ÍNDICES EM BANCOS DE DADOS

Sérgio Mergen

Sumário

- Conceitos básicos
- Tipos de índice
  - Hash
  - Ordenado
- Tipos de Busca em árvores B+
  - Busca por igualdade/intervalo
  - Busca em índices compostos
- Tipos de árvores B+
    - Árvores clusterizadas/não clusterizadas
    - Uso no MySQL
    - Uso no PostgreSQL
- Definição dos atributos a indexar

Conceitos Básicos

- Tabelas muito grandes são mantidas no disco
  - Não cabem na memória primária

- Dada uma consulta

```
SELECT * FROM movie WHERE year = 2000
```

  - Deve-se localizar a página onde os registros solicitados se encontram
  - Sem índices, seria necessário acessar todas as páginas para encontrar os registros desejados

Conceitos Básicos

- Mecanismos de indexação são usados para acelerar o acesso aos dados.
  - Objetivo é reduzir o número de páginas de dados a serem carregados para a memória

- **Chave de busca** – atributo(s) usado(s) para localizar registros em um arquivo.
  - Ex. **CREATE INDEX** i1 **ON** movie(year);
  - No índice criado (i1), a chave de busca é **year**
    - Permite que os registros de movie sejam localizados com base em seu ano

Sumário

- Conceitos básicos
- Tipos de índice
  - Hash
  - Ordenado
- Tipos de Busca em árvores B+
  - Busca por igualdade/intervalo
  - Busca em índices compostos
- Tipos de árvores B+
    - Árvores clusterizadas/não clusterizadas
    - Uso no MySQL
    - Uso no PostgreSQL
- Definição dos atributos a indexar

Tipos de Índice

- Existem várias estruturas de dados que implementam índices
- É necessário analisar alguns critérios para escolher a melhor estrutura
  - Tempo de busca
  - Tempo de inserção
  - Tempo de remoção
  - Sobrecarga de espaço
  - Tipos de acesso eficientes suportados
    - Consulta por equivalência
      - year = 2000
    - Consulta por intervalo
      - year >2000
      - year >2000 **AND** year < 2015

Tipos de Índice

- Dois tipos básicos de índices:
  - **Índices Hash:** chaves de busca são distribuídas uniformemente em buckets usando uma função de hash.
  - **Índices ordenados:** chaves de busca são armazenadas em ordem.

Sumário

- Conceitos básicos
- Tipos de índice
  - Hash
  - Ordenado
- Tipos de Busca em árvores B+
  - Busca por igualdade/intervalo
  - Busca em índices compostos
- Tipos de árvores B+
    - Árvores clusterizadas/não clusterizadas
    - Uso no MySQL
    - Uso no PostgreSQL
- Definição dos atributos a indexar

Índices Hash

- Uma função hash *h*
  - mapeia todas chaves de busca para posições de um vetor(buckets).
  - é usada para localizar registros para acesso, inserção e remoção.
- Registros com valores de chave de busca diferentes podem ser mapeados para o mesmo bucket.
- Todo o bucket deve ser varrido sequencialmente para localizar o registro.
- Custo constante (se usada uma função boa)

Índices Hash

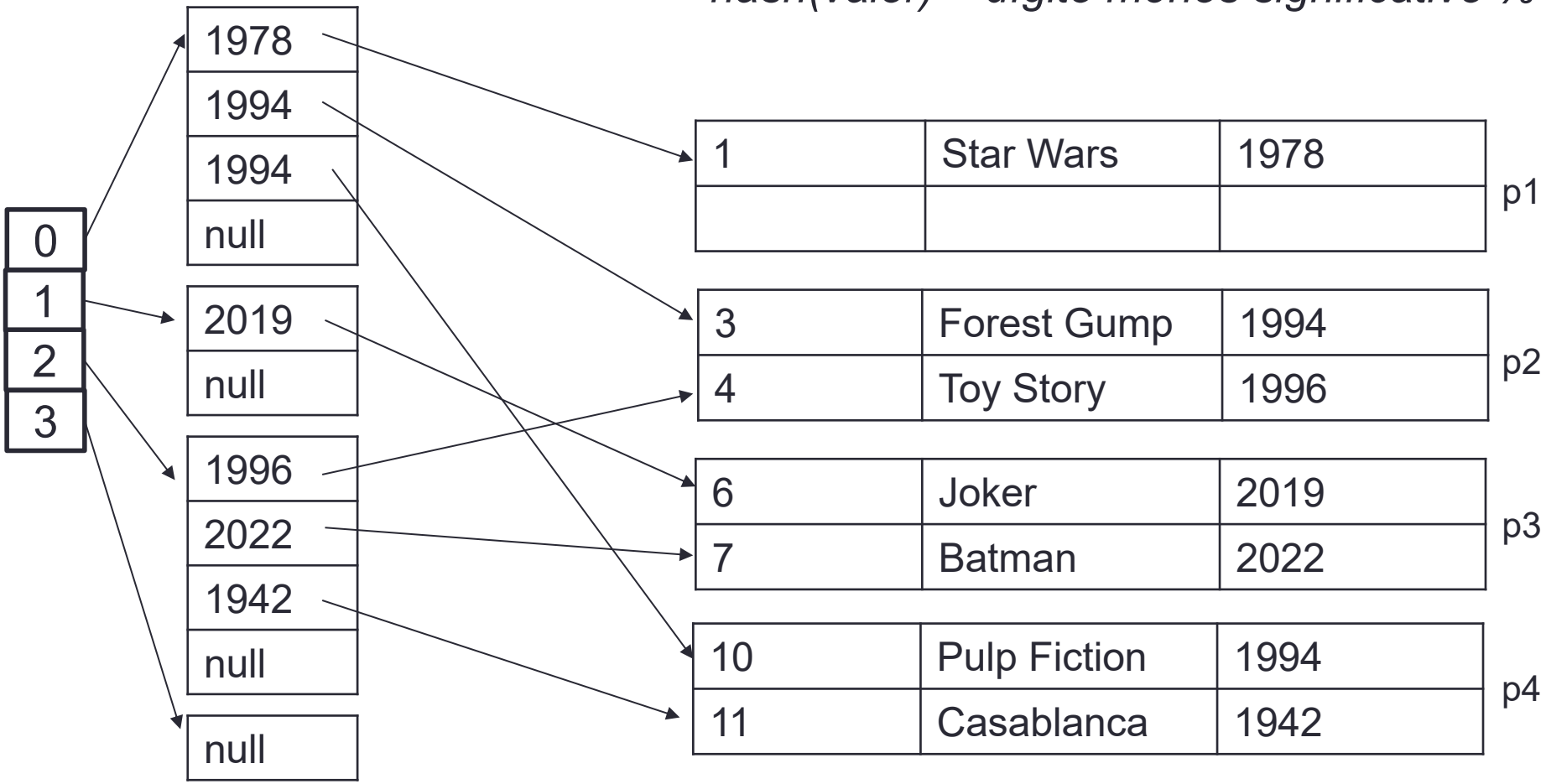
- Um índice hash pode ser usado de duas formas
  - Como um índice em memória
    - Nesse caso, o índice não tem relação nenhuma com a organização do arquivo
  - Como uma forma de organização de arquivo

Índices Hash (em memória)

