva à poucas entradas relevantes
  - Somente para elas deverá ser feita a complementação



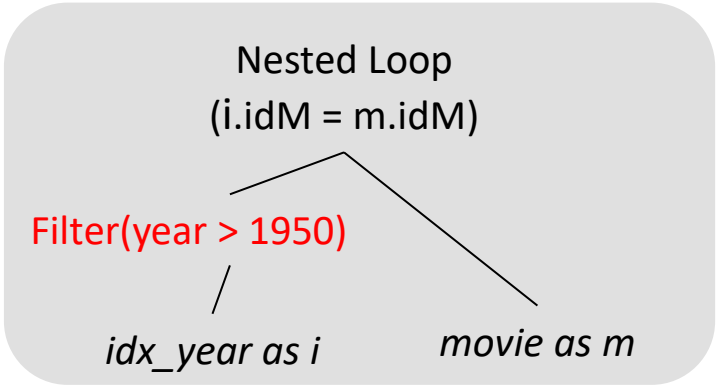
Plano A



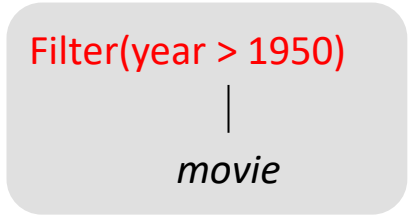
Plano B

Filtros com uso de índice

- Agora o filtro é pouco seletivo
- Qual é melhor?



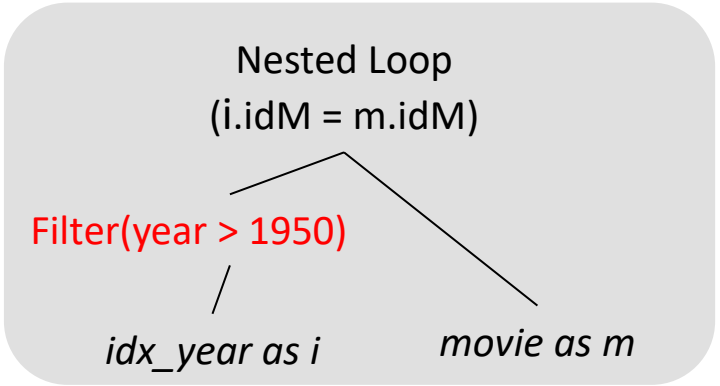
Plano A



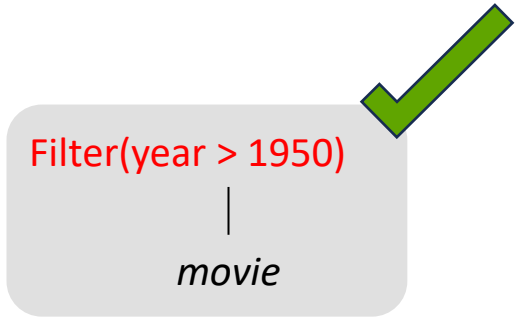
Plano B

Filtros com uso de índice

- O plano B é melhor
  - O filtro é pouco seletivo (a maioria dos filmes são posteriores a 1950)
  - Por isso, é melhor ir direto para a tabela



Plano A



Plano B

Sumário

- Introdução
- Planos de execução em SGBDs
- DBest
- Junções
  - Nested Loop
  - Hash Join
- Filtros
  - Sem uso de índice
  - Com uso de índice
  - Covering Index
  - Filtro disjuntivo
  - Filtro em junções

Covering Index

- Exemplo: retornar anos de filmes superiores a 1950

```
SELECT year
FROM movie
WHERE year > 1950
```

tabelas:
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

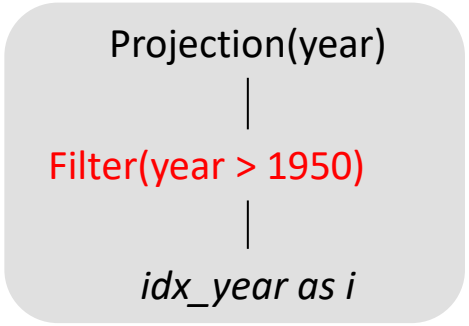
Índices:
Idx_year(year, idM)

