Capítulo 3 - Armazenamento e Recuperação

Elisabete Ferreira

24 de abril de 2018

Mecanismos de armazenamento e recuperação

1. Estruturados em Logs
   - Índices Hash
   - SSTables e LMS-Trees
2. Orientadas à páginas
   - Árvores B
   - Outras estruturas de indexação
3. Otimizados para transações
   - Processamento de transações on-line (OLTP)
4. Otimizados para análise
   - Processamento de transações on-line (OLAP)

Estruturados em Logs

- Utilizado por muitos bancos de dados;
- Sequência de registros somente de anexação;
- Adições são inseridas ao final do arquivo;
- Buscas varrem todo o arquivo de dados do início ao fim.

Criação e alimentação de BD por chave-valor

```
#!/bin/bash
db_set () { echo "$1,$2" >> database }
db_get () {grep "^$1," database | sed -e "s/^$1,//" | tail -n 1}
```

==========

```
$ db_set 123456 '{"name":"London","attractions":["Big Ben","London Eye"]}'
$ db_set 42 '{"name":"San Francisco","attractions":["Golden Gate Bridge"]}'
$ db_get 42
{"name":"San Francisco","attractions":["Golden Gate Bridge"]}
```

Atualização e leitura de registro por chave-valor

```
$ db_set 42 '{"name":"San Francisco","attractions":["Exploratorium"]}'
$ db_get 42
{"name":"San Francisco","attractions":["Exploratorium"]}
```

==========

```
$ cat database
123456,{"name":"London","attractions":["Big Ben","London Eye"]}
42,{"name":"San Francisco","attractions":["Golden Gate Bridge"]}
42,{"name":"San Francisco","attractions":["Exploratorium"]}
```

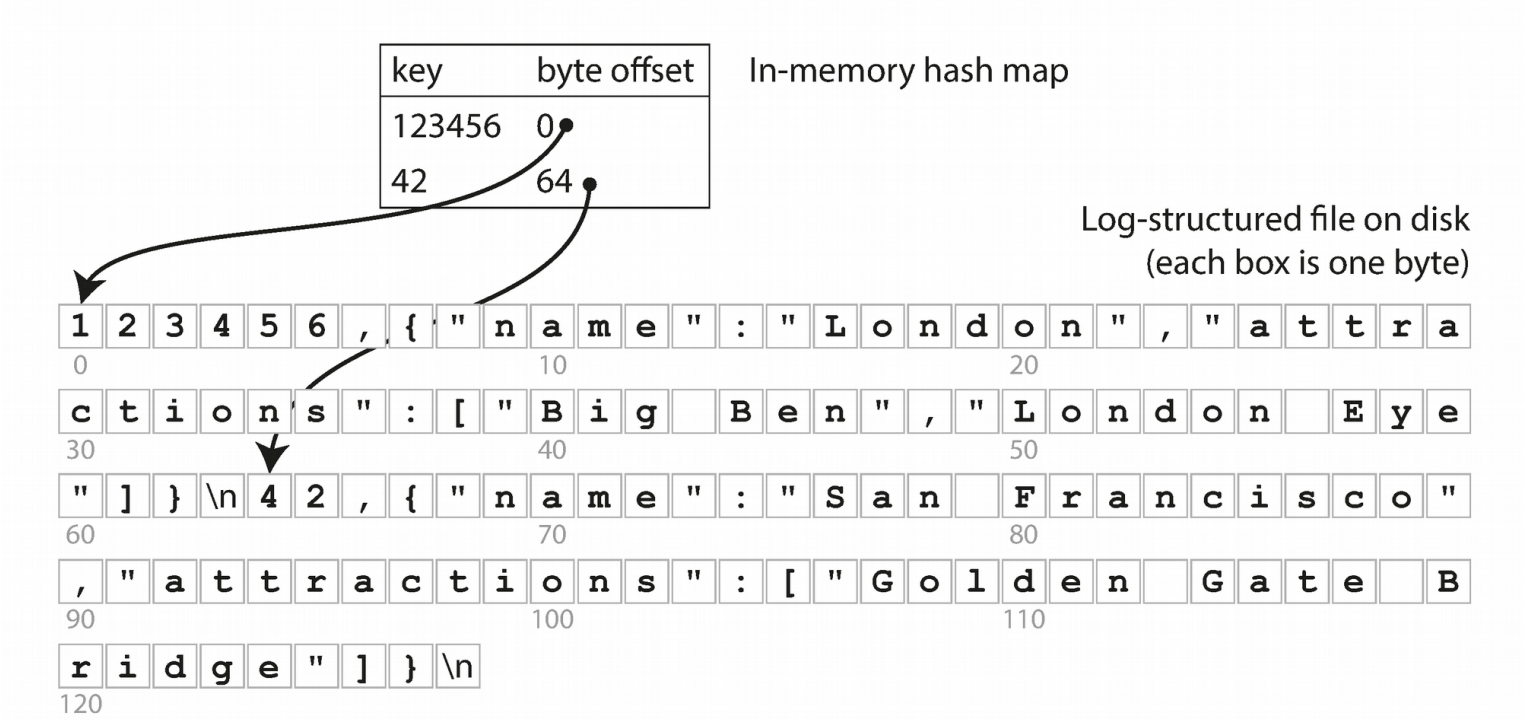
Índices Hash

Índices são Estruturas adicionais, derivadas de dados primários, utilizadas para localização eficiente de valores de chaves específicas;

- Índices bem escolhidos aceleram as buscas de informações;
- Qualquer tipo de índice retarda o processo de gravação de dados.