Hash em memória

Exemplo de função hash:

hash(valor) = dígito menos significativo % 4

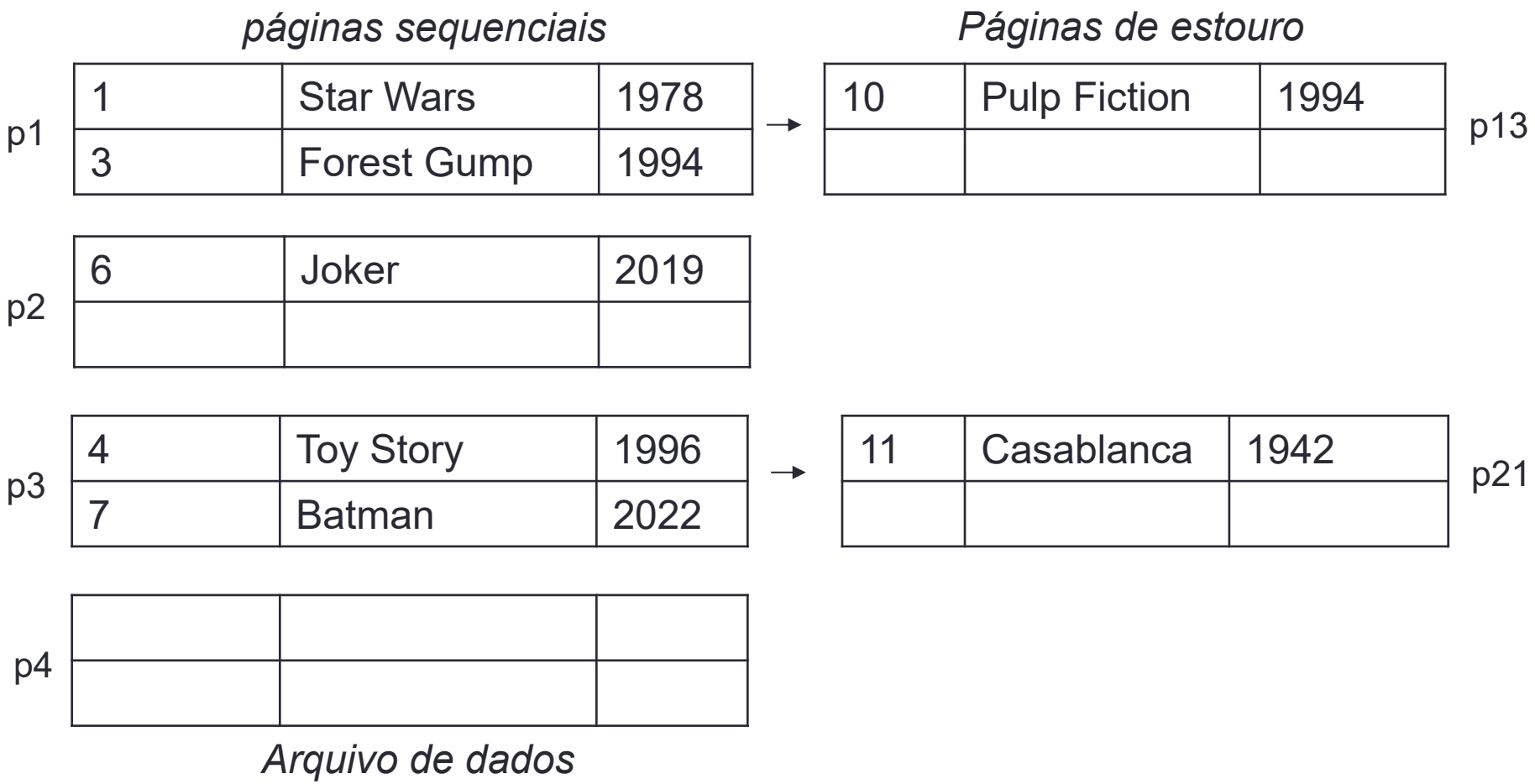


Arquivo de dados

Índices Hash (para organizar o arquivo)

Exemplo de função hash:

hash(valor) = dígito menos significativo % 4



Sumário

- Conceitos básicos
- Tipos de índice
  - Hash
  - Ordenado
- Tipos de Busca em árvores B+
  - Busca por igualdade/intervalo
  - Busca em índices compostos
- Tipos de árvores B+
    - Árvores clusterizadas/não clusterizadas
    - Uso no MySQL
    - Uso no PostgreSQL
- Definição dos atributos a indexar

Índices Ordenados

- Em um **índice ordenado**
  - as entradas do índice são ordenadas pela chave de busca.

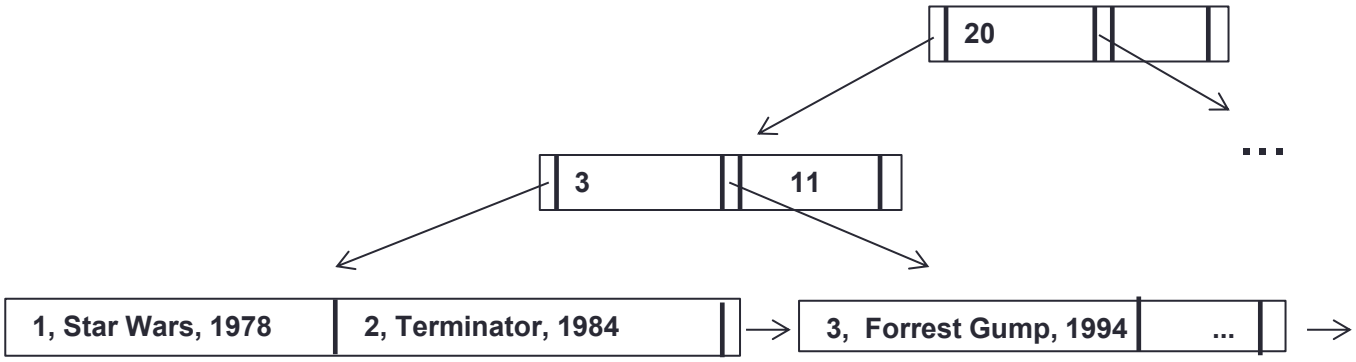
- Um **arquivo de índice** consiste em registros (chamados **entradas do índice**) na forma
  - Chave de busca
  - Ponteiro para o registro que possui a chave de busca

Índices Ordenados

- Arquivos de índice são tipicamente muito menores do que o arquivo original
  - Possuem um número menor de páginas
- Possivelmente todo o arquivo de índice caiba na memória
  - O que reduz o custo quando o índice deve ser utilizado
  - Pois não é necessário se preocupar com a transferência para a memória

Índices Ordenados

- Índices ordenados geralmente são implementados como árvores B+
- Uma árvore B+ armazena pares <chave, valor>
- No exemplo abaixo
  - Chave: idM
  - Valor: registro completo de movie



Comparação dos índices

- Índices Ordenados
  - Desempenho bom em consultas por intervalo
  - Desempenho bom em consultas por igualdade
  - Por isso, é o **índice preferido dos SGBDs**

