Banco de Dados II

Prof. Edson Kaneshima

edson.kaneshima@unifil.br

Indexação e Hashing

- Muitas consultas referenciam apenas uma pequena porção dos registros em uma tabela.
- Portanto necessitamos ser capaz de localizar estes registros diretamente, para este fim podemos projetar estruturas adicionais que são associadas às tabelas.
- Estudaremos duas alternativas : Construção de índices e função de hashing.
- Nenhuma técnica é a melhor, cada técnica tem suas vantagens e desvantagens, bem como adaptação a cada situação do banco de dados.

Indexação e Hashing

Consultas x quantidade de registros

- Encontrar todas as contas da agência Perryridge
- Encontrar o saldo do número da conta A-101

Número_conta	Nome_agência	Saldo
A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

Indexação e Hashing

Para a escolha de cada técnica, devemos sempre lembrar:

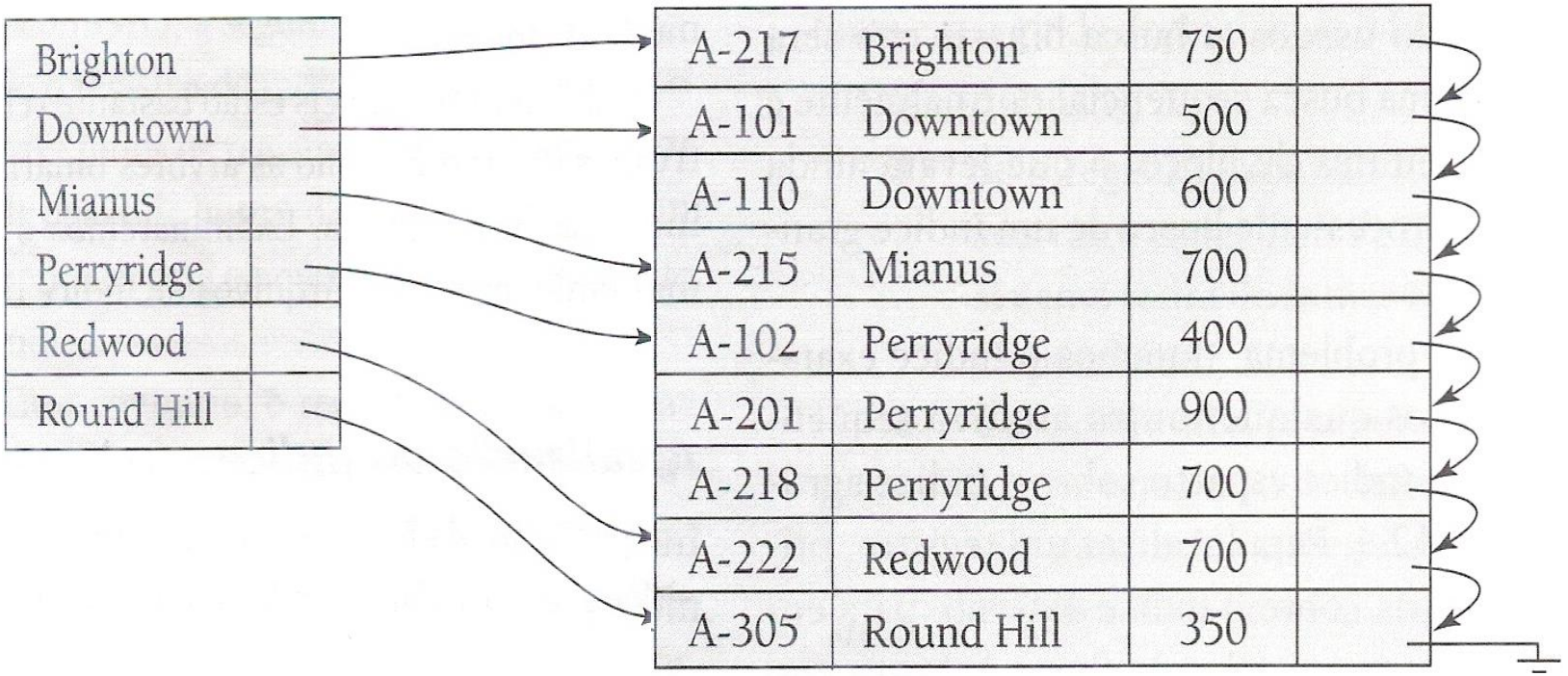
- tempo de acesso: o tempo para encontrar um item de dado particular, usando a técnica em questão;
- tempo de inserção: o tempo para inserir um novo item de dado, inclui o tempo para encontrar o lugar correto e atualizar as estruturas de índices;
- tempo de remoção: o tempo para remover um item de dado, inclui o tempo para atualizar as estruturas de índices;
- espaço extra: o espaço adicional ocupado por uma estrutura de índice (consumo de disco);
- é normal desejarmos mais de um índice ou função de hashing por tabela.

Indexação

- Cada estrutura de índice está associada a uma chave de busca particular.
- O índice primário é o fator de ordenação básico da tabela, normalmente o índice primário utiliza a chave primária.
- Os demais índices são chamados de índice secundário.
- Quanto a técnica de construção os índices podem ser:
  - Índice denso: um registro de índice aparece para cada valor da chave de busca da tabela e contém o valor da chave de busca e o ponteiro para o registro.
  - Índice esparso: registros de índices são criados somente para alguns registros, localizando o registro com o valor menor ou igual ao desejado, depois pesquisa seqüencial.