Índices Hash

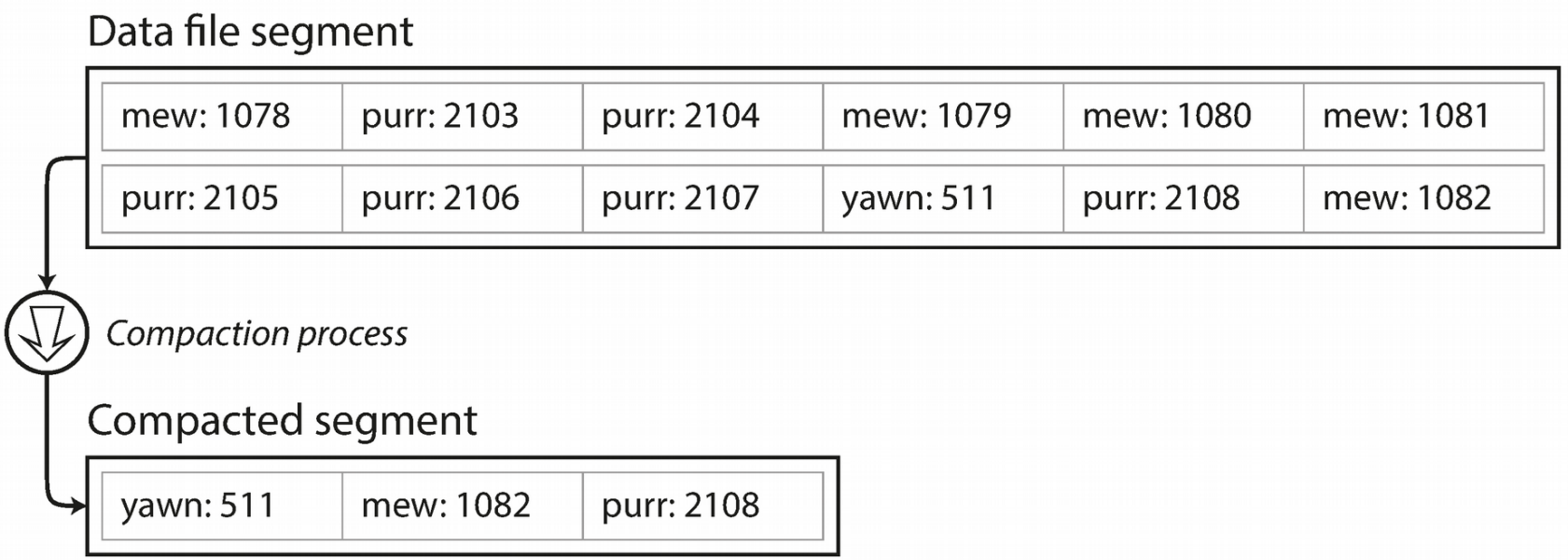
Semelhantes ao tipo dicionário dado, é implementado como um mapa de hash;



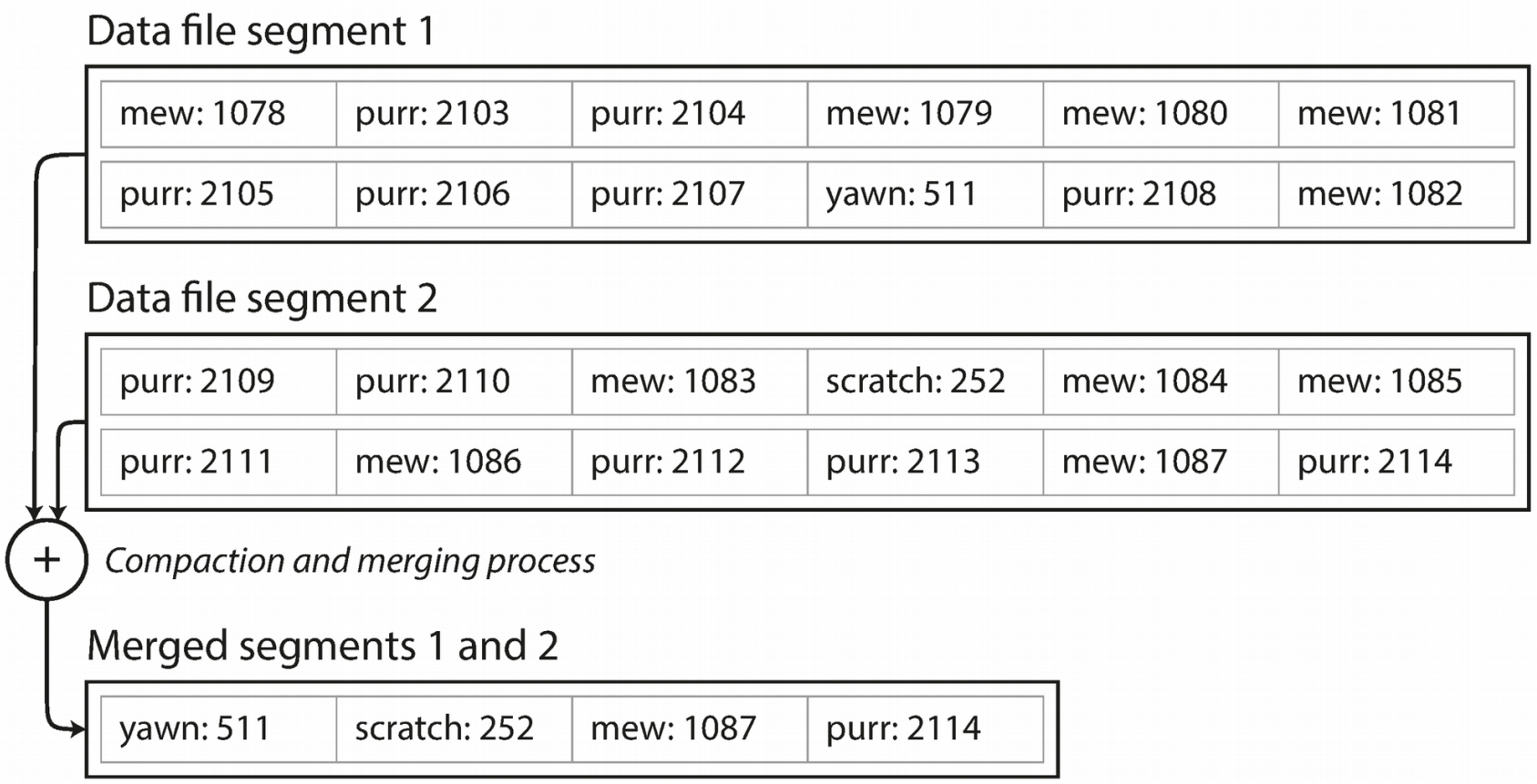
Índices de Hash

- Mantém um mapa de hash na memória onde cada chave é mapeada para um deslocamento de byte no arquivo de dados;
- Sempre que uma nova chave é adicionada ou atualizada, o mapa de hash é automaticamente atualizado;
- Utiliza o mapa de hash para fornecer o dado solicitado;