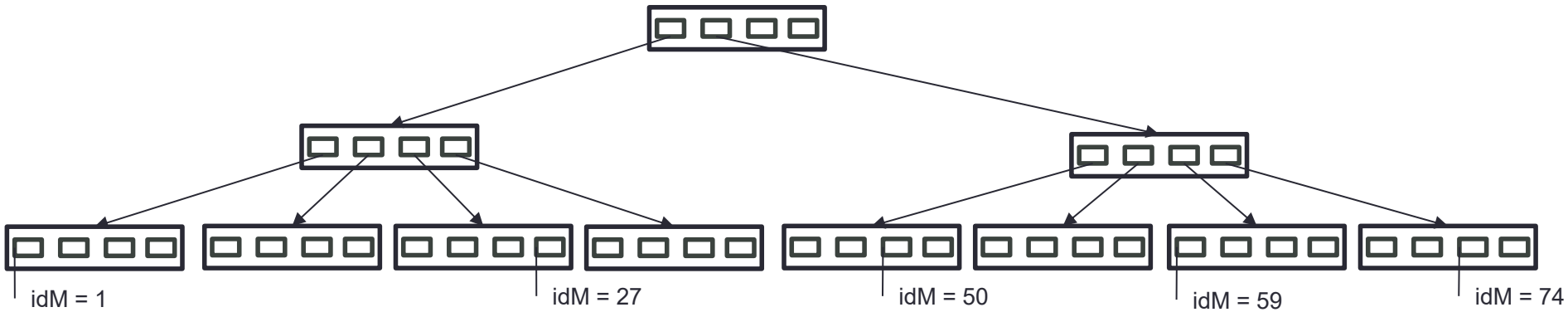
- Índices Hash
  - Desempenho ótimo em consultas por igualdade
  - Desempenho péssimo em consultas por intervalo
  - A função é muito dependente do domínio de dados
  - Se a função for ruim
    - Desempenho cai
  - Por isso, esse índice tem pouco suporte em SGBDs

Sumário

- Conceitos básicos
- Tipos de índice
  - Hash
  - Ordenado
- Tipos de Busca em árvores B+
  - Busca por igualdade/intervalo
  - Busca em índices compostos
- Tipos de árvores B+
    - Árvores clusterizadas/não clusterizadas
    - Uso no MySQL
    - Uso no PostgreSQL
- Definição dos atributos a indexar

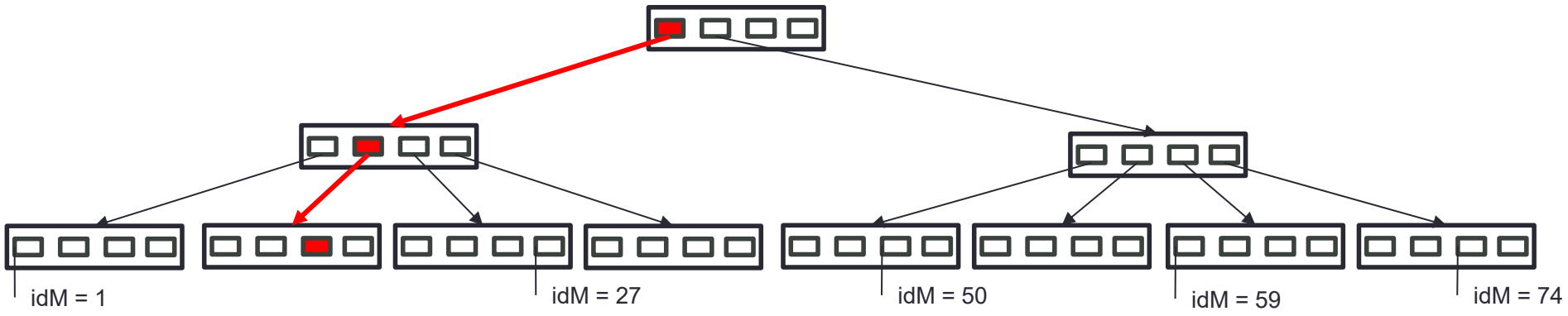
Árvores B+

- A seguir veremos exemplos de buscas que podem ser feitas sobre uma árvore B+
  - onde a chave de busca é a coluna idM



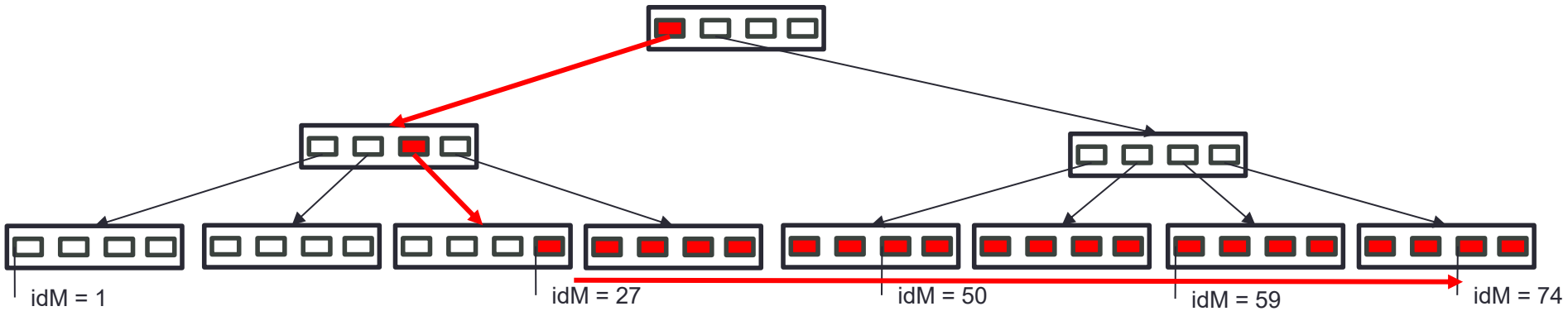
Árvores B+

- Busca por igualdade: idM = 9
  - A busca leva ao nó folha que possui idM=9, caso exista



Árvores B+

- Busca por intervalo: idM >= 27
  - A busca leva ao primeiro nó folha que possua idM >=27.
  - Todas as entradas a partir desse ponto são recuperadas



Sumário

- Conceitos básicos
- Tipos de índice
  - Hash
  - Ordenado
- Tipos de Busca em árvores B+
  - Busca por igualdade/intervalo
  - Busca em índices compostos
- Tipos de árvores B+
    - Árvores clusterizadas/não clusterizadas
    - Uso no MySQL
    - Uso no PostgreSQL
- Definição dos atributos a indexar

Índices compostos

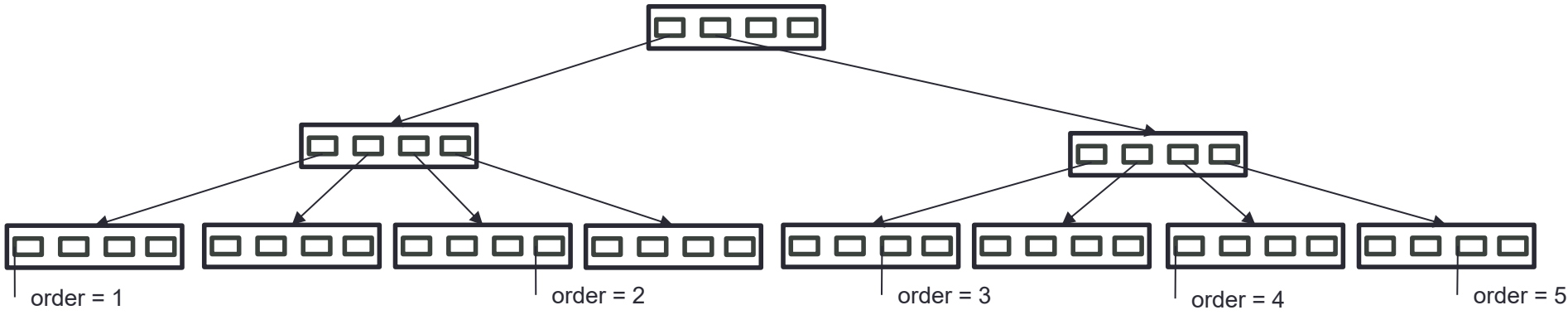
- Índice composto
  - Um índice criado sobre mais do que um atributo
  - Útil para consultas que filtrem por mais do que um atributo