- O filtro é sobre uma coluna indexada

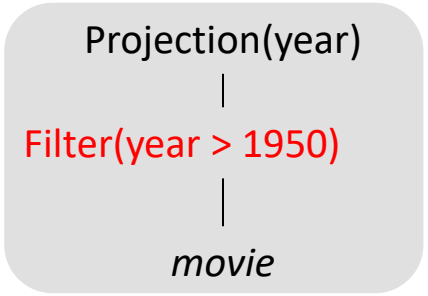
- Apenas a coluna ano deve ser retornada

Covering Index

- No DBest, o operador Projection deixa claro quais colunas devem ser retornadas
  - Sem esse operador, todas as colunas acessíveis são retornadas



Plano A



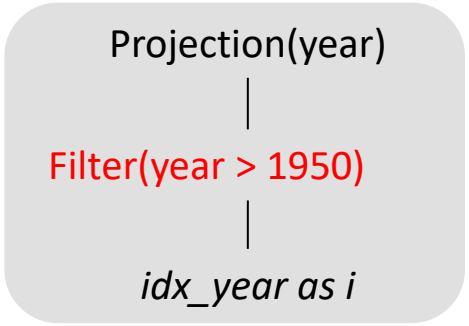
Plano B

tabelas:
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

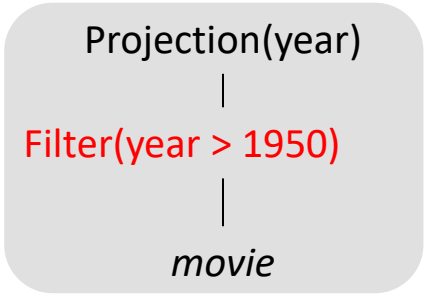
Índices:
Idx_year(year, idM)

Covering Index

- Qual dos dois planos é melhor?



Plano A



Plano B

tabelas:
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

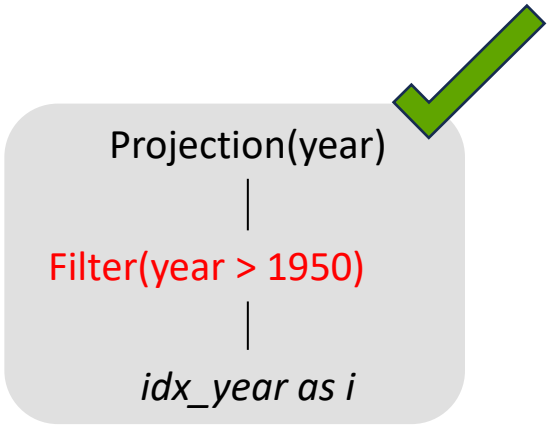
Índices:
Idx_year(year, idM)

Covering Index

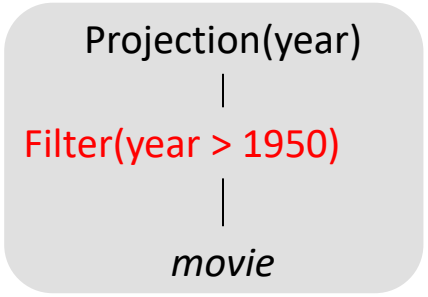
- O plano A é melhor
  - A estrutura de dados usada é menor (o índice é menor do que a tabela)

- Observe que não foi necessária nenhuma complementação
  - O índice possuía todas as colunas necessárias para resolver a consulta

- Para esses casos, o índice utilizado pode ser chamado de covering index



Plano A



Plano B

tabelas:
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

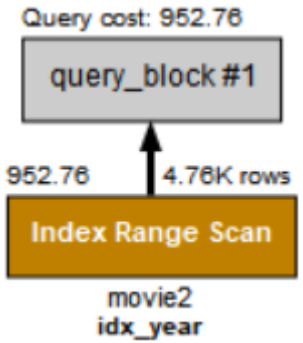
Índices:
Idx_year(year, idM)

Covering Index

SELECT **release_year** FROM **movie2**
WHERE **release_year > 1950;**

Em MySQL, o operador index range scan é usado quando o filtro é por intevalo, e não por igualdade

O plano textual indica que se trata de um covering index



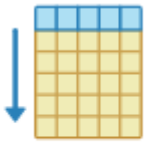
MySQL
-> Filter: (movie2.release_year > 1950)
    -> Covering index range scan on movie2 using idx_year
        over (1950 < release_year)

Covering Index

```
SELECT release_year FROM movie2
WHERE release_year > 1950;
```

O PostgreSQL julgou que seria melhor não usar o índice

PostgreSQL
Seq Scan on movie2
  Filter: (release_year > 1950)



movie

Covering Index

```
SET enable_seqscan TO off;
SELECT release_year FROM movie2
WHERE release_year > 1950;
```

Podemos encorajar que o índice seja usado sugerindo que o PostgreSQL não recorra à busca sequencial.