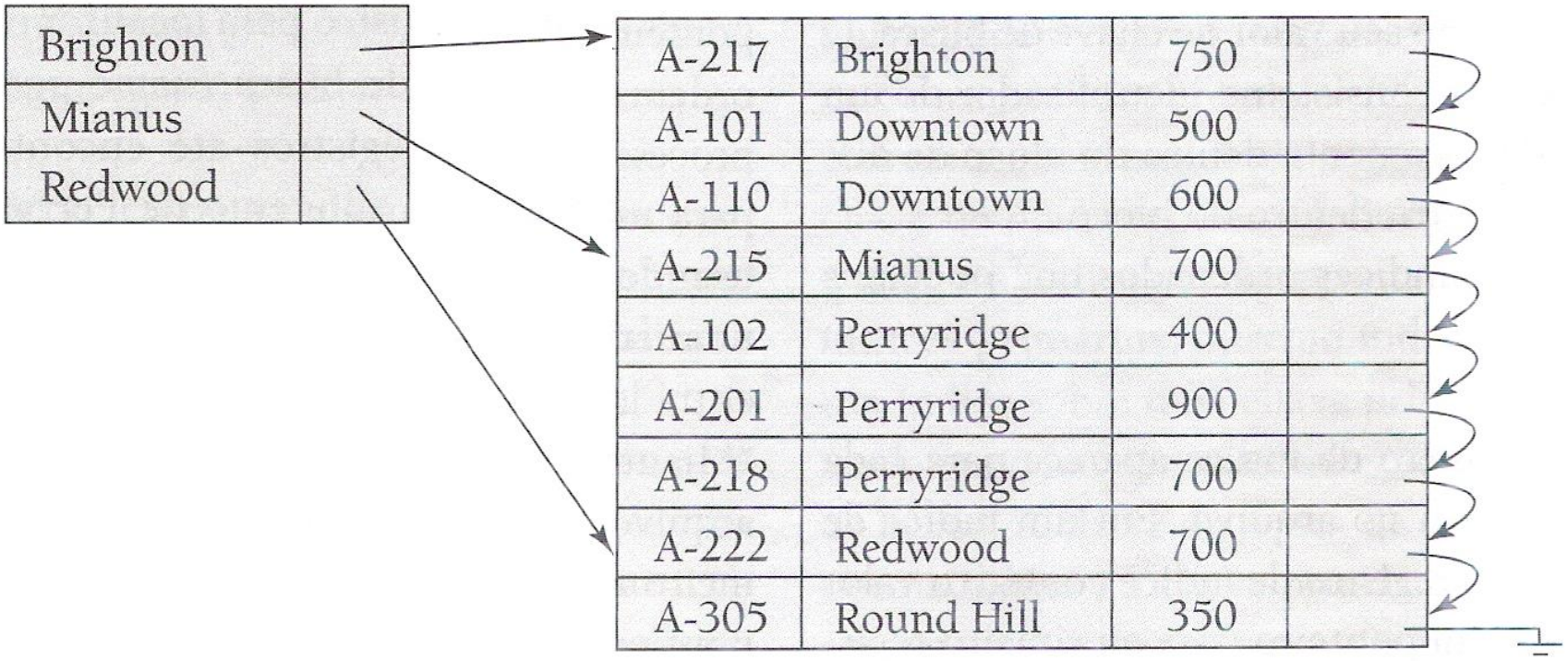
Indexação

- Índice Denso
  - Registro de índice para cada valor de chave de busca



Indexação

- Índice Esparso
  - Localiza pela chave de busca com valor menor ou igual
  - Ocupam menor espaço
  - Tem menor sobrecarga de manutenção



Indexação

- Índice primário utiliza sempre índice denso.
- Índice denso geralmente é mais rápido para localizar um registro, mas gasta mais espaço em disco.
- Índices esparso requerem menos espaço em disco e impõem menos sobrecarga de manutenção para inserção e remoção de registros.
- Pesquisa em um índice esparso é feita uma parte no índice e outra na tabela, sobrecarregando um pouco a pesquisa.
- Um bom equilíbrio para o índice esparso seria uma entrada de índice para cada bloco de dados, pois a cada leitura os registros daquele bloco são lidos junto e podem otimizar as próximas leituras.

Indexação

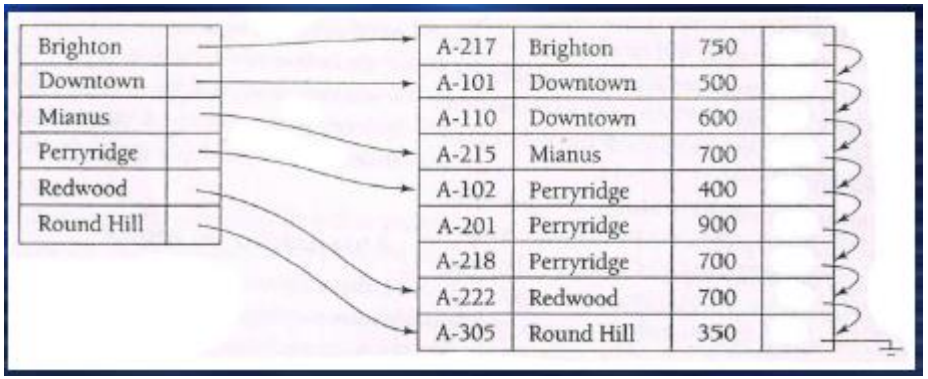
- Inserção: índice denso é obrigatório a atualização, já o esparso, só é necessário atualizar se um bloco novo for criado, gravando esta chave.