Índice Hash - segmentação de logs



Índice Hash - segmentação de logs



Índice Hash - segmentação de logs

- O melhor formato para geração de segmentos é o binário, onde é codificado o tamanho de uma string em bytes e após é seguido pela própria string;
- Para exclusão de uma chave e seu valor associado, anexa-se um registro de exclusão especial no arquivo de dados;
- No momento da mesclagem, a marca indica que a chave e seus demais valores devem ser excluídos;
- Pode-se acelera a recuperação pelo armazenamento instantâneo do mapa de hash de cada segmento no disco (Bitcask);
- Isso permite que o segmento seja carregado mais rapidamente na memória.

Índice Hash - segmentação de logs

- Utiliza checksum para garantir a integridade dos dados, o que permite que parte corrompidas sejam eliminadas do log;
- Os segmentos são apenas anexados e imutáveis, para que posam ser lidos simultaneamente por vários segmentos;
- Mesclar segmentos antigos evita o problema de fragmentação dos arquivos de dados.

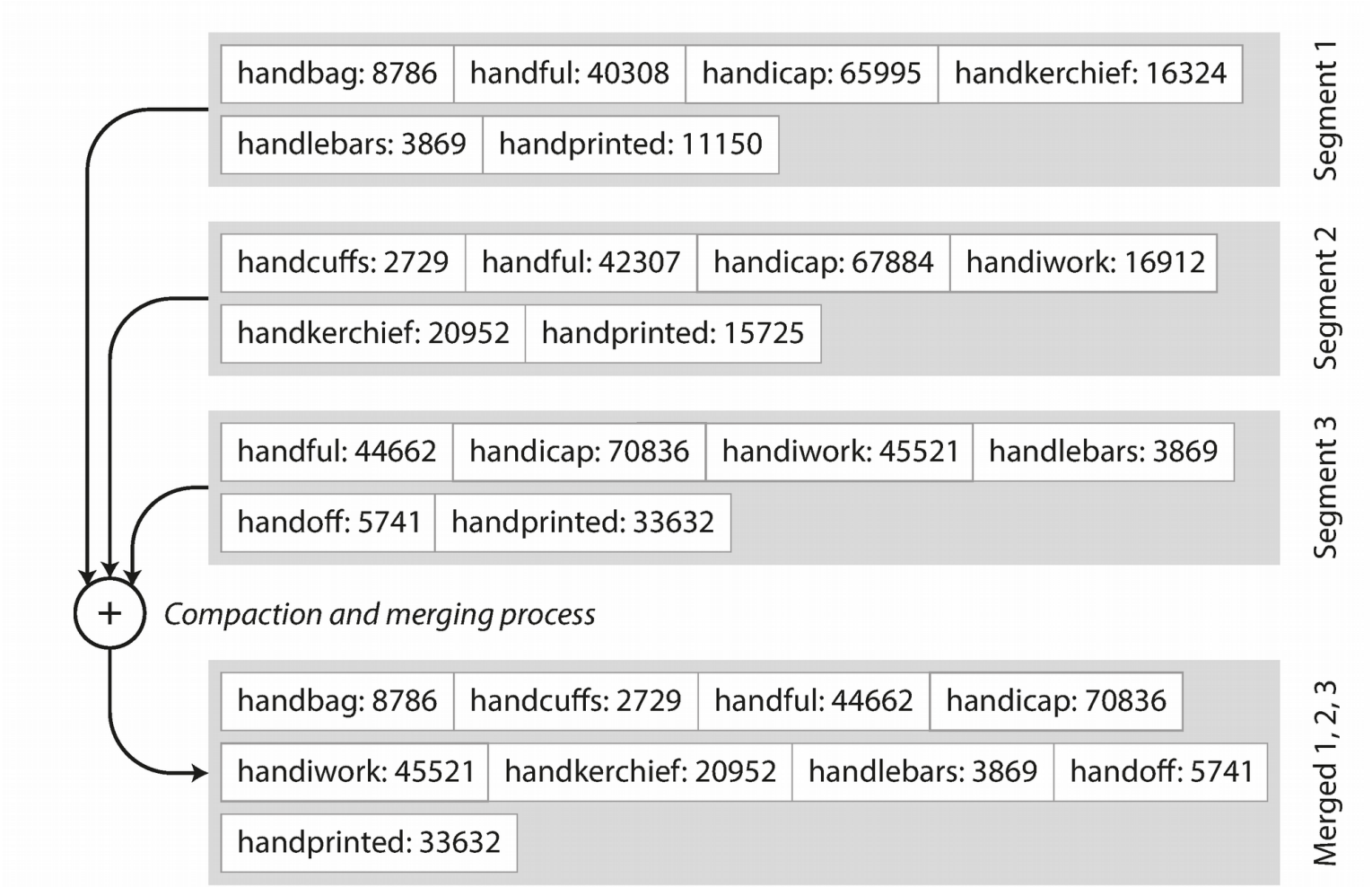
Índice Hash - limitações

- A tabela de hash precisa caber na memória;
- Manter um mapa de hash no disco compromete o desempenho;
- Podem ocorrer muitas colisões de hash.

