Arquitetura de SGBD Relacionais
— Recordação e Motivação para Bases de Dados II —

Caetano Traina Jr.

Grupo de Bases de Dados e Imagens
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo - São Carlos
caetano@icmc.usp.br

27 de fevereiro de 2013
São Carlos, SP - Brasil

Breve recordação de conceitos e motivação para a disciplina de

Bases de Dados II - Arquitetura de Sistemas Gerenciadores de Bases Dados Relacionais.

Outline

Objetivos

Fornecer conceitos, técnicas e características mais avançadas dos sistemas gerenciadores de banco de dados, complementando o conteúdo ministrado nas disciplinas básicas de banco de dados. Oferecer o conhecimento necessário ao profissional que se envolva em atividade de administração e gerenciamento de banco de dados.

Programa

- Processamento e otimização de consultas.
- Ajuste fino de desempenho.
- Processamento de transações e controle de concorrência.
- Recuperação de falhas.
- Bancos de dados distribuídos.
- Bancos de dados de objetos.

Avaliação

Serão atribuídas notas aos trabalhos práticos e às provas realizadas em sala de aula. A nota final será calculada pela média ponderada dessas notas segundo a fórmula:

$$NF = (3P_1 + 3P_2 + 2T_1 + 2T_2)/10$$

Onde NF é a Nota Final, P_i são as notas obtidas nas provas e T_i são as notas obtidas nos trabalhos

Norma de Recuperação

Realização: Até a primeira semana de aulas do semestre posterior

Critério de Aprovação: $N_F + (N_{Rec}/2,5)$, se $N_{Rec} \geq 7,5$; ou
$max\{N_F, N_{Rec}\}$, se $Nrec < 5,0$; ou
5,0, se $5,0 \leq N_{Rec} < 7,5$

Onde: N_F é a Nota Final durante o semestre e
N_{Rec} é a Nota obtida na prova de Recuperação.

Datas das Provas

- Primeira Prova: 02 de maio
- Segunda Prova: 20 de junho
- Substitutiva: 27 de junho

Bibliografia

Básica

- Elmasri, R.; Navathe, S.B. Fundamentals of Database Systems, 4th edition. Pearson/Addison Wesley 2004.
- Silberschat, A.; Korth, H.f.; Sudarshan, S. - Sistemas de Banco de Dados, 3a edição. Editora Makron Books, 1999.
- Christopher J. Date, Introdução a Sistemas de Bancos de Dados. Tradução da 7a edição americana Editora Campus, 2000.
- Heuser, C.A. Projeto de Banco de Dados. Sagra Luzzatto, 2001.
- Ramakrishnan, R.; Gehrke, J. Database Management Systems, 3rd edition. McGraw-Hill, 2003.

Bibliografia

Específica da disciplina:

- O'Neil, P.; O'Neil, E. Database: Principles, Programming, and Performance, 2nd edition, Morgan Kaufmann Publishers, 2001.
- Hector García-Molina, Jeffrey D. Ulhman, Jennifer Widow, Database Systens: The Complete Book: Prentice Hall, 2000.
- N. Bruno, Automated Physical Databases Design and tuning. Boca Raton, FL: CRC Press, 2011.
- Christopher J. Date, SQL and Relational Theory - How to Write Accurate SQL Code, O'Reilly Media, 2009.
- G. Graefe, "Query Evaluation Techniques for Large Databases,"ACM Computing Surveys, Vol. 25, No. 2, June 1993, pp. 98-182.
- G. Graefe, "Modern B-Tree Techniques,"Foundations and Trends in Databases, Vol. 3, No. 4, pp. 203-402.

Conceitos Básicos Necessários

- Um bom conhecimento de Bases de Dados I
  - Modelagem de Dados,
  - Modelo relacional,
  - SQL;
- ☞ Estruturas de Dados
- Formas de interação do Profissional com o SGBD.

Formas de interação do Profissional com o SGBD

- Programadores – usam as estruturas e os dados armazenados para codificar os aplicativos de uma organização;
- Projetistas – realizam a modelagem de dados dos aplicativos e mantém o modelo de dados da organização atualizado;
- Administradores de Dados – Controlam o SGBD de uma organização;
- Fabricantes de SGBD – codificam o "*core engine* do SGBD" e os utilitários associados.