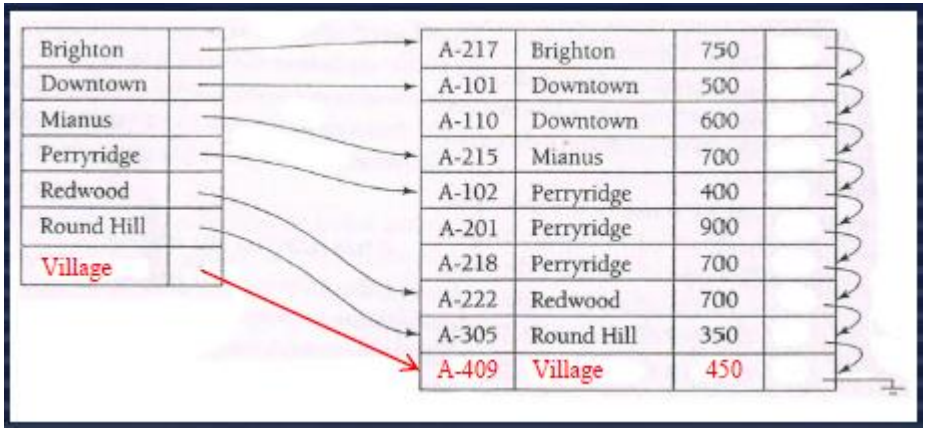
- Remoção: índice denso é obrigatório a atualização, já o esparso, só é necessário atualizar se a chave de busca removida for exatamente a residente no índice.

- Alteração: índice denso não permite alterar o valor de uma chave, tem que remover e inserir novamente. Já o esparso permite alterar uma chave de busca, podendo necessitar uma nova entrada ou não no índice.