  - Ex. **CREATE INDEX** i2 **ON** movie_cast(cast_order, c_name)

  - No caso acima, o índice teria dois níveis explícitos
    - O primeiro para cast_order
    - O segundo para c_name

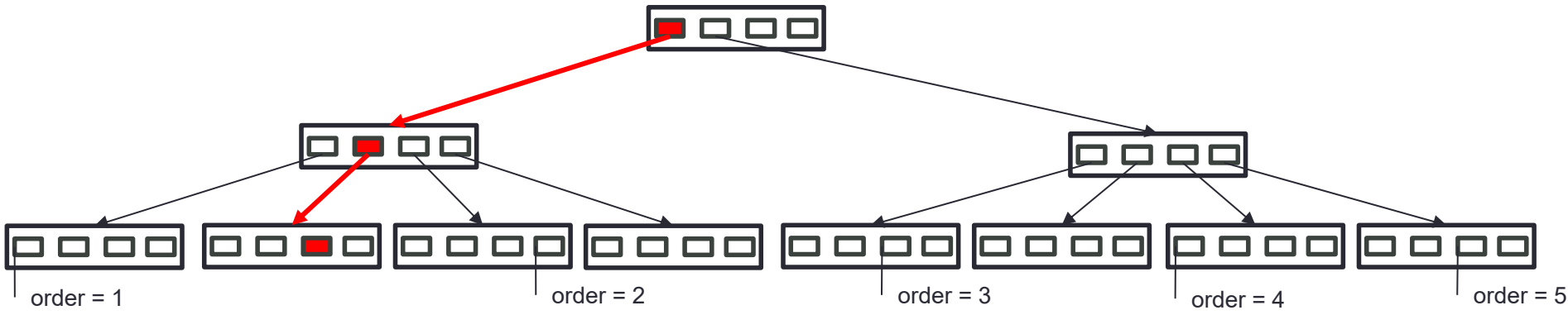
Busca em índices compostos

- O índice abaixo é composto por (cast_order, c_name)
- O nível das colunas no índice indica a ordenação das entradas do índice
  - Ou seja, as entradas são ordenadas primeiro por cast_order, e então por c_name.
    - Somente para as cast_orders iguais que é necessário ordenar por c_name
    - A linha vertical indica em qual entrada do índice aparece o primeiro valor de um cast_order específico



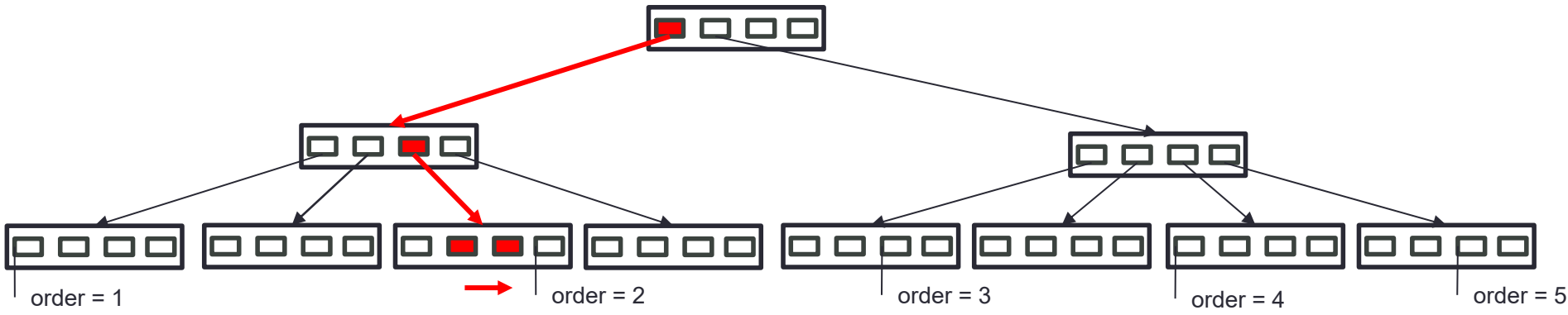
Busca em índices compostos

- **Busca por igualdade**: cast_order = 1 and c_name = ‘Jack’
- A árvore leva diretamente à entrada que satisfaz os dois filtros



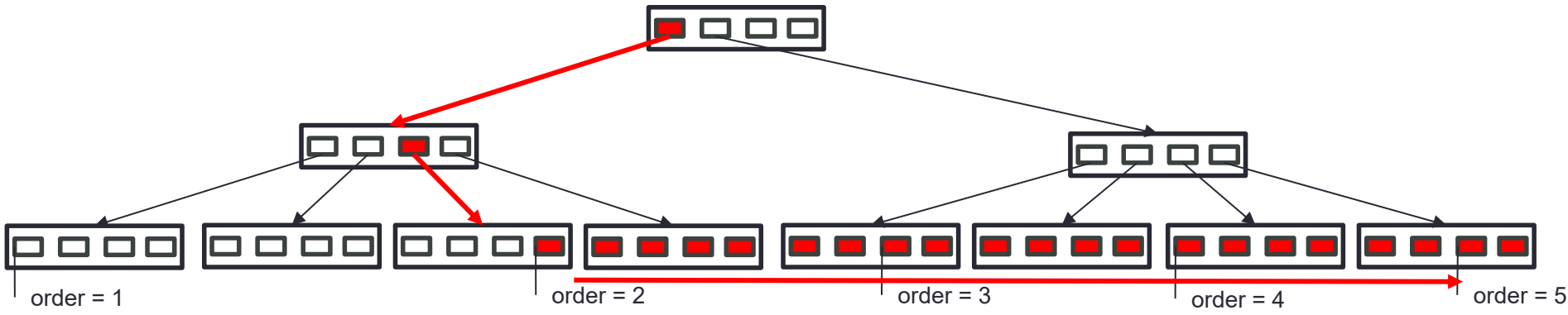
Busca em índices compostos

- **Busca por intervalo**: cast_ order = 1 and c_name > ‘Jack’
- A árvore leva diretamente à primeira entrada que satisfaz os dois filtros
- A partir desse ponto, são lidas as entradas seguintes que tenham cast_order = 1