Nosso interesse nesta disciplina são os **Administradores de Dados!**

Administradores de Dados

Para exercer as atividades de um Administrador de Dados, o profissional precisa de um entendimento amplo e profundo:

- Da aplicação,
- Do sistema operacional,
- Dos princípios físicos do hardware, e
- De como o gerenciador é construído.

Espera-se que o Administrador de Dados realize o "Ajuste Fino" do SGBD para a organização.
Nesta matéria, será estudado como os SGBDs são construídos —
A Arquitetura dos SGBD Relacionais.

Estruturas de Dados

Conhecimentos das materias básicas de Estruturas de Dados que serão necessários:

- Tipos de Dados;
- Ordenação:
  - Em memória,
  - Em disco;
- Estruturas de Dados:
  - *B-tree* (B^+*-tree*, B^**-tree*)
  - Arquivos Invertidos,
  - *Hash*,
  - *Bitmap*,
  - Índices Compostos;
- Complexidade de algoritmos:

☞ Ordenação, operações com conjuntos, memória externa;

- Recursos:

☞ Memória, acesso a disco, tempo de processamento.

Termos importantes

Alguns termos associados que são usados em Bases de Dados, que serão necessários:

- *Indexed Sequential Access Method* - ISAM;
- Índices Densos $\times$ Índices Esparços;
- Índices primários e Secundários (Índices *Clustered* e *não-Clustered*);
- Índices `Unique` e índices com chaves duplicadas;
- *Row-Id*;
- Taxa de ocupação das páginas de disco.

Motivação

- Para um Administrador de Dados, bem como para um Analista ou Programador que faça uso avançado de um SGBD, é importante conhecer como ele funciona;
- Em situações críticas, somente com um bom conhecimento do SGBD se pode obter um bom desempenho do SGBD.

Motivação

Os paradigmas Imperativo × Relacional

- Linguagens de Programação em geral utilizam o paradigma Imperativo;

☞ O programador indica as ações a serem executadas codificando "como fazer".

- Linguagens de Consulta (leia-se SQL) utiliza o paradigma Declarativo;

☞ O programador indica as ações a serem executadas codificando "o que fazer";

☞ e o SGBD fica livre para escolher as opções que ele tem de como fazer.

Com isso, o SGBD tem liberdade para escolher uma boa opção de **como fazer**, e o programador não precisa se preocupar com os detalhes de qual seria uma boa execução d**o que** ele quer fazer.

Motivação

Os paradigmas Imperativo × Relacional

Mas como??

Se o SGBD é livre para escolher a melhor maneira de como executar um comando em SQL, por que o programador precisa conhecer detalhes do funcionamento do SGBD?

☞ Porque o SGBD escolhe a melhor opção disponível, mas o DBA deve criar opções de escolha.

- Além disso, o SGBD sempre tem apenas uma consulta para analisar, e os programadores/DBA têm um conhecimento mais abrangente dos dados e da carga de consulta em geral.

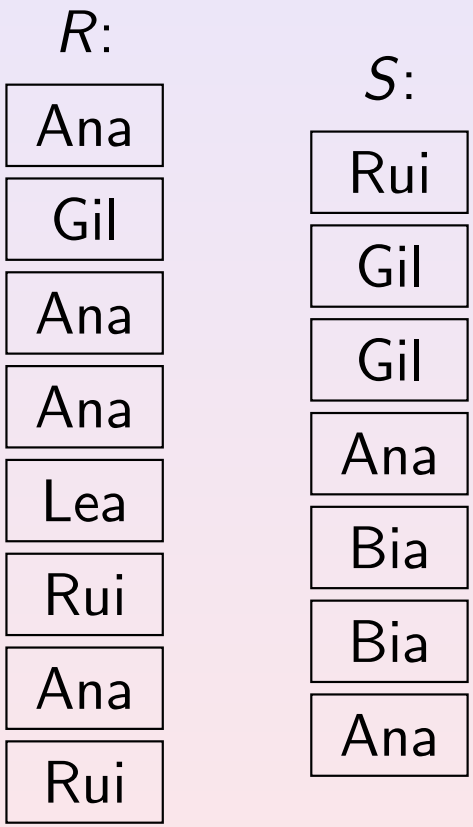
O SGBD sempre busca **"otimizar"** cada consulta individualmente, baseado apenas na consulta e nas informações locais que ele tem sobre os dados armazendos e o estado da máquina no instante de execução da consulta.