Indexação

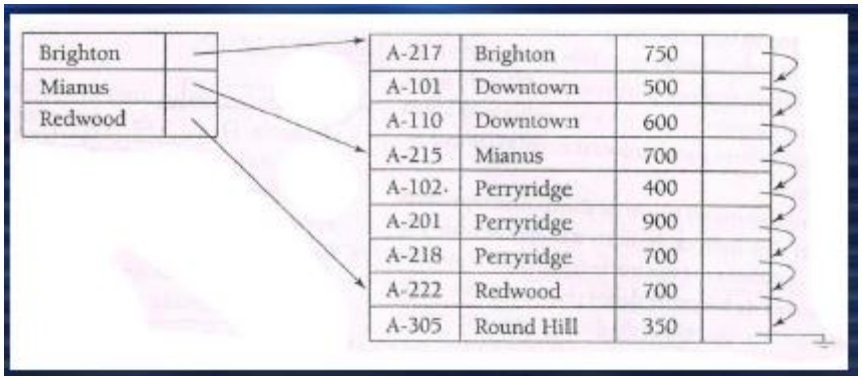
- Inserção em índices densos

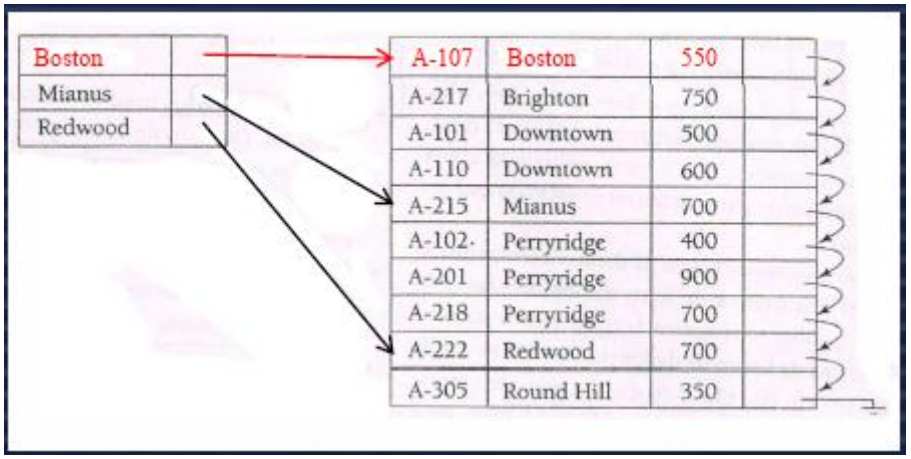




Indexação

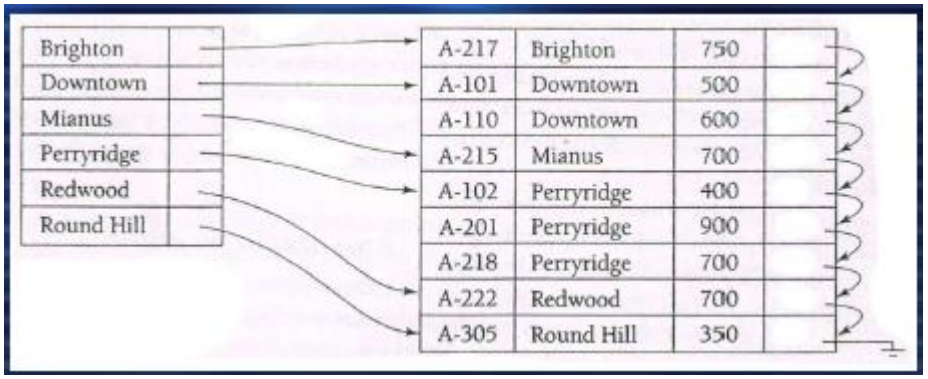
- Inserção em índices esparsos

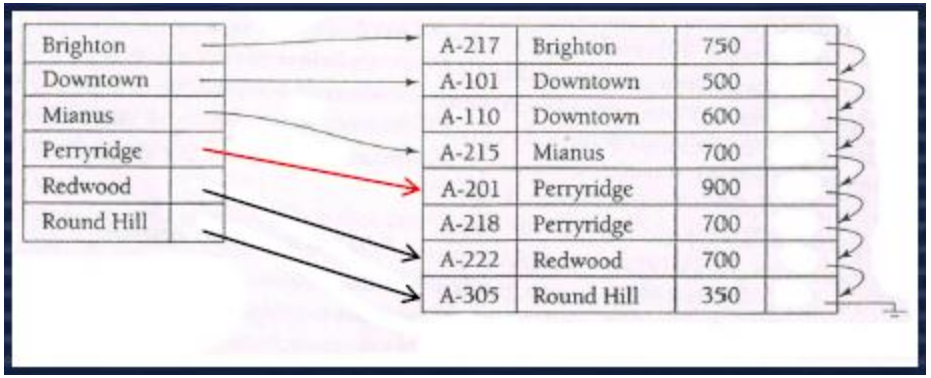




Indexação

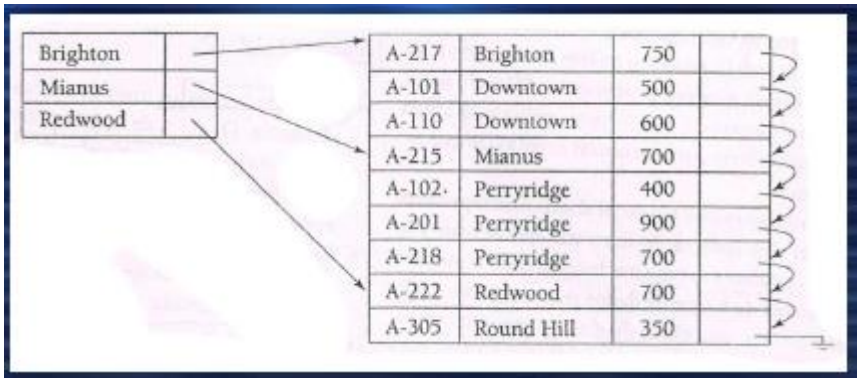
- Exclusão em índices densos – conta A-102 foi excluída