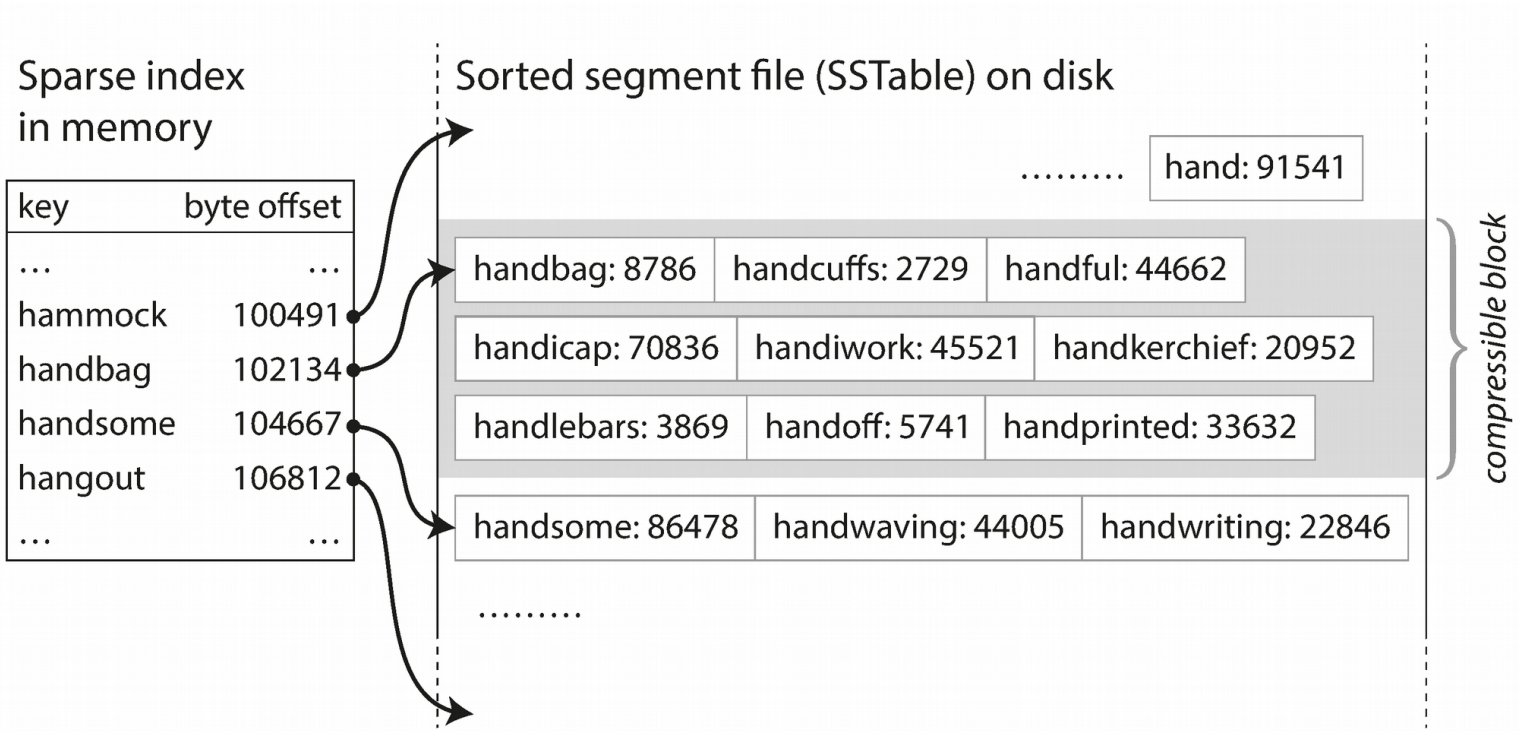
SSTable - sorted string table

- Considera que cada segmento de armazenamento estruturado em log é uma sequência de pares de valores-chave;
- Os valores posteriores têm precedência sobre os valores para a mesma chave no arquivo de log;
- A ordem dos valores-chave no arquivo não importa;
- A sequência de pares de valores-chave é classificada por chave;
- As chaves possuem apenas uma ocorrência por cada arquivo de segmento mesclado.

SSTable - sorted string table



SSTable - sorted string table



Construindo e mantendo SSTables

- Manter uma estrutura ordenada no disco é possível, mas mantê-la na memória é muito mais fácil;
- Para inserir chaves em qualquer ordem e lê-las novamente, classificadas, pode-se lançar mão de árvores binárias como AVL ou rubro-negra;
- O processo de gravação com o uso de árvores binárias é também conhecido como Memtable.

Construindo e mantendo SSTables

- Quando o memtable cresce, acima de alguns megabytes, é gravado em uma SSTable;
- Durante a gravação do memtale, as gravações podem continuar em uma nova instância memtable;
- Esporadicamente, é interessante mesclar e compactar, em segundo plano, para combinar arquivos de segmentos e descartar valores sobrescritos ou excluídos.

SSTables - limitações

- O memtbale permanece na memória;
- Se o BD falhar as gravações mais recentes são perdidas.

LSM-Trees - log-stuctured merge-tree

- São mecanismos de armazenamento, construídos em cima da lógica utilizada por log-structured filesystems, para mesclagem e compactação de arquivos:
  - Mantém um disco de log separado;
  - Restaura a memtable a partir deste registro;
  - Toda vez que a memtable é gravada em uma SSTable, este log é descartado;
  - trabalha com mesclagem e compactação de dados.

LSM-Trees - dicionário de termos

- O metódo LSM de armazenamento de dicionário de termos é utilizado pelo Lucene, um indexador de texto completo, usado pelo Elasticsearch e SolR
- Embora muito mais complexo do que um índice de valor-chave, os índices de textos completos funcionam de forma similar: dada uma palavra em uma consulta de pesquisa, procura localizar todos os documentos (páginas da Web, descrições de produtos etc.) que mencionam a palavra.

LSM-Trees - limitações e soluções

- É preciso varrer todos os segmentos até o mais antigo para identificar chaves que não existem no banco de dados, o que o torna lento;
  - Utiliza-se um filtro Bloom, que informa se uma chave não faz parte do banco de dados, o que evita leituras desnecessárias de disco na busca da mesma.