Exemplo de Motivação

Vamos usar um exemplo de motivação.
Suponha que temos duas coleções de elementos, digamos, de nomes de pessoas R e S, e queremos saber o número de vezes pares com elementos iguais podemos obter, pegando um elemento de cada coleção.
Suponha que exista uma rotina

```
int quantos=ContaPares(R, S);
```

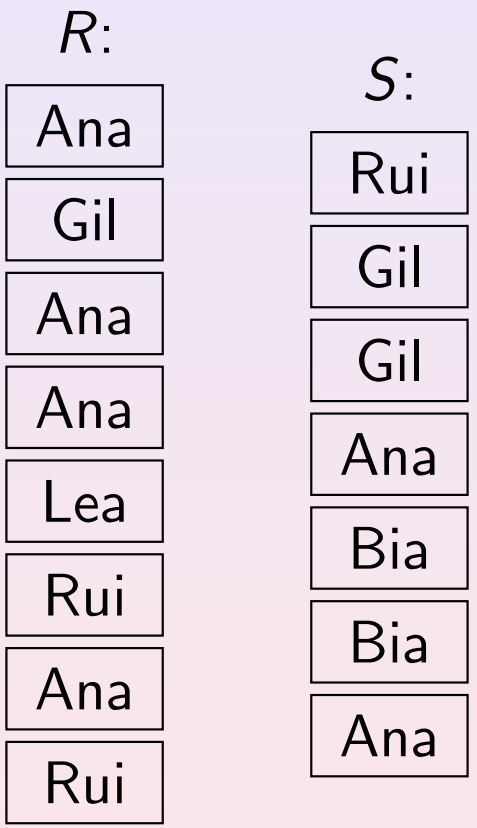
que obtem esse total.



Exemplo de Motivação

Existem muitas maneiras de implementar essa rotina.
Uma implementação simples é parear exastivamente todas as possibilidades e contar o total de vezes que se forma um par de elementos iguais:

```
int ContaPares1 (RId[] R, RId[] S) {
   int c = 0;
   foreach (RId r in R)
      foreach (RId s in S)
         if (Compare(r, s, '=')) c++;
   return c;
}
```



Exemplo de Motivação

- Devemos sempre avaliar quais recursos uma rotina consome:
  - Tempo total?
  - Quantidade de Memória?
  - Quantidade de Comparações?
  - Número de acessos a disco?
  - ...

Exemplo de Motivação

- A rotina `ContaPares1` requer:

```
int ContaPares1 (RId[] R, RId[] S) {
   int c = 0;
   foreach (RId r in R)
      foreach (RId s in S)
         if (Compare(r, s, '=')) c++;
   return c;
}
```

- Tempo total: $O(|R| \cdot |S|)$
- Quantidade de Memória: 0
- Quantidade de Comparações: $O(|R| \cdot |S|)$

Exemplo de Motivação

- No entanto, pode-se melhorar essa rotina.
- Por exemplo, poderia ser criada uma estrutura *hash* para cada valor distinto de R, armazenando junto a cada valor sua multiplicidade:

```
int ContaPares2 (RId[] R, RId[] S) {
   // A tabela hash é um dicionário <elemento, quantos>
   Dictionary<RId, int> ht = new Dictionary<RId,int>();
   foreach (RId r in R)
      if (ht.ContainsKey(r)) ht[r]++;
        else ht.Add(r, 1);
   // Executa a contagem
   int c = 0;
   foreach (RId s in S)
      if (ht.ContainsKey(s)) c += ht[s];
   return c;
}
```

Exemplo de Motivação

- A rotina `ContaPares2` requer:
  - Tempo total: $O(\max(|R|, |S|))$
  - Quantidade de Memória: $O(|R|)$
  - Quantidade de Comparações: $O(\log_2 |R| \cdot |S|)$

Como bons banqueiros de dados, vamos organizar os dados numa relação:

Algoritmo	Tempo	Memória	Comparações
Força Bruta	$O(\|R\| \cdot \|S\|)$	0	$O(\|R\| \cdot \|S\|)$
Hash	$O(\max(\|R\|, \|S\|))$	$O(\|R\|)$	$O(\log_2 \|R\| \cdot \|S\|)$

Tabela: Custo dos algoritmos(2)

Exemplo de Motivação

- Outra alternativa, seria ordenar as duas coleções.
- A idéia aqui é que podemos percorrer as duas coleções ordenadas de maneira sincronizada.

```
int ContaPares3 (RId[] R, RId[] S) {
   int c = 0;
   int pR = 0, pS = 0;
   Sort(R); Sort(S);
   while (pR < |R| && pS < |S|)
      if (R[pR] < S[pS]) pR++;
      else if (R[pR] > S[pS]) pS++;
      else { // R[pR]==S[pS]
         int cR = 1, cS = 1;
         while (pR + cR < |R| && R[pR] == R[pR + cR]) cR++;
         while (pS + cS < |S| && S[pS] == S[pS + cS]) cS++;
         c += cR * cS;
         pR += cR; pS += cS;
      }
   return c;
}
```

Exemplo de Motivação

- A rotina `ContaPares3` requer:
  - Tempo total:

$$
\begin{aligned}
O(|R| \cdot \log_2 |R| + |S| \cdot \log_2 |S| + |R| + |S|) &= \\
O((|R| + 1) \cdot \log_2 |R| + (|S| + 1) \cdot \log_2 |S|) &= \\
O(|R| \cdot \log_2 |R| + |S| \cdot \log_2 |S|)
\end{aligned}
$$

  - Quantidade de Memória: $O(\max(|R|, |S|))$
  - Quantidade de Comparações: $O(|R| \cdot \log_2 |R| + O(|S| \cdot \log_2 |S|))$

Nossa relação fica então:

Algoritmo	Tempo	Memória	Comparações
Força Bruta	$O(\|R\| \cdot \|S\|)$	0	$O(\|R\| \cdot \|S\|)$
Hash	$O(\max(\|R\|, \|S\|))$	$O(\|R\|)$	$O(log_2\|R\| \cdot \|S\|)$
Sort-Merge	$O(\|R\| \cdot \log_2 \|R\| + \|S\| \cdot \log_2 \|S\|)$	$O(\max(\|R\|, \|S\|))$	$O(\|R\| \cdot \log_2 \|R\| + \|S\| \cdot \log_2 \|S\|)$

Tabela: Custo dos algoritmos(3)

Exemplo de Motivação

- Poderia acontecer que os dados já estivessem ordenados. Assim, não seria necessário o passo de ordenação:

```
int ContaPares4 (RId[] R, RId[] S) {
   int c = 0;
   int pR = 0, pS = 0;
   while (pR < |R| && pS < |S|)
      if (R[pR] < S[pS]) pR++;
      else if (R[pR] > S[pS]) pS++;
      else { // R[pR]==S[pS]
         int cR = 1, cS = 1;
         while (pR + cR < |R| && R[pR] == R[pR + cR]) cR++;
         while (pS + cS < |S| && S[pS] == S[pS + cS]) cS++;
         c += cR * cS;
         pR += cR; pS += cS;
      }
   return c;
}
```

Exemplo de Motivação

Nossa relação fica então:

Algoritmo	Tempo	Memória	Comparações
Força Bruta	$O(\|R\| \cdot \|S\|)$	0	$O(\|R\| \cdot \|S\|)$
Hash	$O(\max(\|R\|, \|S\|))$	$O(\|R\|)$	$O(log_2\|R\| \cdot \|S\|)$
Sort-Merge	$O(\|R\| \cdot \log_2 \|R\| + \|S\| \cdot \log_2 \|S\|)$	$O(\max(\|R\|, \|S\|))$	$O(\|R\| \cdot \log_2 \|R\| + \|S\| \cdot \log_2 \|S\|)$
Merge	$O(\|R\| + \|S\|)$	$O(1)$	$O(\|R\| + \|S\|)$

Tabela: Custo dos algoritmos(4)

Exemplo de Motivação

Note-se que o algoritmo *Sort* requer que os dados estejam previamente ordenados. Assim, nossa coleção de algoritmos requer manter o conhecimento sobre determinadas propriedades dos dados (no caso, se eles estão ou não ordenados).