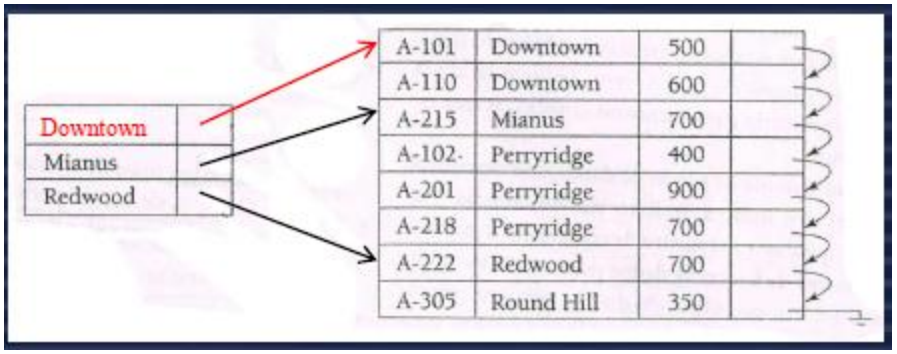




Indexação

- Exclusão em índices esparsos – conta A-217 foi excluída



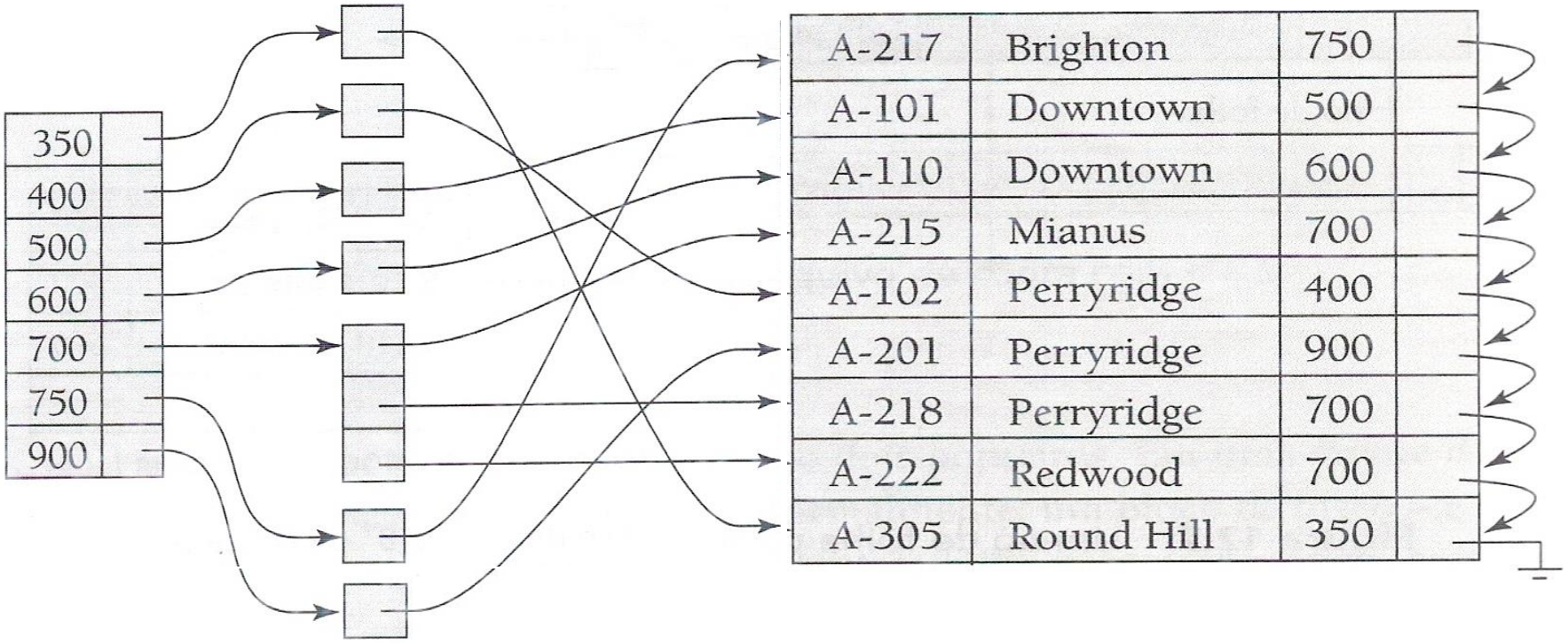


Índice secundário

- Índices secundário não apontam diretamente para a tabela, e sim para um bucket e este aponta para a tabela.
- Isto significa que do índice secundário para o bucket, é índice esparso e do bucket para a tabela é índice denso.
- Os buckets também podem utilizar a técnica do índice esparso, reduzindo o seu tamanho consideravelmente.
- Os índices secundários aumentam o desempenho de consultas que utilizam chaves diferentes da primária, mas isto impõem uma sobrecarga séria e grande nas modificações sofridas pela tabela.

Índice secundário

- Índice secundário com o campo saldo.



Árvore B+

- A medida que uma tabela cresce e seu índice também, o tempo de resposta começa a degradar.

- Para poder otimizar um pouco esta situação, o índice pode adotar uma arquitetura de árvore para encadeamento das informações.

- Árvore B+ tem a forma balanceada da raiz até as folhas, assim todo caminho da raiz até a folha é igual e previsível (n/2) onde n é o número de registros da tabela.

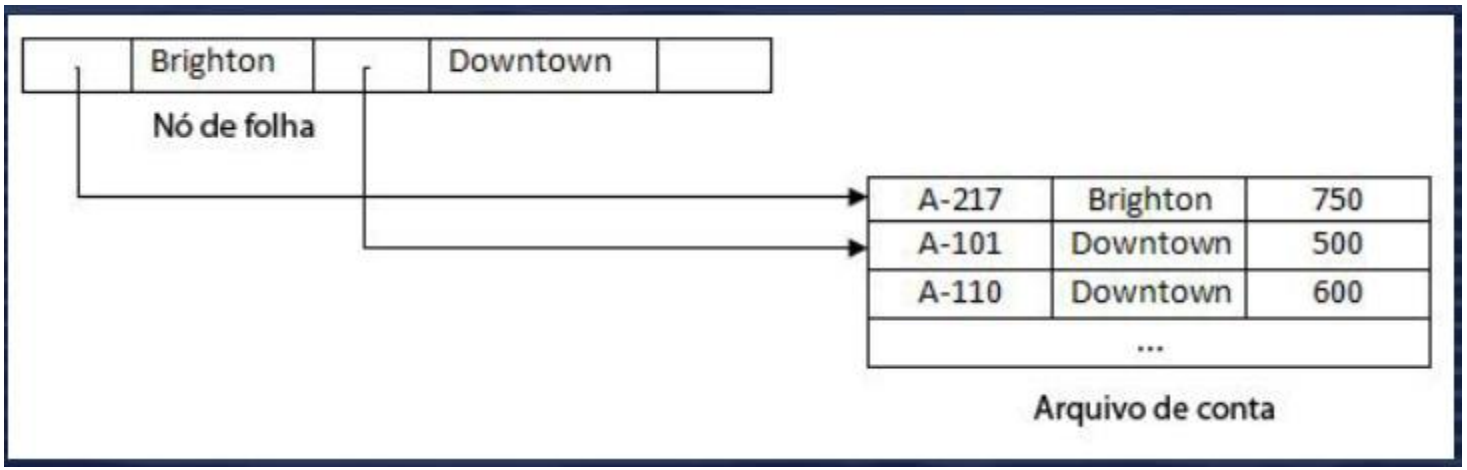
Árvore B+

- O endereço do registro está sempre em um nó folha, ou seja é necessário percorrer toda a árvore (da raiz até a folha) para identificar um registro.

- O B+ indica um “balanceamento” de carga, o que permite a igualdade de tempo para cada registro solicitado.