Obs.: trata-se apenas de uma sugestão. Não se pode obrigá-lo a tomar um caminho

PostgreSQL

```
Bitmap Heap Scan on movie2
  Recheck Cond: (release_year > 1950)
  ->  Bitmap Index Scan on idx_year
        Index Cond: (release_year > 1950)
```

Sumário

- Introdução
- Planos de execução em SGBDs
- DBest
- Junções
  - Nested Loop
  - Hash Join
- Filtros
  - Sem uso de índice
  - Com uso de índice
  - Covering Index
  - Filtro disjuntivo
  - Filtro em junções

Índices e filtros disjuntivos

- Exemplo: retornar filmes de 1950 ou que tenham duração maior do que 120 minutos

```
SELECT *
FROM movie
WHERE year = 1950 OR duration > 120
```

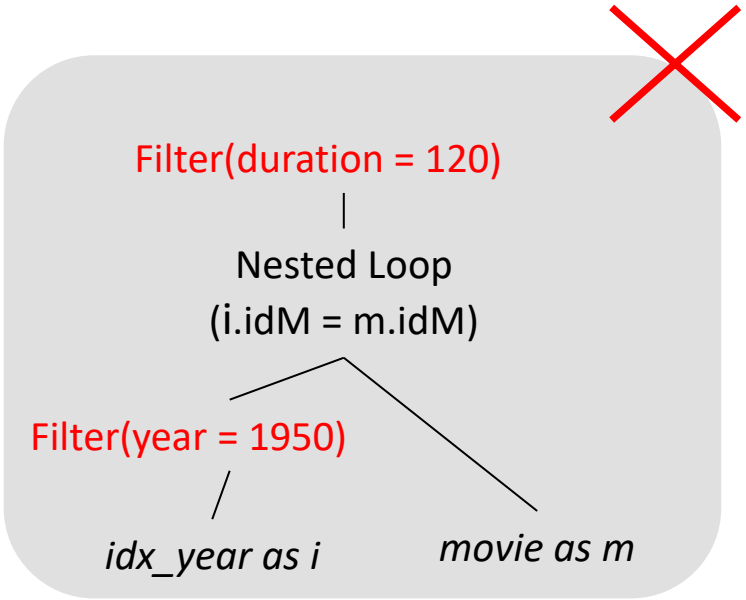
tabelas:
movie(idM, title, year)
person(idP, name)
movie_cast(idM, idP, c_name, order)

Índices:
Idx_year(year, idM)
Idx_dur(duration, idM)