Algoritmo	Tempo	Memória	Comparações	Ordenado
Força Bruta	$O(\|R\| \cdot \|S\|)$	0	$O(\|R\| \cdot \|S\|)$	–, –
Hash	$O(\max(\|R\|, \|S\|))$	$O(\|R\|)$	$O(log_2\|R\| \cdot \|S\|)$	–, –
Sort-Merge	$O(\|R\| \cdot \log_2 \|R\| + \|S\| \cdot \log_2 \|S\|)$	$O(\max(\|R\|, \|S\|))$	$O(\|R\| \cdot \log_2 \|R\| + \|S\| \cdot \log_2 \|S\|)$	–, – /
Merge	$O(\|R\| + \|S\|)$	$O(1)$	$O(\|R\| + \|S\|)$	S, S

Tabela: Custo dos algoritmos e propriedades

Exemplo de Motivação

- Considerando que ao menos uma das coleções já está ordenada, é possível conceber um outro algoritmo:
  - Suponha que S está ordenado;
  - Percorre R, e procura cada elemento `R[pR]` em S, usando busca binária para achar a sua primeira ocorrência;
  - Se existir um par, percorre os elementos seguintes em S para saber quantos pares devem ser contados.

```
int ContaPares5 (RId[] R, RId[] S) {
   int c = 0;
   foreach (RId r in R) {
      int pS = BuscaBinaria(S, r);
      while (pS < |S| && S[pS] == r) {
         c++;
         pS++;
      }
   return c;
}
```

Exemplo de Motivação

Nossa relação de custos fica então:

Algoritmo	Tempo	Memória	Comparações	Ordenado
Força Bruta	$O(\|R\| \cdot \|S\|)$	0	$O(\|R\| \cdot \|S\|)$	–, –
Hash	$O(\max(\|R\|, \|S\|))$	$O(\|R\|)$	$O(log_2\|R\| \cdot \|S\|)$	–, –
Sort-Merge	$O(\|R\| \cdot \log_2 \|R\| + \|S\| \cdot \log_2 \|S\|)$	$O(\max(\|R\|, \|S\|))$	$O(\|R\| \cdot \log_2 \|R\| + \|S\| \cdot \log_2 \|S\|)$	–, –
Merge	$O(\|R\| + \|S\|)$	$O(1)$	$O(\|R\| + \|S\|)$	S, S
Binary Search	$O(\|R\| \cdot \log_2 \|S\|)$	$O(1)$	$O(\|R\| \cdot \log_2 \|S\|)$	–, S
	...		...	

Tabela: Custo dos algoritmos(5)

Outras variações podem ser desenvolvidas...

Exemplo de Motivação

- O Algoritmo *Search* precisa de uma variação que ordene uma das coleções, caso nenhuma esteja ordenada.
- Nesse caso, qual das coleções ordenar, R ou S?
- Aquela que minimizar (ordenar + pesquisar):
  $O((|R| + |S|) \cdot \log_2 |S|)$ ou $O((|S| + |R|) \cdot \log_2 |R|)$
- e portanto deve-se ordenar a menor das duas coleções.
- Como tanto o algoritmo *Binary Search* quanto o *Merge* dependem que uma ou duas das coleções estejam ordenadas, poderiamos escrever um algoritmo de ordenação, que poderia ser chamado sempre que *Binary Search* ou *Merge* precisassem ser executados.
- Assim, nem a variação do algoritmo *Search* nem o algoritmo Sort-Merge seriam necessários.

Exemplo de Motivação

Objetivo

Ter uma ferramenta flexível e robusta para contar quantos pares de elementos iguais podemos formar tomando um elemento de cada coleção R e S.

Qual das várias rotinas `ContaPares` é a melhor?