LSM-Trees - compactação por camada e nivelada

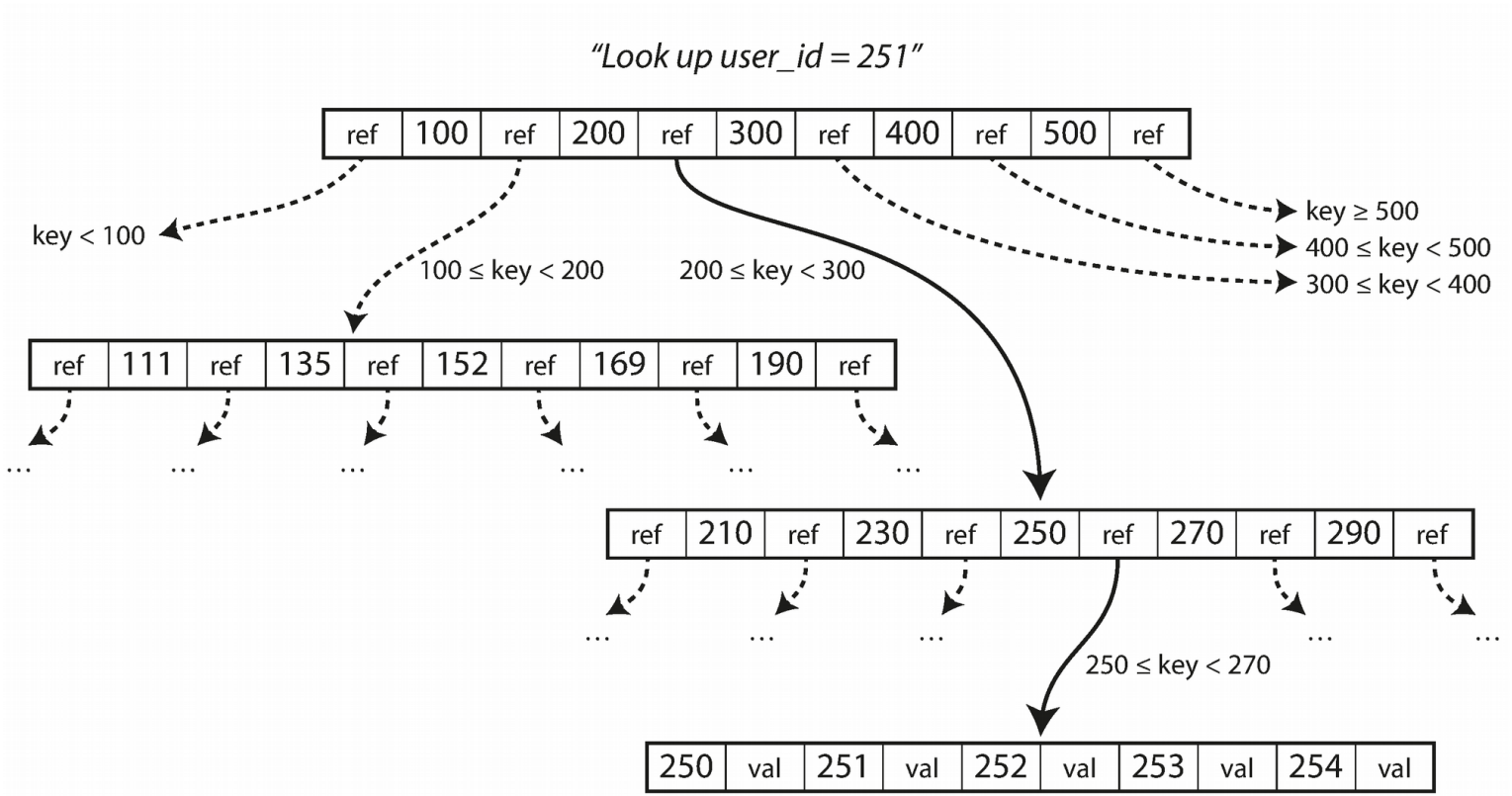
- São estratégias para definir a ordem e o tempo para mesclagem e compactação de SSTables;
  - Camadas - SSTables mais novos e menores são mesclados sucessivamente em SSTables mais antigos e maiores.
  - Nivelada - SSTables menores e os dados antigos são movidos para "níveis" separados, o que permite a compactação de forma incremental e com menos uso de espaço em disco.

Árvores B

É a estrutura de indexação mais utilizada em quase todos os bancos de dados relacionais e, em muitos não relacionais.

- Mantém pares de valores-chaves classificado por chaves;
- Divide o DB em blocos ou páginas de tamanho fixo;
- Lê e escreve uma página por vez;
- Cada página possui um endereço ou local;
- Permite que uma página se refira a outra em disco em vez de na memória (ponteiro).

Localizando um registro em uma árvore B

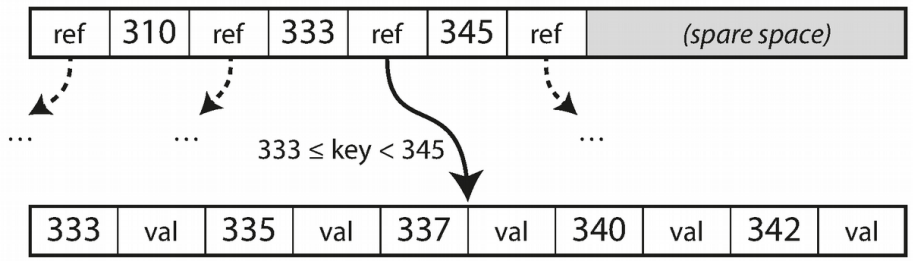


Árvores B

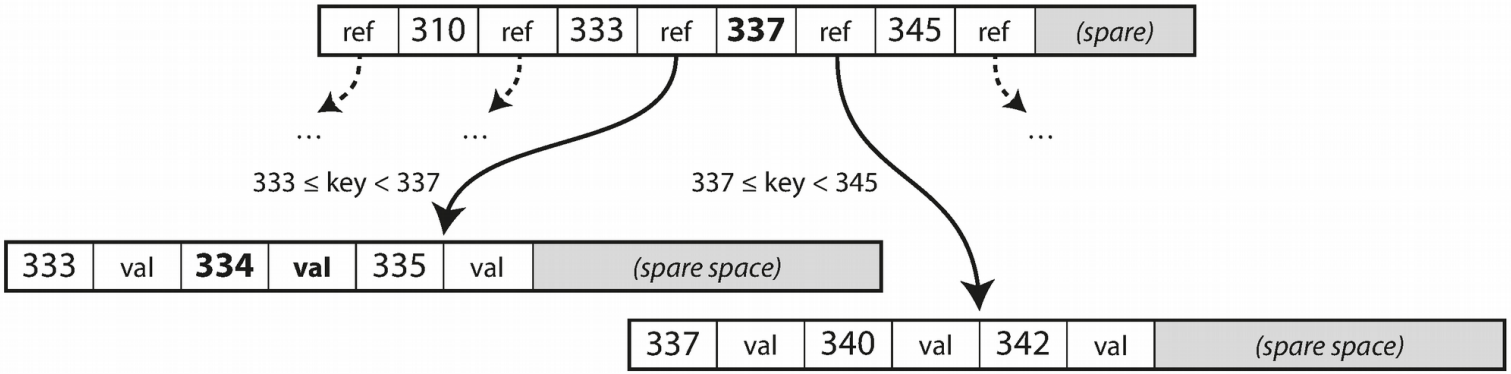
Atualizar e inserir valor em uma árvore B