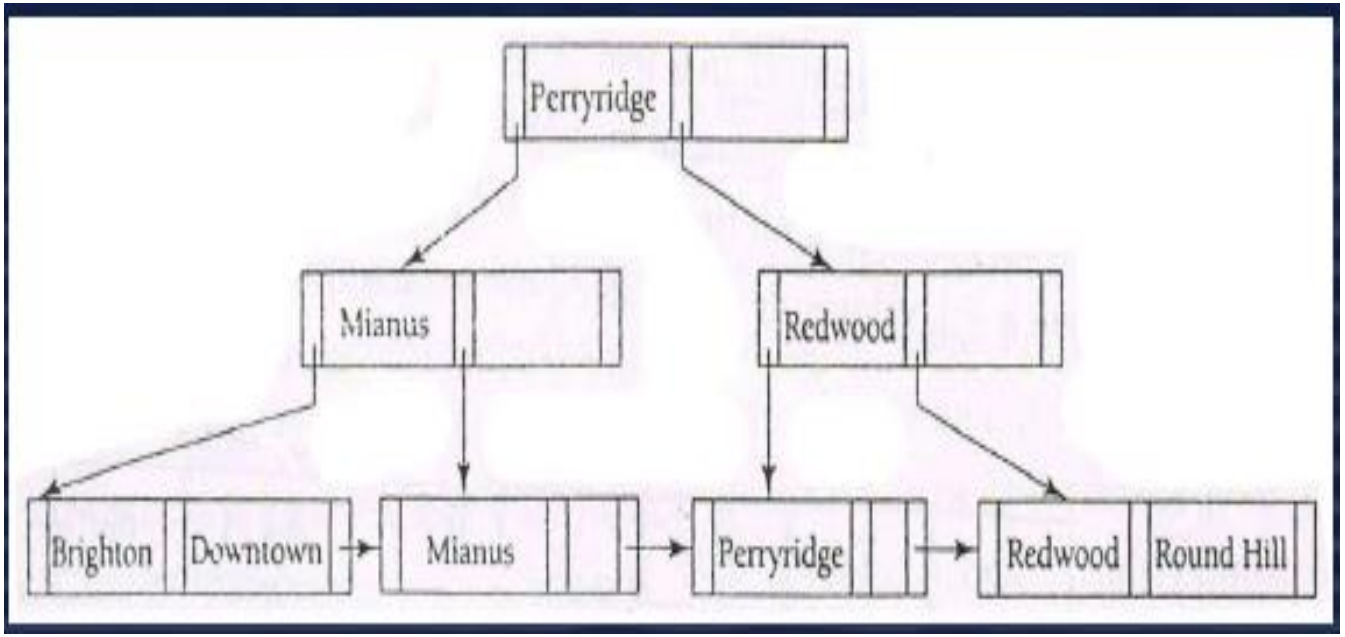
- Inserção e remoção são difíceis devido às constantes necessidades de atualização no nó galho afim de manter a árvore balanceada.

Estrutura de uma Árvore B+



Estrutura de uma Árvore B+

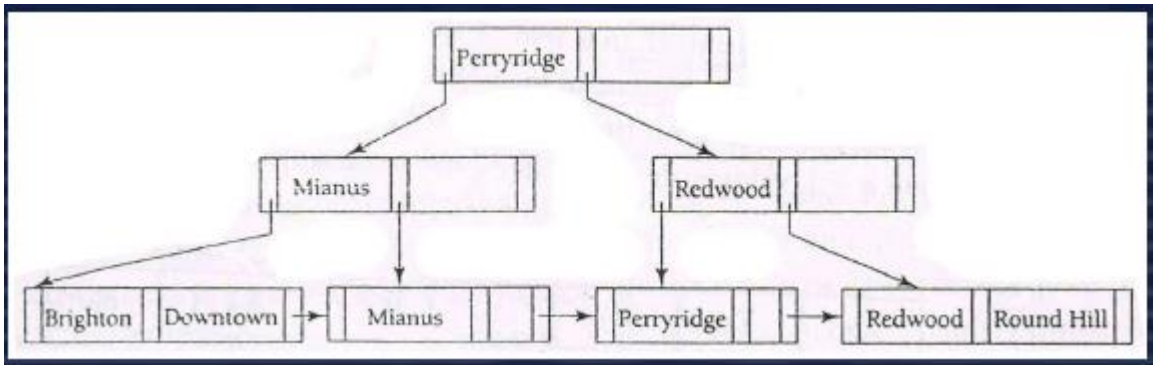
- Nós não-folha formam um índice esparso
- Consultas em árvore B+
  - Pesquisar chave de busca Downtown, Perryridge e Village