- O filtro é disjuntivo
- Existem índices sobre as duas colunas filtradas

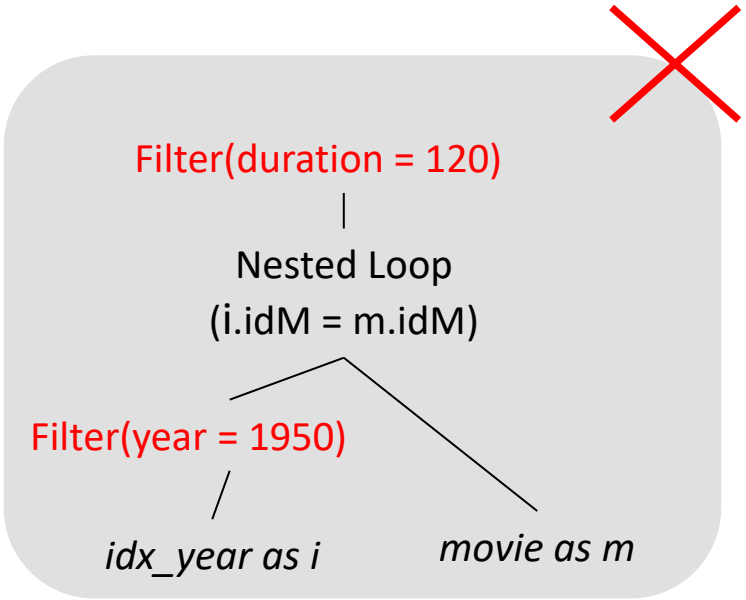
Índices e filtros disjuntivos

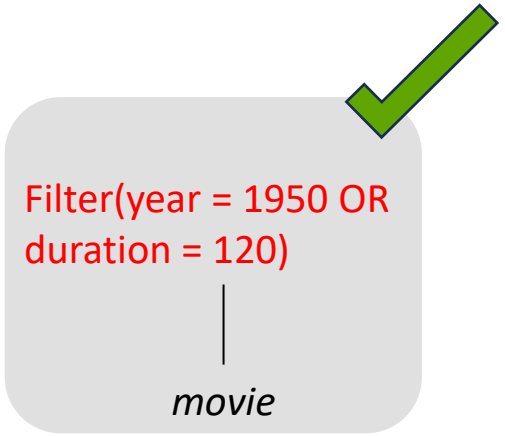
- O plano B abaixo é inválido
  - Ao executar o filtro sobre year, apenas registos que satisfizerem esse filtro são retornados
  - E quanto aos filmes com duração = 120 que não sejam desse ano?



Índices e filtros disjuntivos

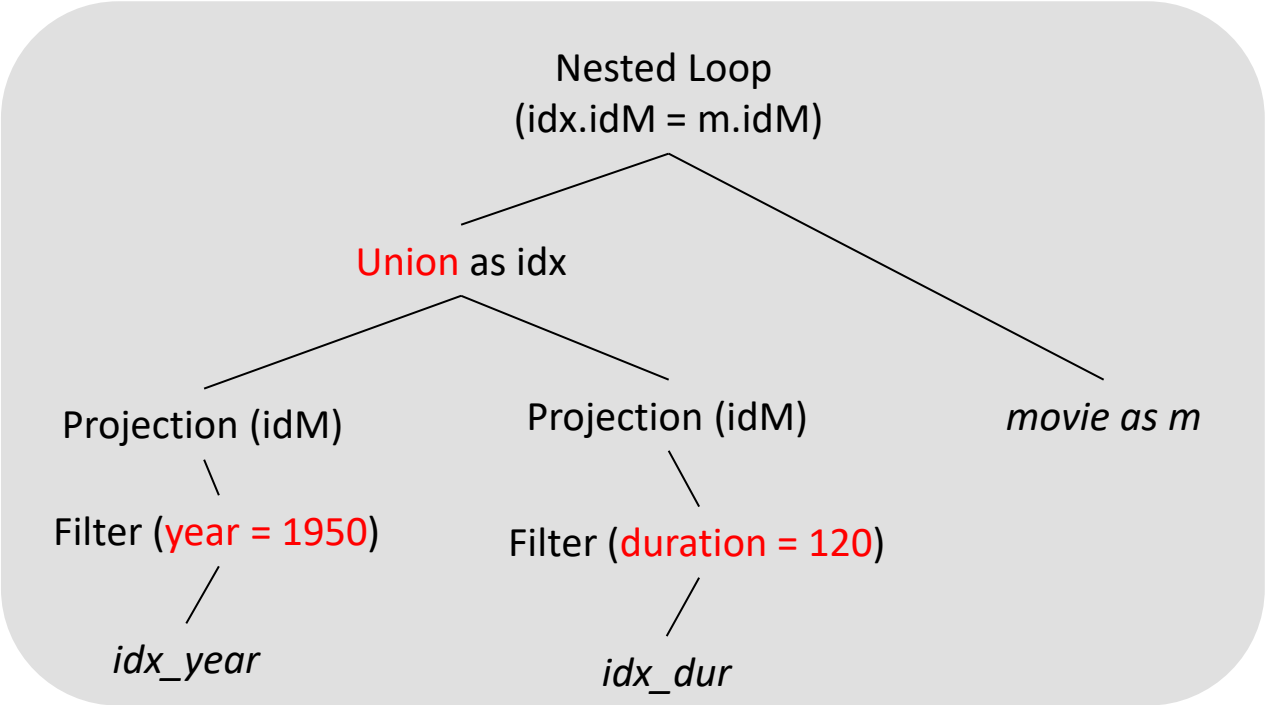
- Para evitar inconsistências, os dois filtros devem ser mantidos juntos
  - Com isso, o índice sobre ano não pode ser usado