- Atualizar.
  - Pesquisar a página de folhas que contêm a chave a ser atualizada;
  - Atualizar o valor da chave;
  - Gravar, novamente, a página no disco.
- Adicionar.
  - Encontrar a página cujo intervalo engloba a nova chave;
  - Adicionar a nova chave à página localizada;
  - Senão houver espaço livre suficiente na página encontrada, dividi-la em duas páginas;
  - atualizar a página pai para considerar a nova submissão dos intervalos de chaves.

Inserindo um valor em uma Árvore B



After adding key 334:



Segurança em uma árvore B

- Implementações em árvore B incluem uma estrutura adicional em disco: um log write-ahead (redo log);
- Toda inserção ou modificação na árvore é previamente escrita antes de ser aplicada às páginas da árvore.

Árvores B X Árvores LSM

- Árvores B:
  - São mais rápidas para leituras, pois cada chave existe em exatamente um lugar no índice;
  - O tempo de resposta de consultas orientadas à páginas são mais previsíveis que em LSM;
  - cada parte dos dados são gravados, pelo menos duas vezes: uma para o log de gravação e outra na árvore em si;
  - Há a sobrecarga de ter que escrever uma página inteira, mesmo quando apenas poucos bytes são alterados.
- Árvores LSM:
  - São mais rápidas para gravações, basta anexar a chave-valor ao arquivo de log;
  - Os arquivos compactados em LSM são menores que os das árvores B;
  - São mais lentas nas leituras pois precisam verificar várias estruturas de dados e SSTable em diferentes estágios de compactação;
  - O tempo de resposta de consultas em estruturas de log podem ser bastante altos.

- Índices secundários;
- Índices clusterizados: armazenam todos os dados de linha dentro do índice;
- Índices não clusterizados: armazenam referências aos dados dentro do índice;
- Índices multi-colunas:
  - Concatenados: combinam vários campos em uma chave, anexando uma coluna a outra;
  - Multidimensionais: para consultar várias colunas ao mesmo tempo.
- Índices difusos;

Exemplo de índice multidimensional

```
SELECT *  FROM restaurants
WHERE
        latitude > 51.4946 AND latitude < 51.5079
AND
        longitude > -0.1162 AND longitude < -0.1004;
```

Processamento de transações on-line (OLTP)

Utilizados para processamento de dados corporativos.

- Controle de Estoque;
- Controle de Vendas;
- Controle de Pessoal;
- Controle de Compras;
- Controle de Fornecedores;

Processamento analítico (OLAP)

Utilizados para processamento de dados analíticos que apoiam decisões gerenciais (Business Inteligence).

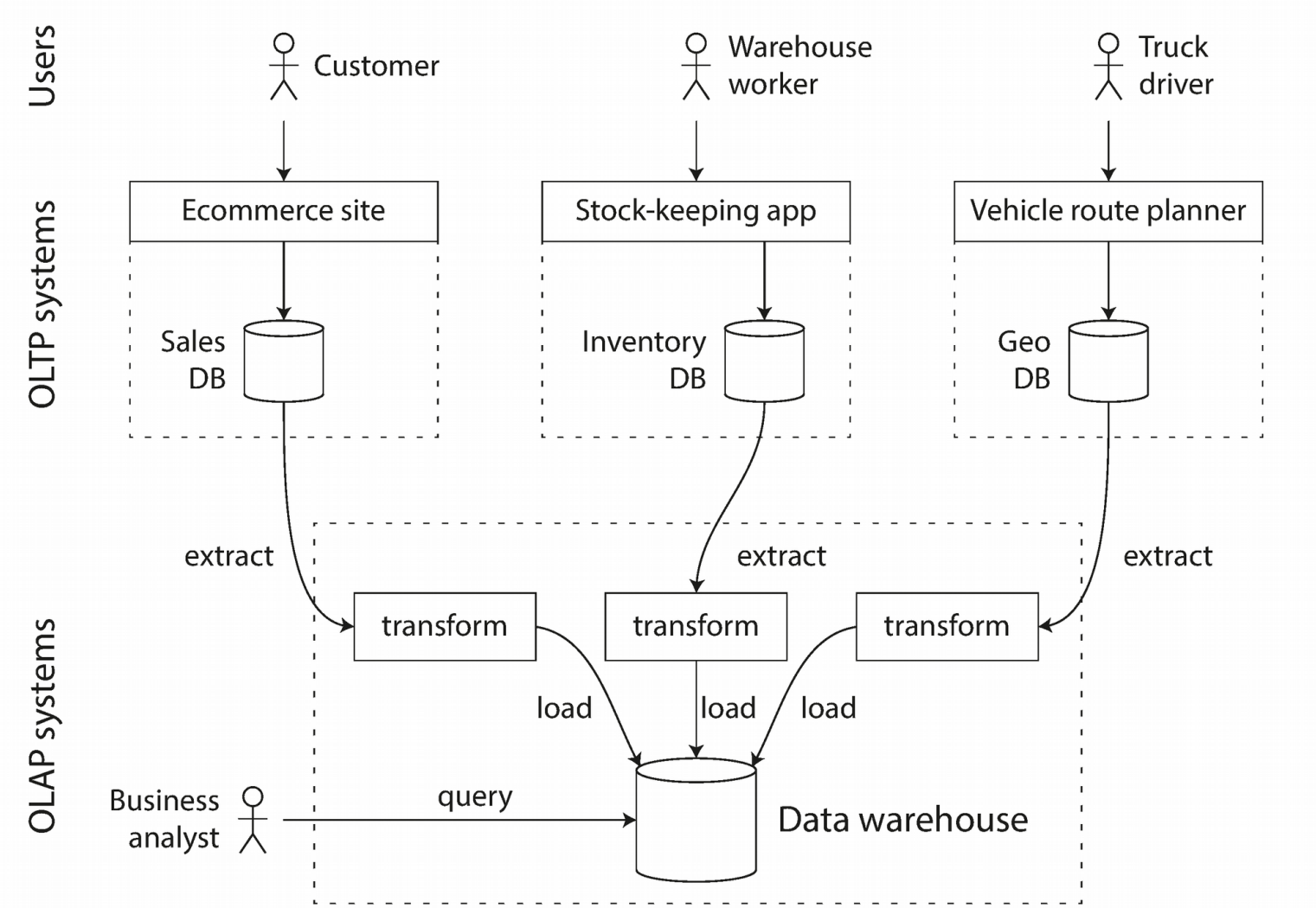
- Análise de Dados;
- Cálculo estatísticos;
- Prospecção de Hábitos de Consumo;
- Avaliação dos retornos de promoções;
- Gerenciamento de Pessoal;