Atualizações em Árvore B+

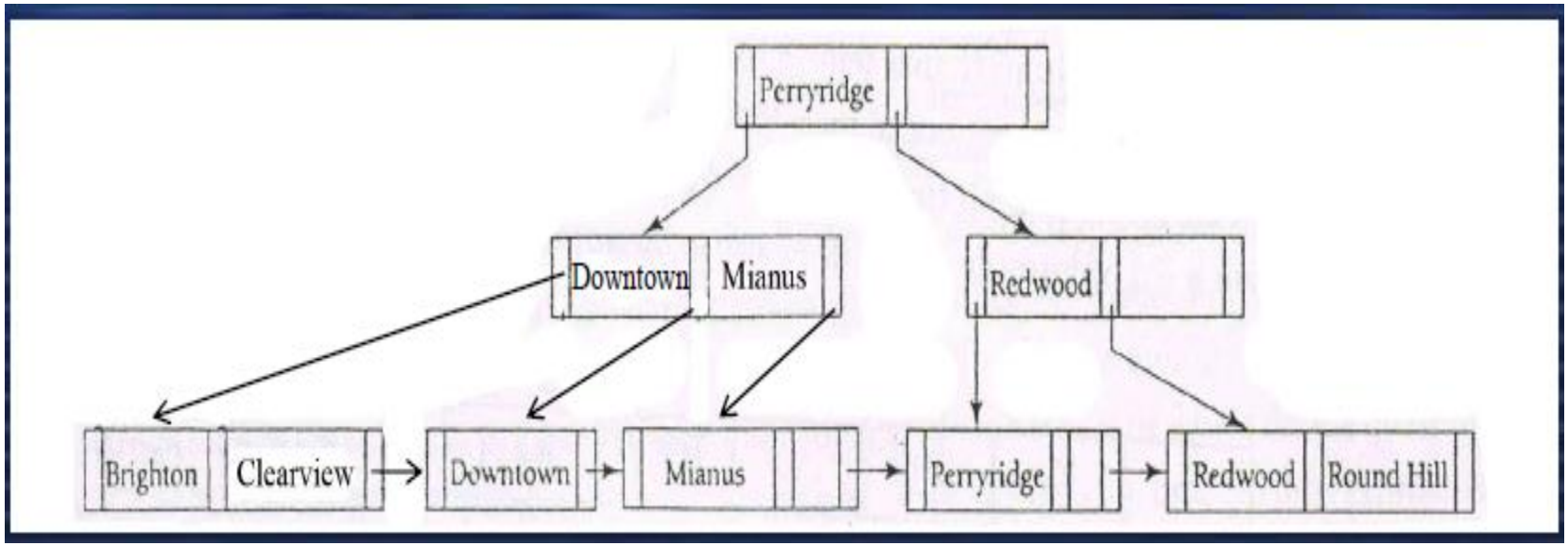
- Inserção ou exclusão
  - Semelhante a técnica da consulta
  - Pode necessitar divisão ou união de nós para garantir o balanceamento

- Inserir um registro com valor igual a “CleanView”.



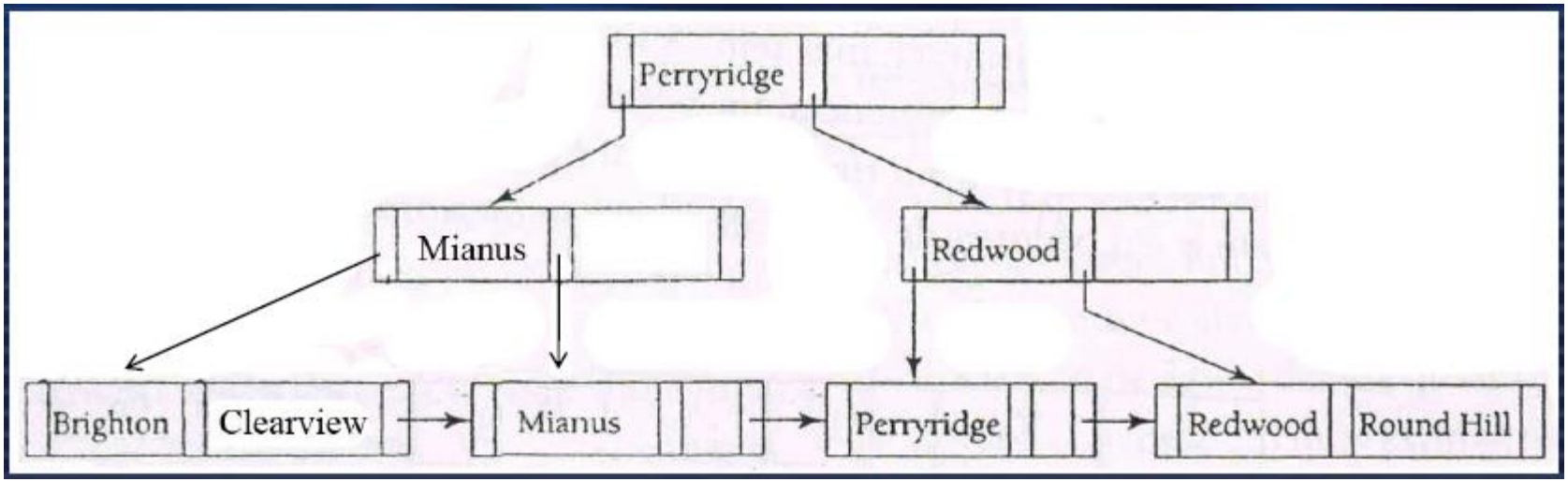
Inserção em Árvore B+

- Clearview deve aparecer no nó entre Brighton e Downtown
- Cria-se um novo nó para Downtown
- Downtown é inserido no nó pai e Mianus é deslocado para direita



Exclusão em Árvore B+

- Excluir o registro Downtown
- Excluir Downtown do nó folha
- Excluir Downtown do nó pai
- Mover Mianus e atualizar ponteiros