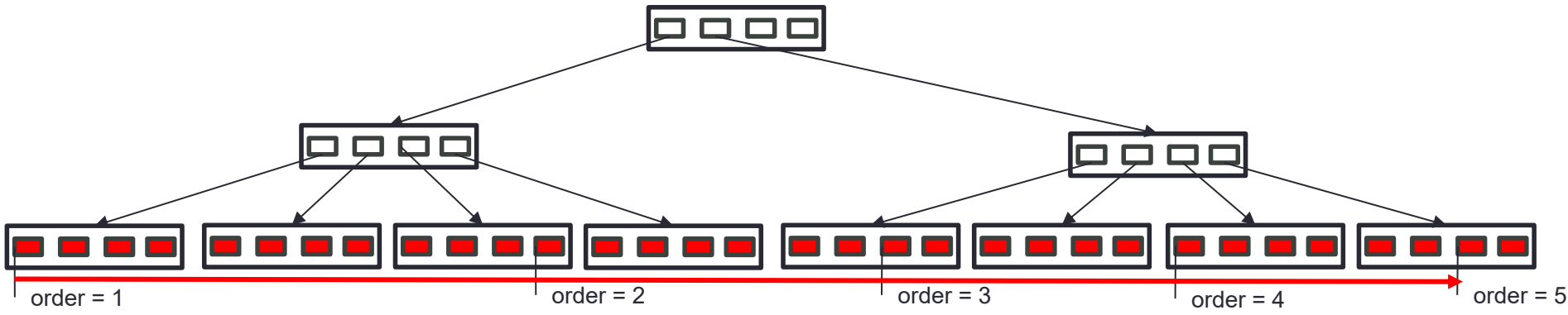
Busca em índices compostos

- **Busca por intervalo**: cast_ order > 1 and c_name = ‘Jack’
- A árvore leva diretamente à entrada que satisfaz o primeiro critério (cast_order > 1)
- A partir desse ponto, qualquer entrada pode conter c_name = ‘Jack’.
- Por isso é necessário ler todas as entradas seguintes



Busca em índices compostos

- **Filtros Disjuntivos**: cast_ order = 1 or c_name = ‘Jack’
- Qualquer entrada pode conter c_name = ‘Jack’.
- Por isso, é necessário ler toda a tabela



Busca em índices compostos

- Melhor cenário
  - Busca por igualdade em todos os níveis do índice composto
    - Permite um filtro mais completo
- Cenário médio
  - Busca por igualdade nos primeiros níveis X do índice composto e por intervalo no nível seguinte X + 1
    - Permite filtro apenas até esse nível (X+1)
    - Para os demais é necessário uma leitura sequencial
- Pior cenário
  - Busca disjuntiva (or)
    - O filtro não é possível
    - Todas as entradas do índice devem ser lidas

Sumário

- Conceitos básicos
- Tipos de índice
  - Hash
  - Ordenado
- Tipos de Busca em árvores B+
  - Busca por igualdade/intervalo
  - Busca em índices compostos
- Tipos de árvores B+
    - Árvores clusterizadas/não clusterizadas
    - Uso no MySQL
    - Uso no PostgreSQL
- Definição dos atributos a indexar

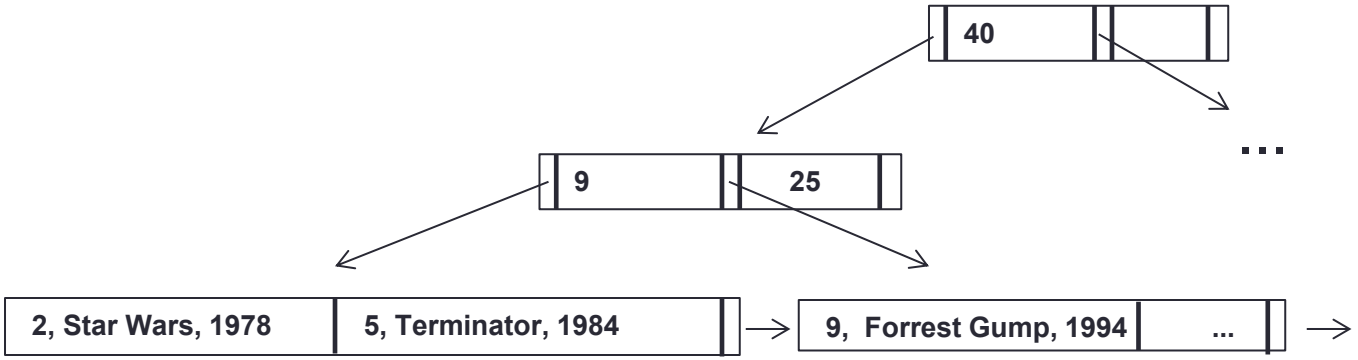
Árvores B+ clusterizadas

- Índice **clusterizado**
  - A própria árvore B+ armazena os registros
    - O nível folha guarda todos os registros
    - Formato das entradas: <chave de busca, registro completo>

- Cada tabela só pode ter um índice clusterizado
  - Criado sobre sua chave primária
  - Este índice também é chamado de índice primário

Árvores B+ clusterizadas

- Páginas no nível folha de um índice clusterizado
  - Contém os registros
  - Organizadas por ordem de chave primária dos registros



Árvores B+ não clusterizadas

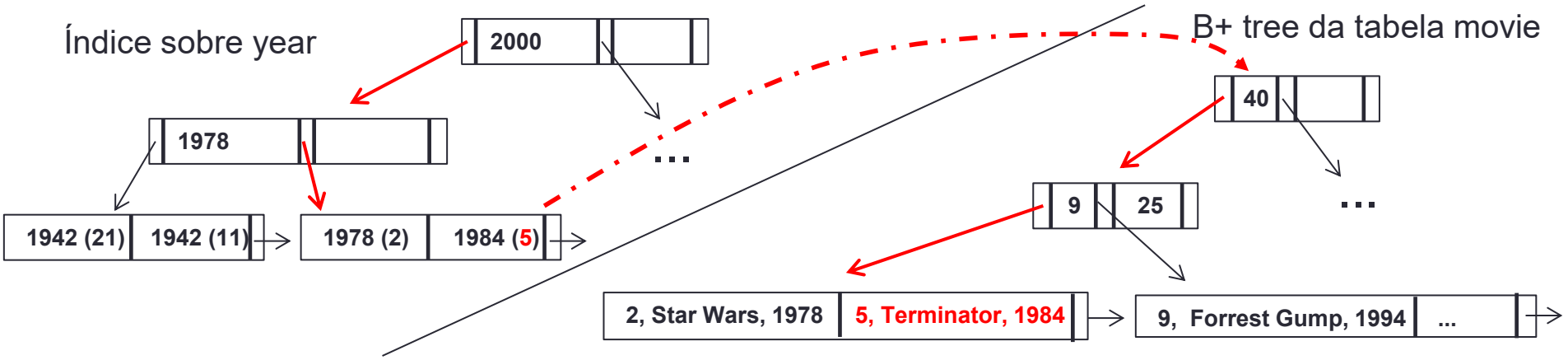
- Índice **não clusterizado**
  - É uma árvore usada apenas para facilitar o acesso por meio de uma chave de busca
    - o nível folha guarda um ponteiro para o registro
  - Também chamado de índice secundário

  - O formato do ponteiro depende de como as tabelas são organizadas
    - Como uma B+tree: o ponteiro é a chave primária do registro
    - Como uma heap: o ponteiro é o RID do registro

RID (Record ID): local físico onde o registro foi armazenado

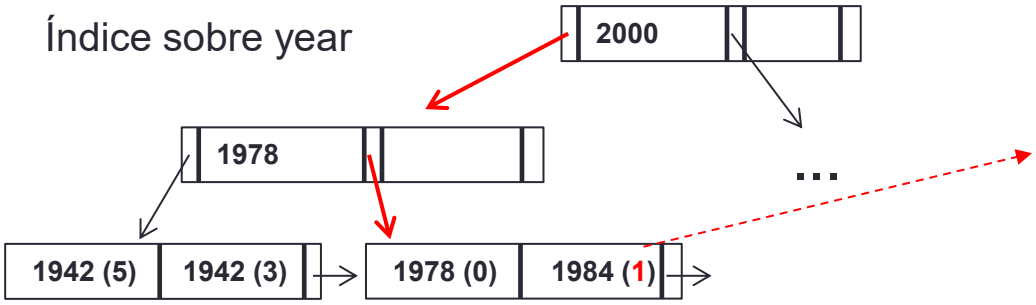
Árvores B+ não clusterizadas

- Se a tabela for organizada como uma B+ tree
  - O nível folha do índice secundário guarda a chave primária
  - Para acessar o registro, deve-se fazer uma busca na B+ tree que indexa essa chave
- Ex.
  - o filme com ano=1984 tem como ponteiro a chave primária 5