Índices e filtros disjuntivos

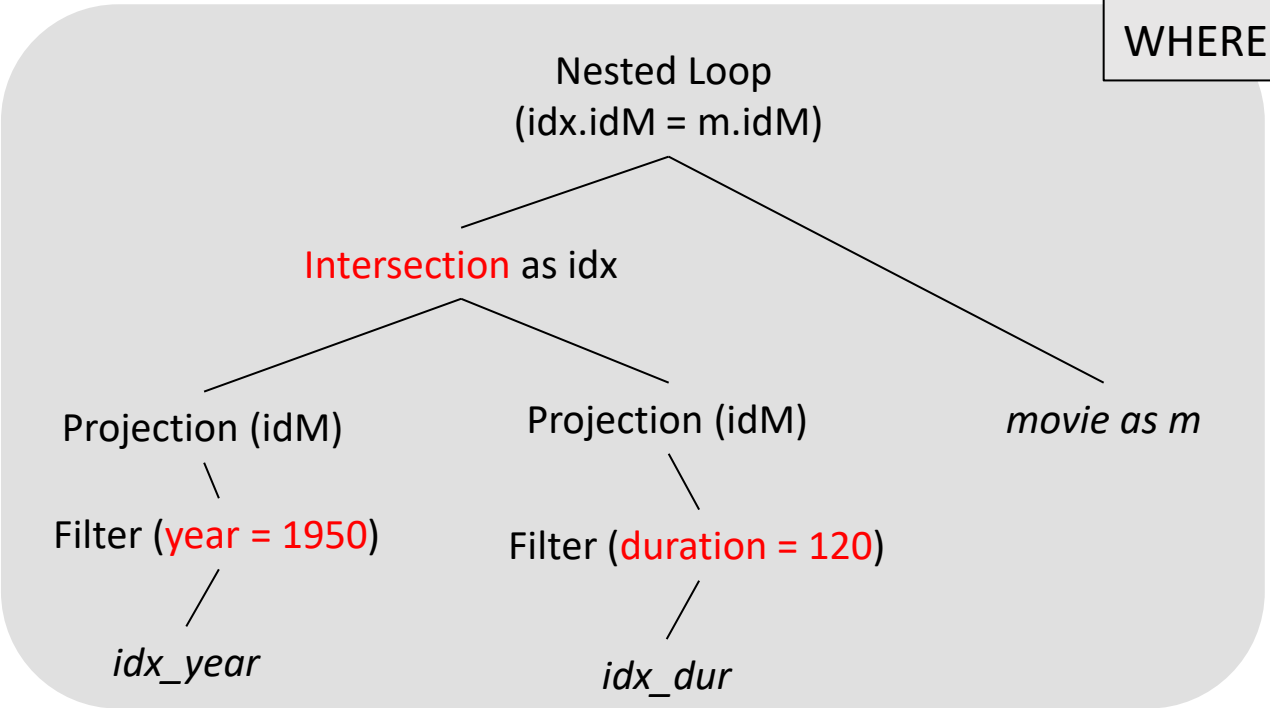
- Outra possibilidade é usar a técnica de união de ponteiros
  - Caso as duas colunas filtradas sejam indexadas



Índices e filtros disjuntivos

- Caso fosse usado AND, daria para usar uma técnica similar, baseada em interseção de ponteiros

```
SELECT *
FROM movie
WHERE year = 1950 AND duration > 120
```



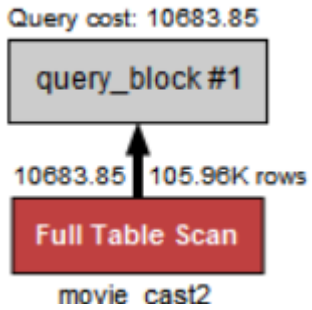
Índices e filtros disjuntivos

SELECT * FROM **movie_cast2**
WHERE **cast_order > 0** OR **movie_id > 0;**

Apesar de existirem dois índices que podem ser usados, a presença de filtros poucos seletivos levou o MySQL a usar um Full Table Scan