Funções de Hashing

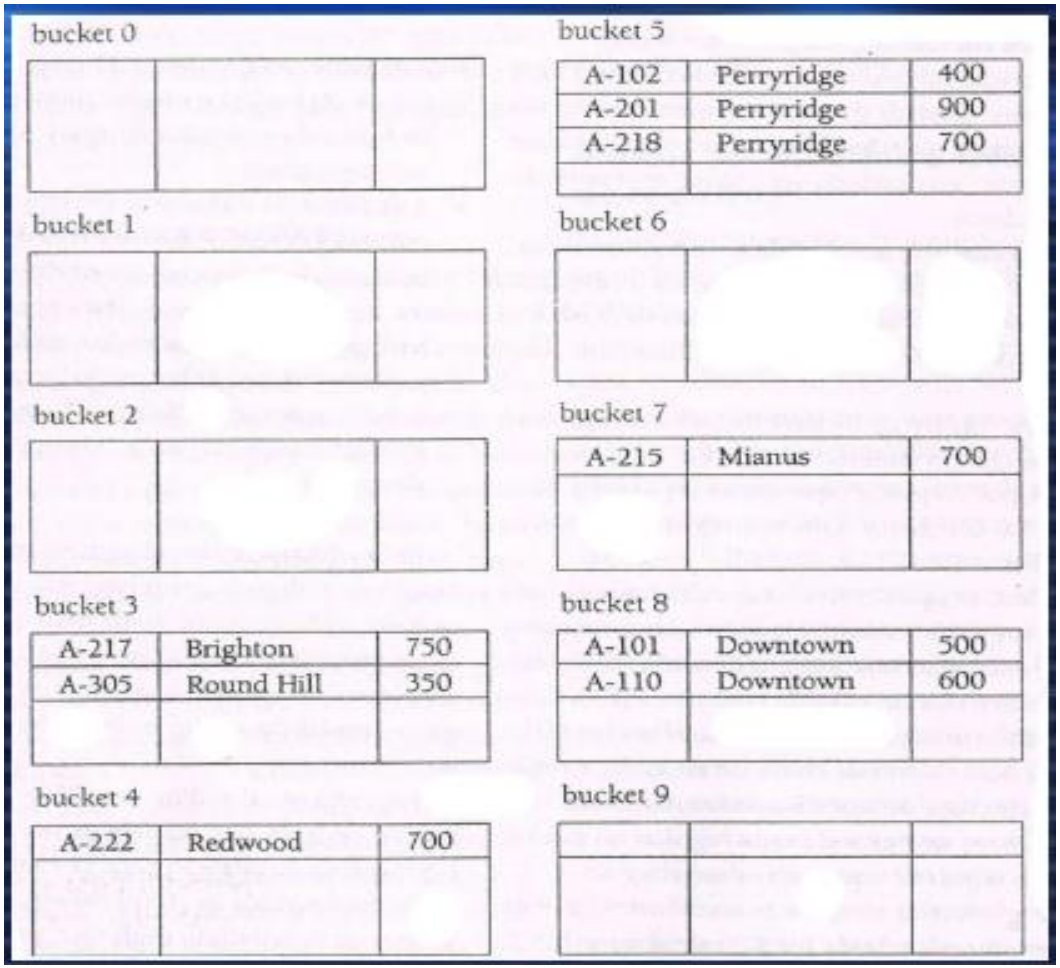
- Uma desvantagem dos índices é o consumo de espaço em disco, tempo para manutenção e pesquisa.
- A função de hashing trabalha com um cálculo matemático baseado em todos os valores de chave de busca (K), mais todos os endereços de buckets (B).
- Representado por h=f(K,B)
- O valor retornado pela função é o endereço físico do registro na tabela. Com isto fazemos apenas uma leitura.
- Cada fabricante tem em sua função de hashing, um segredo industrial.
- Necessita um projeto cuidadoso, pois uma má função pode retornar o mesmo endereço para duas chaves diferentes.

Funções de Hashing

- Pesquisa, inserção e exclusão:
  - Necessitam calcular o valor de hashing.
- Pior função:
  - Mapeia todos os valores de chave de busca para o mesmo bucket.
- Função ideal:
  - Distribui as chaves armazenadas uniformemente por todos os buckets.

Funções de Hashing

- Organização de hash da Conta, com chave nome_agencia



bucket 0

bucket 1

bucket 2

bucket 3

A-217	Brighton	750
A-305	Round Hill	350

bucket 4

A-222	Redwood	700

bucket 5

A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700

bucket 6

bucket 7

A-215	Mianus	700

bucket 8

A-101	Downtown	500
A-110	Downtown	600

bucket 9

Funções de Hashing

- Hashing estático: a função de hashing trabalha com um número fixo de registros possíveis para a tabela, evitando com isto que duas chaves tenham o mesmo valor de hashing. Uma alteração na quantidade de registros pode causar uma necessidade de regeração da função.

- Hashing dinâmico: a função de hashing não depende da quantidade de registros da tabela, necessita de muito mais elaboração por parte do fabricante pois o range de registros possíveis em uma tabela deve ser levado em consideração.

Definição de índice no SQL

- Cria um índice que contém uma chave que permite registros duplicados.

  CREATE INDEX <NOME> ON <TABELA> (<ATRIBUTO CHAVE>) ;

- Exemplo:

  Create index ind_cliente_nom on cliente (nom_cliente);

Definição de índice no SQL

- Cria um índice que contém uma chave que não permite registros duplicados.

  CREATE UNIQUE INDEX <NOME> ON <TABELA> (<ATRIBUTO CHAVE>) ;

- Exemplo:

  Create unique index ind_itemloc on item_locacao (num_locacao, num_item);

Definição de índice no SQL

- Exclusão de um índice.

  DROP INDEX <NOME INDICE>;

- Exemplo:

  Drop index ind_itemloc;