Árvores B+ não clusterizadas

- Se a tabela for organizada como uma heap
  - O nível folha do índice secundário guarda o RID do registro
  - Para recuperar o registro, deve-se acessar o arquivo na posição RID
- Ex.
  - o filme com ano=1984 tem como ponteiro o RID 1



heap da tabela movie

rid			
0	2	Star Wars	1978
1	5	Terminator	1984
2	9	Forrest Gump	1994
3	11	Bambi	1942
4	20	Joker	2019
5	21	Casablanca	1942

Sumário

- Conceitos básicos
- Tipos de índice
  - Hash
  - Ordenado
- Tipos de Busca em árvores B+
  - Busca por igualdade/intervalo
  - Busca em índices compostos
- Tipos de árvores B+
    - Árvores clusterizadas/não clusterizadas
    - Uso no MySQL
    - Uso no PostgreSQL
- Definição dos atributos a indexar

Índices no MySQL

- InnoDB é a principal engine do MySQL

- Essa engine usa árvores B+ para armazenar os dados de tabelas
  - Os próximos slides mostram como essa engine utiliza índices considerando essa estrutura

Índices no MySQL (engine InnoDB)

O índice primário de cada tabela é organizado por uma B+tree clusterizada.

Como o nível folha possui todos os registros, os dados da tabela são acessados por meio desse índice

Tabela movie

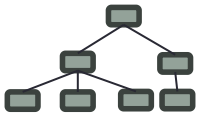


Tabela movie_cast

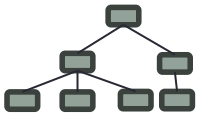
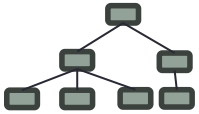


Tabela person



Esquema

Movie (idM, title, year)

Person (idP, p_name)

Movie_cast (idM, idP, c_name, ...)
  idM referencia movie
  idP referencia person

Estruturas MySQL

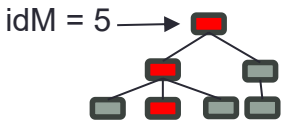
nome	Tipo	tipo	chave	valor
movie	tabela	B+tree clust.	idM	idM, title, year
movie_cast	tabela	B+tree clust.	idM, idP	idM, idP, c_name, ...
person	tabela	B+tree clust.	idP	idP, p_name
...	...	...	...	...

Índices no MySQL (engine InnoDB)

Consultas sobre a chave primária podem ser respondidas usando o índice primário

O nível folha dá acesso ao registro completo

Tabela movie



Esquema

Movie (idM, title, year)

Person (idP, p_name)

Movie_cast (idM, idP, c_name, ...)
  idM referencia movie
  idP referencia person

Estruturas MySQL

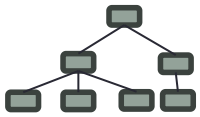
nome	Tipo	tipo	chave	valor
movie	tabela	B+tree clust.	idM	idM, title, year
movie_cast	tabela	B+tree clust.	idM, idP	idM, idP, c_name, ...
person	tabela	B+tree clust.	idP	idP, p_name
...	...	...	...	...

Índices no MySQL (engine InnoDB)

Índices secundários são B+trees não clusterizadas

O valor é a chave primária do registro referenciado

idx_year



Esquema

Movie (idM, title, year)

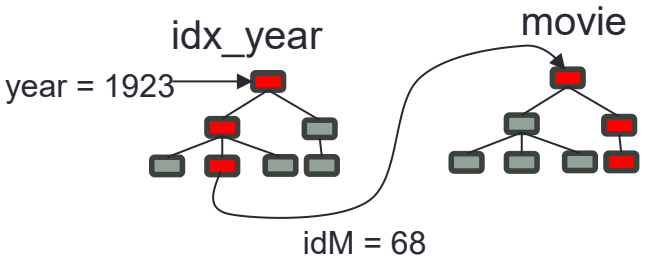
Estruturas MySQL

nome	Tipo	tipo	chave	valor
movie	tabela	B+tree clust.	idM	idM, title, year
Idx_year	Índice	B+tree ñ clust.	year	idM
...	...	...	...	...

Índices no MySQL (engine InnoDB)

Uma busca sobre uma coluna indexada pode ser respondida usando o índice secundário

O índice primário correspondente deve ser acessado para recuperar o registro completo



Esquema

Movie (idM, title, year)

Estruturas MySQL

nome	Tipo	tipo	chave	valor
movie	tabela	B+tree clust.	idM	idM, title, year
→ Idx_year	Índice	B+tree ñ clust.	year	idM
...	...	...	...	...

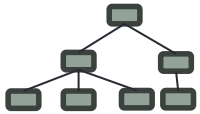
Índices no MySQL (engine InnoDB)

Chaves estrangeiras são automaticamente indexadas.

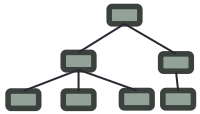
Uma chave estrangeira também é um índice secundário

O valor é a chave primária do registro referenciado

fk_mc_m



fk_mc_p



Esquema

Movie_cast (idM, idP, c_name, ...)
  idM referencia movie
  idP referencia person

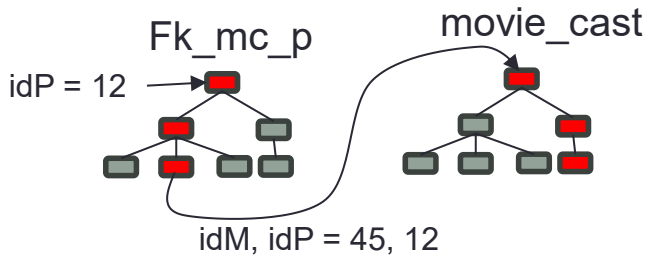
Estruturas MySQL

nome	Tipo	tipo	chave	valor
movie_cast	tabela	B+tree clust.	idM, idP	idM, idP, c_name, ...
Fk_mc_p	Índice	B+tree ñ clust.	idP	idM, idP
Fk_mc_m	índice	B+tree ñ clust.	idM	idM, idP
...	...	...	...	...

Índices no MySQL (engine InnoDB)

Uma busca sobre uma chave estrangeira pode ser respondida usando o índice

O índice primário correspondente deve ser acessado para recuperar o registro completo



Esquema

Movie_cast (idM, idP, c_name, ...)
  idM referencia movie
  idP referencia person

Estruturas MySQL

nome	Tipo	tipo	chave	valor
movie_cast	tabela	B+tree clust.	idM, idP	idM, idP, c_name, ...
Fk_mc_p	Índice	B+tree ñ clust.	idP	idM, idP
Fk_mc_m	índice	B+tree ñ clust.	idM	idM, idP
...	...	...	...	...