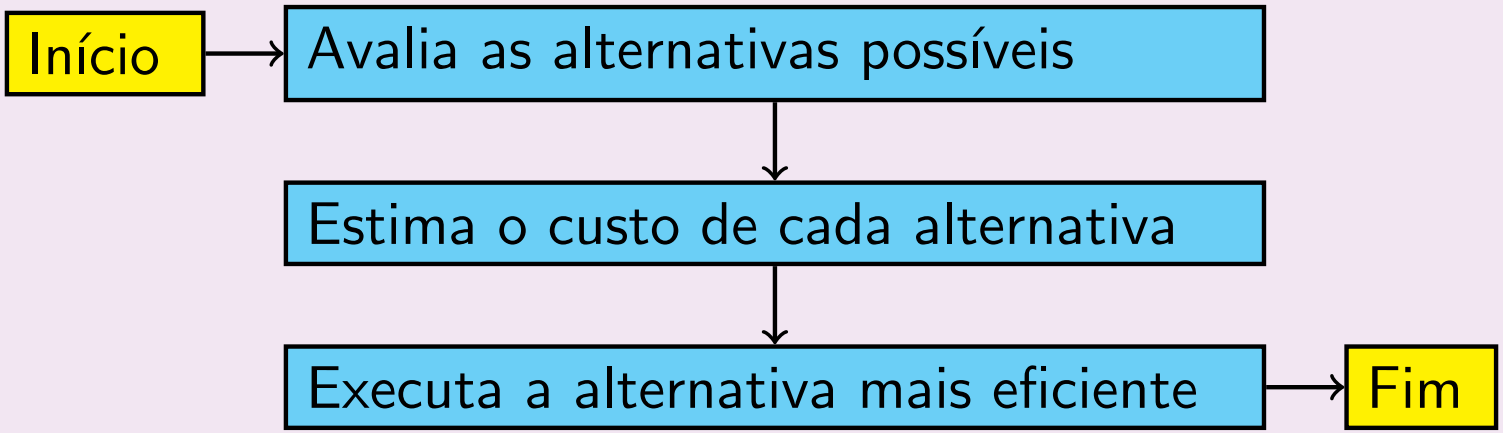
- Depende das propriedades dos dados!
  - Da cardinalidade de cada coleção,
  - Se está ordenado,
  - da memória disponível,
  - Do tipo de dados (como é feita a comparação), etc.

Exemplo de Motivação

Problema

- Nenhuma das rotinas `ContaPares` é sempre a melhor.
- Qualquer uma delas pode ser a melhor em determinadas combinações de propriedades.

- É necessário ter todas elas implementadas, e sua execução deveria ser precedida por uma avaliação de qual é a melhor opção naquele caso:



Exemplo de Motivação

- Incorporar um passo que avalia as alternativas e escolhe a melhor introduz um certo grau do paradigma Declarativo;
- Poder escolher se deve ser feita a chamada para um algoritmo de ordenação para alguma (ou ambas) as coleções, aumenta a flexibilidade e a "declaratividade";
- Nem todos os operadores podem ser executados sempre — por exemplo, se a comparação R[pR]θS[pS] não for $\theta =$ " $=$"?

  (o algoritmo de força bruta é especialmente importante nesses casos!);
- E se quizermos generalizar as operações que podemos fazer?

O Modelo Relacional permite modelar o problema, e os SGBDs são a implementação que realiza, em última instância, esse tipo de tarefa.

Exemplo de Motivação

- De fato, usando SQL, é possível expressar nosso problema com facilidade:

```
SELECT COUNT(R.r)
   FROM R, S
   WHERE R.r = S.s
```

- Os SGBD Relacionais modernos são sistemas sofisticados, que avaliam grande quantidade de alternativas para escolher uma que leve à execução eficiente das consultas expressas em SQL.
- Portanto, podemos imaginar que o **motor de um SGBD Relacional** implementa esses conceitos, seguindo como modelo de **organização** e de **otimização** o **Modelo Relacional**.
- É isso o que estudaremos no restante desta disciplina.

Tenham uma boa disciplina de
BD II - A arquitetura dos SGBDs Relacionais.

Arquitetura de SGBD Relacionais
— Recordação e Motivação para Bases de Dados II —

Caetano Traina Jr.

Grupo de Bases de Dados e Imagens
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo - São Carlos
caetano@icmc.usp.br

27 de fevereiro de 2013
São Carlos, SP - Brasil

FIM