Tabela 1. Comparando características do processamento de transações versus sistemas analíticos

Propriedade	**Sistemas de processamento de transações (OLTP)**	**Sistemas analíticos (OLAP)**
Padrão principal de leitura	Pequeno número de registros por consulta, buscado pela chave.	Agregar em um grande número de registros.
Padrão principal de escrita	Gravações de acesso aleatório e baixa latência a partir da entrada do usuário.	Importação em massa (ETL) ou fluxo de eventos.
Usado principalmente por	Usuário final/ cliente, via aplicativo da web	Analise interna, para apoio à decisão.
Quais dados representam	Último estado dos dados (ponto atual no tempo).	Histórico dos eventos ocorridos ao longo do tempo.
Tamanho do conjunto de dados	Gigabytes para terabytes.	Terabytes para petabytes.

Data warehousing)

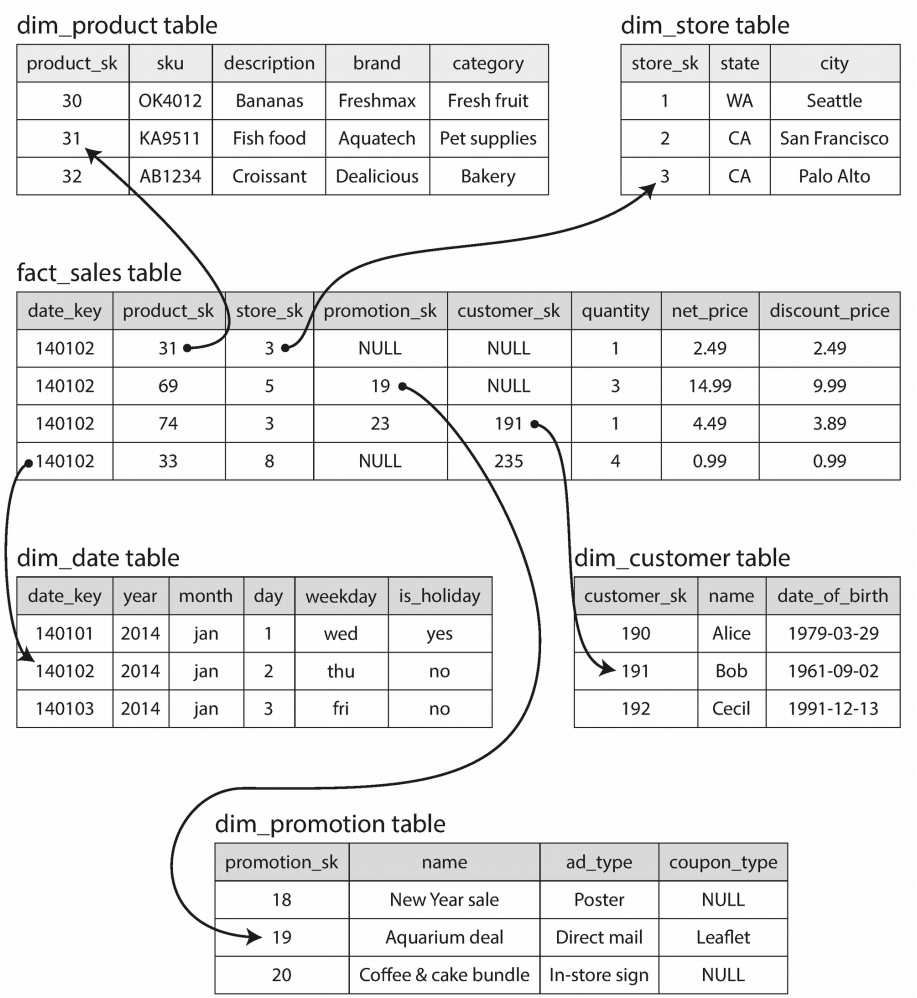
É um banco de dados que contém uma cópia de somente leitura de dados contidos em vários sistemas OLTP de uma empresa, obtidos pelo processo Transform - Load - Extract (ETL).


Users
OLTP systems
OLAP systems
Customer
Warehouse worker
Truck driver
Ecommerce site
Stock-keeping app
Vehicle route planner
Sales DB
Inventory DB
Geo DB
extract
extract
extract
transform
transform
transform
load
load
load
Business analyst
query
Data warehouse

Esquemas para análise

- Estrela: Os relacionamentos da tabela são visualizados, a tabela de fatos fica no meio, cercada por suas tabelas de dimensões.
- Flocos de Neve: As dimensões são subdivididas em subdimensões.