MySQL

-> Filter: ((movie_cast2.cast_order > 0) or (movie_cast2.movie_id > 0))
    -> Table scan on movie_cast2



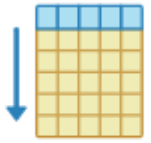
Índices e filtros disjuntivos

```
SELECT * FROM movie_cast2
WHERE cast_order > 0 OR movie_id > 0;
```

O PostgreSQL seguiu o mesmo caminho

PostgreSQL

```
Seq Scan on movie_cast2
  Filter: ((cast_order > 0) OR (movie_id > 0))
```



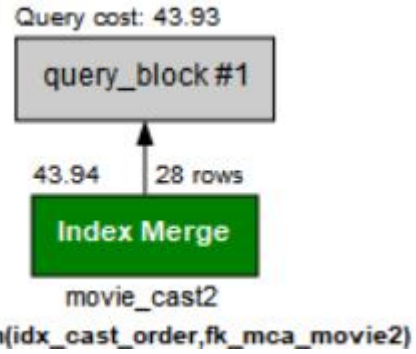
movie_cast2

Índices e filtros disjuntivos

```
SELECT *
FROM movie_cast2
WHERE cast_order > 220 OR movie_id = 5;
```

Com filtros mais seletivos, o MySQL passa a usar a técnica de união de ponteiros
- Foi usado index range scan para cada filtro
- E um sort-deduplicate para a união de ponteiros

Curiosamente, é aplicado um filtro final
- Isso sugere que a técnica sort-deduplicate gera falso-positivos



MySQL

-> Filter: ((movie_cast2.cast_order > 220) or (movie_cast2.movie_id = 5))
  -> Sort-deduplicate by row ID
    -> Index range scan on movie_cast2 using idx_cast_order over (220 < cast_order)
    -> Index range scan on movie_cast2 using fk_mca_movie2 over (movie_id = 5)

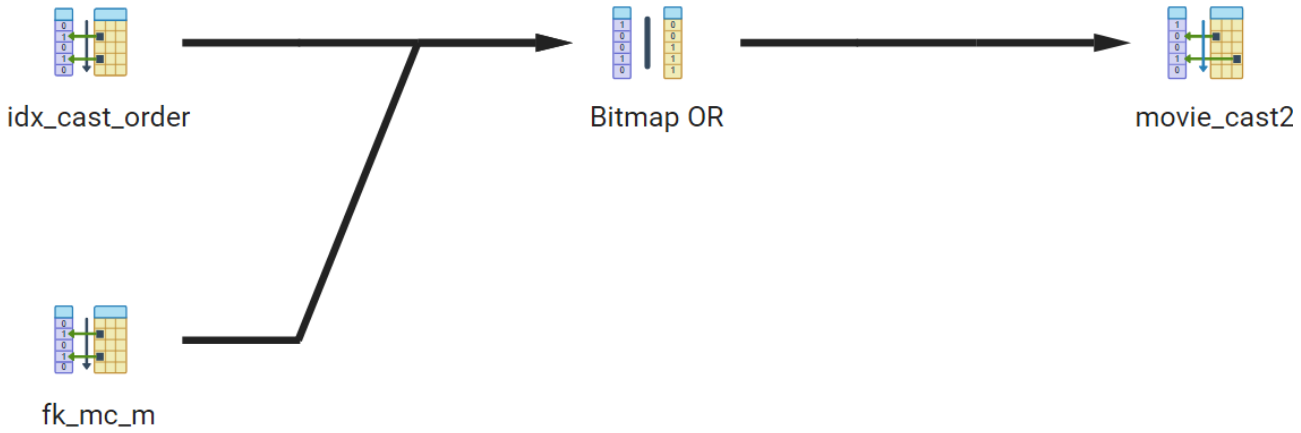
Índices e filtros disjuntivos

```
SELECT *
FROM movie_cast2
WHERE cast_order > 220 OR movie_id = 5;
```

Neste exemplo, o PostgreSQL também usou a técnica de união de ponteiros
- É usado o bitmap index scan para cada um dos índices
- E um BitmapOr para a união dos ponteiros

PostgreSQL

```
Bitmap Heap Scan on movie_cast2
  Recheck Cond: ((cast_order > 220) OR (movie_id = 5))
  ->  BitmapOr
      ->  Bitmap Index Scan on idx_cast_order
            Index Cond: (cast_order > 220)
      ->  Bitmap Index Scan on fk_mc_m
            Index Cond: (movie_id = 5)"(movie_id = 5)
```