Sumário

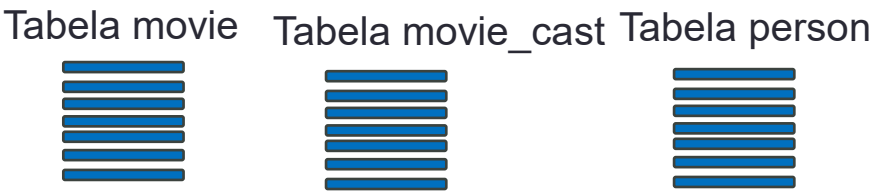
- Conceitos básicos
- Tipos de índice
  - Hash
  - Ordenado
- Tipos de Busca em árvores B+
  - Busca por igualdade/intervalo
  - Busca em índices compostos
- Tipos de árvores B+
    - Árvores clusterizadas/não clusterizadas
    - Uso no MySQL
    - Uso no PostgreSQL
- Definição dos atributos a indexar

Índices no PostgreSQL

- O PostgreSQL usa heaps para armazenar os dados de tabelas
  - Heap: estrutura de dados onde os registros são armazenados em qualquer espaço disponível, sem preocupação com ordem
  - O MySQL, versão MyIsam, também usa heaps
    - Existe um paralelo entre os índices no MyIsam e PostgreSQL
- Os próximos slides mostram como o PostgreSQL utiliza índices considerando essa estrutura

Índices no PostgreSQL

Tabelas armazenadas em arquivos heap
Registros:
- não preservam a ordem
- usam o endereço físico (RID) para localização



Esquema

Movie (idM, title, year)

Person (idP, p_name)

Movie_cast (idM, idP, c_name, ...)
idM referencia movie
idP referencia person

Estruturas PostgreSQL

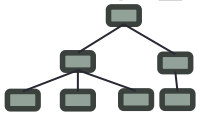
nome	Tipo	tipo	chave	valor
movie	tabela	heap	ridM	idM, title, year
movie_cast	tabela	heap	ridMC	idM, idP, c_name, ...
person	tabela	heap	ridP	idP, p_name
...	...	...	...	...

Índices no PostgreSQL

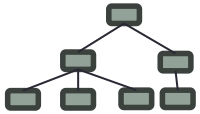
O índice primário é uma árvore B+ tree não clusterizada

O valor é a posição física (RID) onde o registro está localizado na heap

Índice pk_m



Índice pk_mc



Esquema

Movie (idM, title, year)

Movie_cast (idM, idP, c_name, ...)
  idM referencia movie
  idP referencia person

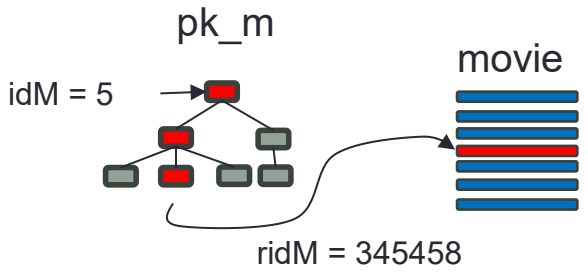
Estruturas PostgreSQL

nome	Tipo	tipo	chave	valor
movie	tabela	heap	ridM	idM, title, year
pk_m	índice	B+tree ñ clust.	idM	ridM
movie_cast	tabela	heap	ridMC	idM, idP, c_name, ...
pk_mc	índice	B+tree ñ clust.	idM, idP	ridMC
...	...	...	...	...

Índices no PostgreSQL

Buscas usando a chave primária podem ser respondidas usando o índice primário

A árvore B+tree remete à posição física onde o registro está localizado



Esquema

Movie (<u>idM</u>, title, year)

Movie_cast (<u>idM, idP</u>, c_name, ...)
  idM referencia movie
  idP referencia person

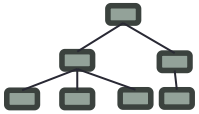
Estruturas PostgreSQL

nome	Tipo	tipo	chave	valor
movie	tabela	heap	ridM	idM, title, year
pk_m	índice	B+tree ñ clust.	idM	ridM
movie_cast	tabela	heap	ridMC	idM, idP, c_name, ...
pk_mc	índice	B+tree ñ clust.	idM, idP	ridMC
...	...	...	...	...

Índices no PostgreSQL

Índices secundários também são B+trees não clusterizadas que armazenam a posição física (RID) onde o registro está localizado

idx_year



Esquema

Movie (idM, title, year)

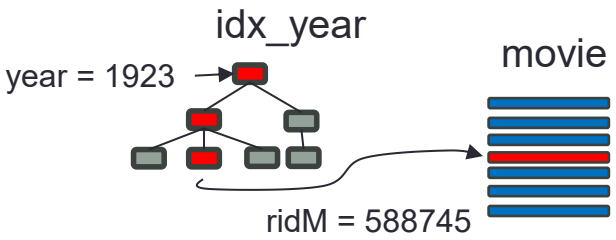
Estruturas PostgreSQL

nome	Tipo	tipo	chave	valor
movie	tabela	heap	ridM	idM, title, year
Idx_year	Índice	B+tree ñ clust.	year	ridM
...	...	...	...	...

Índices no PostgreSQL

Buscas usando uma coluna indexada podem ser respondidas usando o índice secundário

A árvore B+tree remete à posição física onde o registro está localizado



Esquema

Movie (idM, title, year)

Estruturas PostgreSQL

nome	Tipo	tipo	chave	valor
movie	tabela	heap	ridM	idM, title, year
Idx_year	Índice	B+tree ñ clust.	year	ridM
...	...	...	...	...

Índices no PostgreSQL

Chaves estrangeiras **não** são automaticamente indexadas. Cabe ao DBA decidir se compensa criar índices para as chaves estrangeiras

No exemplo, dois índices foram explicitamente criados
- movie_cast.idM
- movie_cast.idP

Esquema

Movie_cast (idM, idP, c_name, ...)
idM referencia movie
idP referencia person

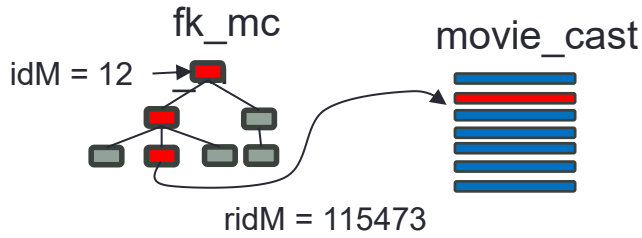
Estruturas PostgreSQL

nome	Tipo	tipo	chave	valor
movie_cast	tabela	heap	ridMC	idM, idP, c_name, ...
Fk_mc_m	Índice	B+tree ñ clust.	idM	ridMC
Fk_mc_p	índice	B+tree ñ clust.	idP	ridMC

Índices no PostgreSQL

Filtros envolvendo uma chave estrangeira podem ser respondidos recorrendo ao índice B+ tree

O índice remete à uma posição específica do arquivo heap



Esquema

Movie_cast (idM, idP, c_name, ...)
  idM referencia movie
  idP referencia person

Estruturas PostgreSQL

nome	Tipo	tipo	chave	valor
movie_cast	tabela	heap	ridMC	idM, idP, c_name, ...
Fk_mc_m	Índice	B+tree ñ clust.	idM	ridMC
Fk_mc_p	índice	B+tree ñ clust.	idP	ridMC

Sumário

- Conceitos básicos
- Tipos de índice
  - Hash
  - Ordenado
- Tipos de Busca em árvores B+
  - Busca por igualdade/intervalo
  - Busca em índices compostos
- Tipos de árvores B+
    - Árvores clusterizadas/não clusterizadas
    - Uso no MySQL
    - Uso no PostgreSQL
- Definição dos atributos a indexar