Esquemas para análise - Estrela



Armazenamento orientado por colunas

fact_sales table

date_key	product_sk	store_sk	promotion_sk	customer_sk	quantity	net_price	discount_price
140102	69	4	NULL	NULL	1	13.99	13.99
140102	69	5	19	NULL	3	14.99	9.99
140102	69	5	NULL	191	1	14.99	14.99
140102	74	3	23	202	5	0.99	0.89
140103	31	2	NULL	NULL	1	2.49	2.49
140103	31	3	NULL	NULL	3	14.99	9.99
140103	31	3	21	123	1	49.99	39.99
140103	31	8	NULL	233	1	0.99	0.99

Columnar storage layout:

date_key file contents:	140102, 140102, 140102, 140102, 140103, 140103, 140103, 140103
product_sk file contents:	69, 69, 69, 74, 31, 31, 31, 31
store_sk file contents:	4, 5, 5, 3, 2, 3, 3, 8
promotion_sk file contents:	NULL, 19, NULL, 23, NULL, NULL, 21, NULL
customer_sk file contents:	NULL, NULL, 191, 202, NULL, NULL, 123, 233
quantity file contents:	1, 3, 1, 5, 1, 3, 1, 1
net_price file contents:	13.99, 14.99, 14.99, 0.99, 2.49, 14.99, 49.99, 0.99
discount_price file contents:	13.99, 9.99, 14.99, 0.89, 2.49, 9.99, 39.99, 0.99

Compressão por coluna)

Column values:

product_sk:	69	69	69	69	74	31	31	31	31	29	30	30	31	31	31	68	69	69

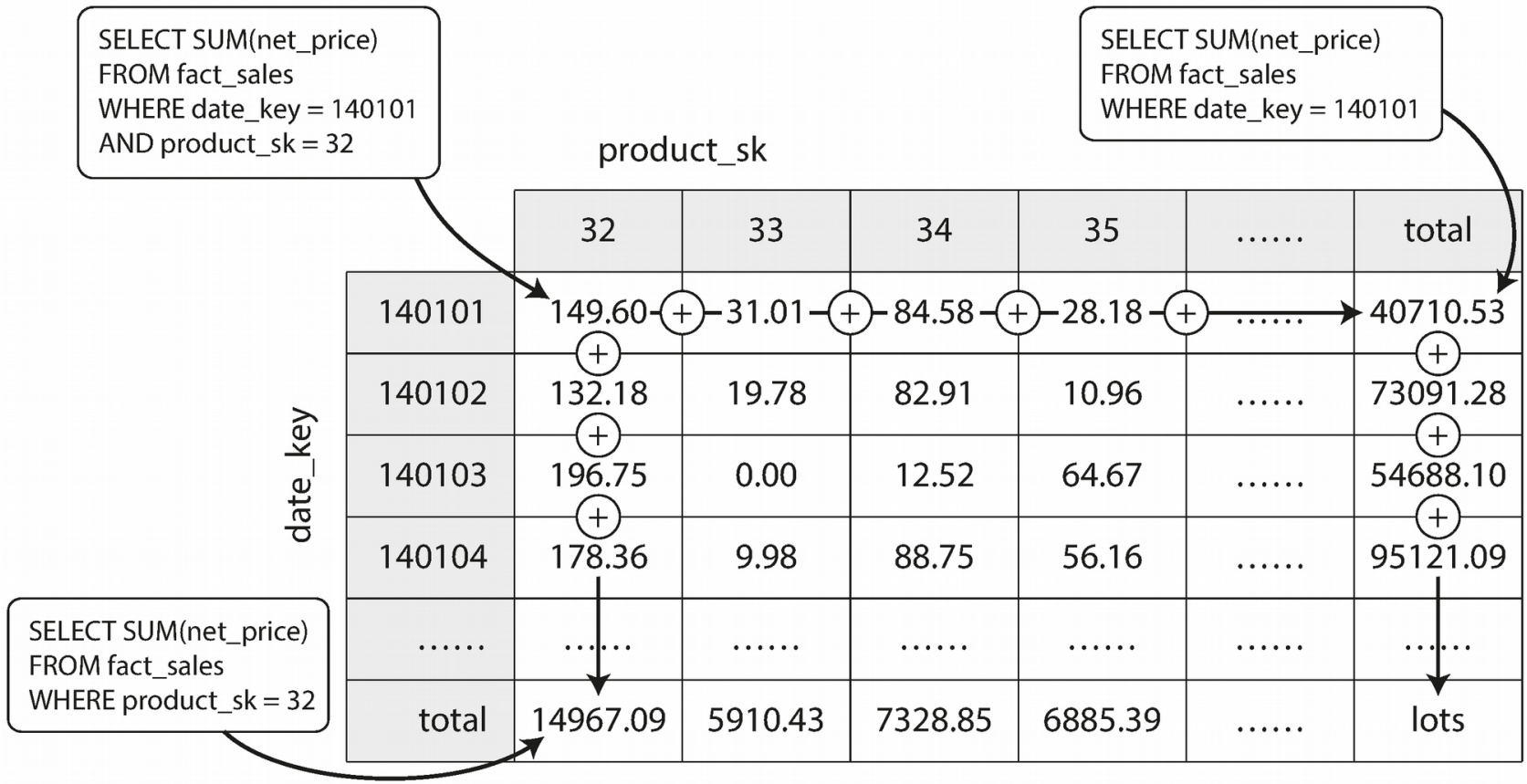
Bitmap for each possible value:

product_sk = 29:	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
product_sk = 30:	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
product_sk = 31:	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0	0
product_sk = 68:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
product_sk = 69:	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
product_sk = 74:	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Run-length encoding:

product_sk = 29:	9, 1	(9 zeros, 1 one, rest zeros)
product_sk = 30:	10, 2	(10 zeros, 2 ones, rest zeros)
product_sk = 31:	5, 4, 3, 3	(5 zeros, 4 ones, 3 zeros, 3 ones, rest zeros)
product_sk = 68:	15, 1	(15 zeros, 1 one, rest zeros)
product_sk = 69:	0, 4, 12, 2	(0 zeros, 4 ones, 12 zeros, 2 ones)
product_sk = 74:	4, 1	(4 zeros, 1 one, rest zeros)

Cubos de dados e visões materializadas



Exercícios

- Como o problema de falta de memória é resolvido em LSMs?
- O que é uma tabela de fatos em Data warehouse?

References

Martin Kleppmann (2017)

Designing Data-Intensive Applications

O'Reilly Media, Inc, 69 – 103.

Muito Obrigada!