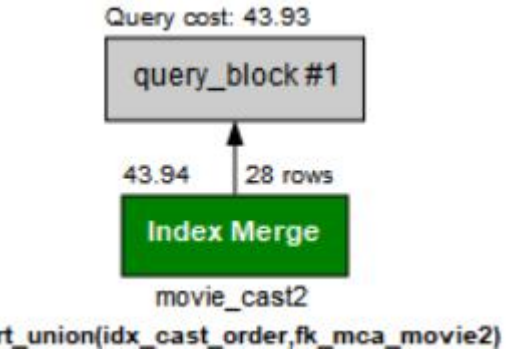


Índices e filtros disjuntivos

```
SELECT * /*+ index_merge(mc)*/
FROM movie_cast2
WHERE cast_order > 220 OR movie_id > 5;
```

Com filtros menos seletivos (ex. Movie_id > 5), é mais provável que seja usado um table scan

Nesses casos, um index hint encoraja o MySQL a usar a técnica de união de ponteiros



MySQL

-> Filter: ((movie_cast2.cast_order > 220) or (movie_cast2.movie_id = 5))
  -> Sort-deduplicate by row ID
    -> Index range scan on movie_cast2 using idx_cast_order over (220 < cast_order)
    -> Index range scan on movie_cast2 using fk_mca_movie2 over (movie_id = 5)

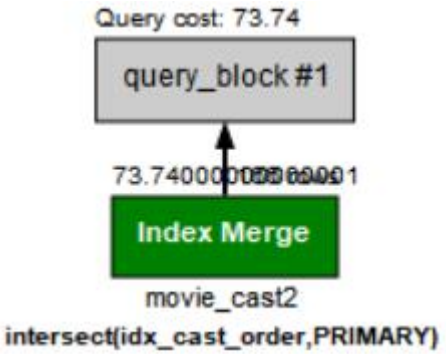
Índices e filtros disjuntivos

SELECT * FROM **movie_cast**
WHERE **cast_order = 50** AND **movie_id > 100**

Este exemplo mostra que o MySQL suporta a técnica baseada em interseção de ponteiros, para consultas com filtros conjuntivos (AND)

MySQL

-> Filter: ((movie_cast2.cast_order = 50) and (movie_cast2.movie_id > 100))
    -> Intersect rows sorted by row ID
        -> Index range scan on movie_cast2 using idx_cast_order
            over (cast_order = 50 AND 100 < movie_id)



Sumário

- Introdução
- Planos de execução em SGBDs
- DBest
- Junções
  - Nested Loop
  - Hash Join
- Filtros
  - Sem uso de índice
  - Com uso de índice
  - Covering Index
  - Filtro disjuntivo
  - Filtro em junções

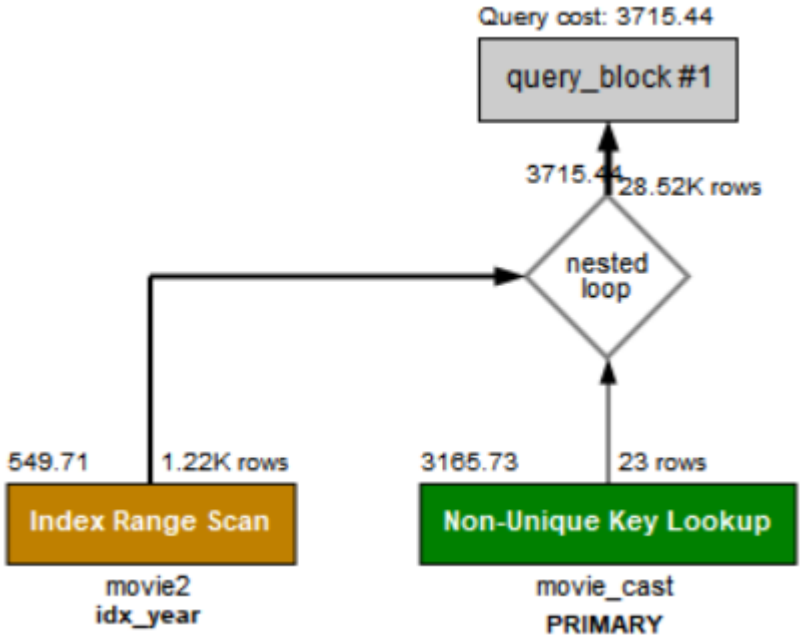
Filtros em junções

- Consultas complexas provavelmente terão vários filtros e junções
- Nesses casos, os filtros podem ser usados de duas maneiras
  - para reduzir a quantidade de registros a processar
  - Para reduzir o consumo de memória
- Quem define o caminho a seguir é o SGBD