Definição de índices

- Índices aceleram a localização de registros
- No entanto, eles apresentam sobrecarga
  - Na atualização
  - No espaço ocupado
- Por isso, não vale a pena indexar todos os atributos de todas as tabelas
  - Os índices seriam maiores do que as tabelas
  - A inserção de um registro seria muito mais demorada
- A escolha de quais atributos terão índices deve ser criteriosa

Critérios na escolha dos índices

- Que critérios usar?
  - Indexar atributos muito usados em junções e filtros
  - Atributos usados por outras partes da consulta também podem ser indexados
    - Cláusula ORDER BY
    - Cláusula GROUP BY
    - ...
  - Indexar atributos com alta seletividade
    - Atributos que, se filtrados, tragam poucos registros
- Obs. Não se deve indexar todos atributos com essas características
  - Deve-se escolher aqueles que acarretem no maior benefício

Indexar atributos seletivos

- Seletividade de um atributo
  - Relativa à quantidade de registros que se espera recuperar a cada consulta
- Quanto menos registros são recuperados
  - Mais seletiva é a consulta
  - Mais vantajoso é usar um índice
- Ex.
  - sexo = ‘F’
    - Pouco seletivo (muitos registros são recuperados)
  - Nome = ‘Fulano Beltrano’
    - bastante seletivo (poucos registros são recuperados)

Exercício

- Qual a ordem de importância dos atributos abaixo considerando a criação de índices?

- Projeto
  - Id
  - Nome
  - Situação
  - Custo
  - Data_inicio
  - Data_termino
  - url

Exercício

- A consulta abaixo poderia se valer de um índice sobre nível
- Valeria a pena criar esse índice?

```
SELECT salário
FROM func
WHERE nivel = 'A1'
```

Estatísticas

N(func) = 1.000.000

V(nível, func) = 5

Exercício

- A consulta abaixo poderia se valer de um índice sobre nível
- Valeria a pena criar esse índice?

```
SELECT salário
FROM func
WHERE nivel = 'A1'
```

Estatísticas
N(func) = 1.000.000
V(nível, func) = 5

Não. Ele é pouco seletivo.
20% do arquivo teria que ser acessado

Exercício

- A consulta abaixo poderia se valer de um índice sobre nome
- Valeria a pena criar esse índice?

```
SELECT nome, data_termino
FROM projeto
WHERE nome<>'ACME'
```

Estatísticas

N(projeto) = 1.000.000

V(nome, projeto) = 1.000.000

Exercício

- A consulta abaixo poderia se valer de um índice sobre nome
- Valeria a pena criar esse índice?

```
SELECT nome, data_termino
FROM projeto
WHERE nome<>'ACME'
```

Estatísticas
N(projeto) = 1.000.000
V(nome, projeto) = 1.000.000

Não. Ele é pouco seletivo.
Praticamente todos os registros seriam retornados, de forma não sequencial

Exercício

- A consulta abaixo poderia se valer de um índice sobre nome
- Valeria a pena criar esse índice?

```
SELECT nome, data_termino
FROM projeto
WHERE nome > 'X'
```

Estatísticas

N(projeto) = 1.000.000

V(nome, projeto) = 1.000.000

Exercício

- A consulta abaixo poderia se valer de um índice sobre nome
- Valeria a pena criar esse índice?

```
SELECT nome, data_termino
FROM projeto
WHERE nome > 'X'
```

Estatísticas
N(projeto) = 1.000.000
V(nome, projeto) = 1.000.000

Sim. Poucos projetos devem iniciar com a letra X.

Uma ínfima parte do arquivo teria que ser acessada

Exercício

- Qual o conjunto mínimo de índices você criaria?
  - (contratacao)
  - (contratacao, exoneracao)
  - (exoneracao)
  - (exoneracao, contratacao)

```
SELECT nome
FROM func
WHERE contratacao < 2001 AND exoneracao =2014
```

Exercício

- Qual o conjunto mínimo de índices você criaria?
  - (contratacao)
  - (contratacao, exoneracao)
  - (exoneracao)
  - (exoneracao, contratacao)

```
SELECT nome
FROM func
WHERE contratacao < 2001
```

```
SELECT nome
FROM func
WHERE exoneracao > 2001
```

Exercício

- Qual o conjunto mínimo de índices você criaria?
  - (contratacao)
  - (contratacao, exoneracao)
  - (exoneracao)
  - (exoneracao, contratacao)

```
SELECT nome
FROM func
WHERE contratacao < 2001 AND
exoneracao =2014
```

```
SELECT nome
FROM func
WHERE contratacao < 2001
```

```
SELECT nome
FROM func
WHERE contratacao > 2001
```

```
SELECT nome
FROM func
WHERE exoneracao = 2011
```

Exercício

- No moodle foi disponibilizado um arquivo com dados e um script SQL
- Crie um banco de dados conforme script postado no moodle
  - Para acelerar a importação, use as dicas de otimização mencionadas no script

Exercício

- Execute a seguinte consulta

```
SELECT SQL_NO_CACHE    *    FROM proj
WHERE custoEstimado = 70000 AND custoReal = 1760000
and duracaoEstimada = 1
```

- Obs.
  - A flag SQL_NO_CACHE serve para impedir que o tempo reduza entre uma execução e outro devido ao cache
  - Isso é interessante para medição de desempenho

Exercício

- Use uma estratégia baseada em índices para acelerar a busca

- Dica: bons candidatos para indexação são as colunas com maior seletividade

Exercício

- Perguntas:
  1. Quanto tempo levou o script de importação?
  2. Quantos registros foram importados?
  3. Quais registros foram recuperados pela consulta?
  4. Quanto tempo levou para executar a consulta?
  5. Que índice(s) você criou?
  6. Com o(s) índice(s) à disposição, quanto tempo a consulta levou?

Atividade Individual

- Foi disponibilizado no moodle um script de criação de um banco de dados de filmes (movie)
  - Crie esse banco de dados
- Elabore uma consulta que retorne
  - títulos de filmes que tenham no nome um prefixo do nome de algum personagem
    - O personagem não precisa necessariamente ser do mesmo filme
- Crie um índice para acelerar a execução da consulta

Atividade Individual

Exemplo da tabela de resposta

Title	Character_name
Ted	Ted the Bellhop
RED	Red Six (Porkins)
RED	Red Three (Biggs)
RED	Red Leader
RED	Red Two (Wedge)
RED	Red Four (John D)
RED	Red Two (voice) (uncredited)
Forrest Gump	Forrest Gump Jr.
Dick	Dick Cavett
...	...

Atividade Individual

- Escreva um relatório simples em PDF contendo
  - SQL da consulta criada
  - Índice criado
  - Tempo de resposta da consulta sem uso do índice
  - Tempo de resposta da consulta com uso do índice

Atividade Individual

- Para que a medição seja mais precisa
  - Use a flag SQL_NO_CACHE
  - Execute a consulta algumas vezes e faça uma média dos tempos de resposta
    - Ignore o tempo de resposta da primeira execução
  - Use IGNORE INDEX quando a intenção for executar a consulta sem recorrer ao índice criado

  - Ex.

```
SELECT SQL_NO_CACHE…
FROM … IGNORE INDEX (idx_col1)
WHERE …;
```