Filtros em junções

SELECT * FROM **movie2** JOIN **movie_cast** USING(**movie_id**)
WHERE **release_year > 2010;**

MySQL escolheu usar o índice para reduzir a quantidade de registros a processar



Filtros em junções

```
SELECT * FROM movie2 JOIN movie_cast USING(movie_id)
WHERE release_year > 2010;
```

Já o Postgres escolheu usar o índice para reduzir o tamanho da tabela hash utilizada

Importante lembrar

- Os filtros reduzem a quantidade de registros que devem ser processados

- Geralmente os filtros mais seletivos são processados primeiro
  - Isso significa que menos registros precisam ser processados pela parte restante do plano

- Além disso, é dada preferência para filtros seletivos que possam se valer de índices
  - Pois é mais barato acessar um índice do que uma tabela

Importante lembrar

- A elaboração de um plano de execução depende de diversos fatores
  - Os operadores disponíveis (ex. Postgres tem mais opções do que o MySQL)
  - Estruturas de dados usadas (ex. Postgres usa Heap, MySQL usa árvores B+
    - O que leva o PostgreSQL a por vezes optar pelo HashJoin em vez do Nested Loop
  - Quantidade de recursos disponíveis na máquina

Importante lembrar

- É possível sugerir que o SGBD escreva um plano de forma diferente
- Exemplos
  - Escolhendo outros operadores SQL
  - Indicando o uso de estratégias de acesso aos dados
  - Solicitando que índices sejam ignorados ou usados

- Cada SGBD possui seus próprios mecanismos para controlar como essas sugestões são elaboradas

Atividade Individual

- Exemplo usado na introdução
  - A consulta abaixo retorna os títulos de filmes onde Brad Pitt trabalhou como ator ou como membro da equipe

Consulta levemente demorada.

SQL:

```
select distinct title
from movie  m

join movie_cast m_cast on m.movie_id = m_cast.movie_id
join person p_cast on m_cast.person_id = p_cast.person_id

join  movie_crew m_crew on m.movie_id = m_crew.movie_id
join  person p_crew on m_crew.person_id = p_crew.person_id

where (p_cast.person_name = 'Brad Pitt' OR p_crew.person_name = 'Brad Pitt')
order by 1;
```

Atividade Individual

- Foi disponibilizado no moodle
  - O banco de dados de movie, para trabalhar no MySQL
  - Os .dat do banco de movie, para trabalhar no DBest
  - A consulta original em SQL
  - Dois planos de execução DBest
    - Plano de execução original
      - Um plano semelhante ao gerado pelo MySQL para a consulta original
    - Um plano de execução otimizado

- O objetivo da atividade é analisar os dois planos e descobrir como gerar uma consulta SQL otimizada, cuja estrutura se assemelhe ao plano de execução otimizado gerado pelo DBest

Atividade Individual

- Para trabalhar no DBest
  - Arraste os .dat para a ferramenta
    - Isso deve gerar as tabelas no painel à esquerda
  - Arraste a consulta para a ferramenta
    - Isso deve gerar a árvore da consulta, que pode ser executada

- O arquivo fonte da consulta faz menção ao local onde os .dat estão
  - Mas, se os .dat forem arrastados primeiro, a consulta deve localizar os .dat, mesmo que que o caminho especificado no arquivo seja diferente

Atividade Individual

- Entregue um relatório contendo
  - Da consulta SQL original
    - A consulta
    - O plano de execução em MySQL
    - O tempo de execução no MySQL, gerado pelo EXPLAIN ANALYZE
  - Da consulta SQL otimizada
    - A consulta
    - o plano de execução no MySQL
    - o tempo de execução no MySQL, gerado pelo EXPLAIN ANALYZE