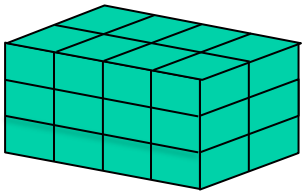
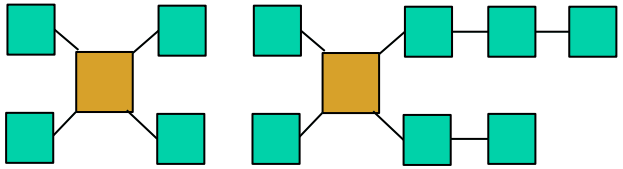
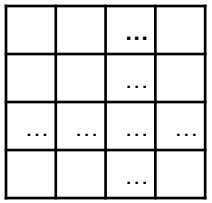
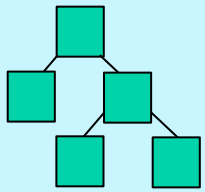
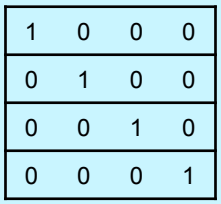
Modelagem Multidimensional
- Nível Físico -

Processamento Analítico de Dados

Profa. Dra. Cristina Dutra de Aguiar Ciferri

Arquitetura de 3 Camadas

	esquema	operações
conceitual	 metáfora do cubo de dados	Cube Álgebra
lógico	 *esquemas estrela e floco de neve* ROLAP  *estruturas matriciais* MOLAP	SQL MDX ...
físico	 *índices: árvores*  *índices bitmap* ROLAP	*estruturas e algoritmos proprietários* MOLAP	processamento e otimização de consultas

Índices Bitmap

- Índice bitmap sobre um atributo A de uma relação R
  - sequência ordenada de valores de chave, sendo que cada chave representa um valor distinto do domínio ativo de A
- Cada valor de chave
  - associado a um vetor de bits
  - especifica o conjunto de tuplas de R em que A assume aquele valor

Índices Bitmap

- Cada vetor de bits
  - possui tantos bits quanto as tuplas de R
  - i-ésimo bit
    - 1 se o valor de A na i-ésima tupla de R é igual ao valor de chave do vetor associado
    - 0 caso contrário

Exemplo

filial

chaveFilial	nomeFilial	cidade	estado	regiao	pais
1	Filial 1	Sao Carlos	SP	SE	Brasil
2	Filial 2	Araraquara	SP	SE	Brasil
3	Filial 3	Recife	PE	NE	Brasil
4	Filial 4	Ribeirao Preto	SP	SE	Brasil
5	Filial 5	Jaboatao	PE	NE	Brasil

Araraquara	Recife	Ribeirao Preto	Sao Carlos	Jaboatao
0	0	0	1	0
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	0	1

índice bitmap para o atributo cidade

NE	SE
0	1
0	1
1	0
0	1
1	0

índice bitmap para o atributo região

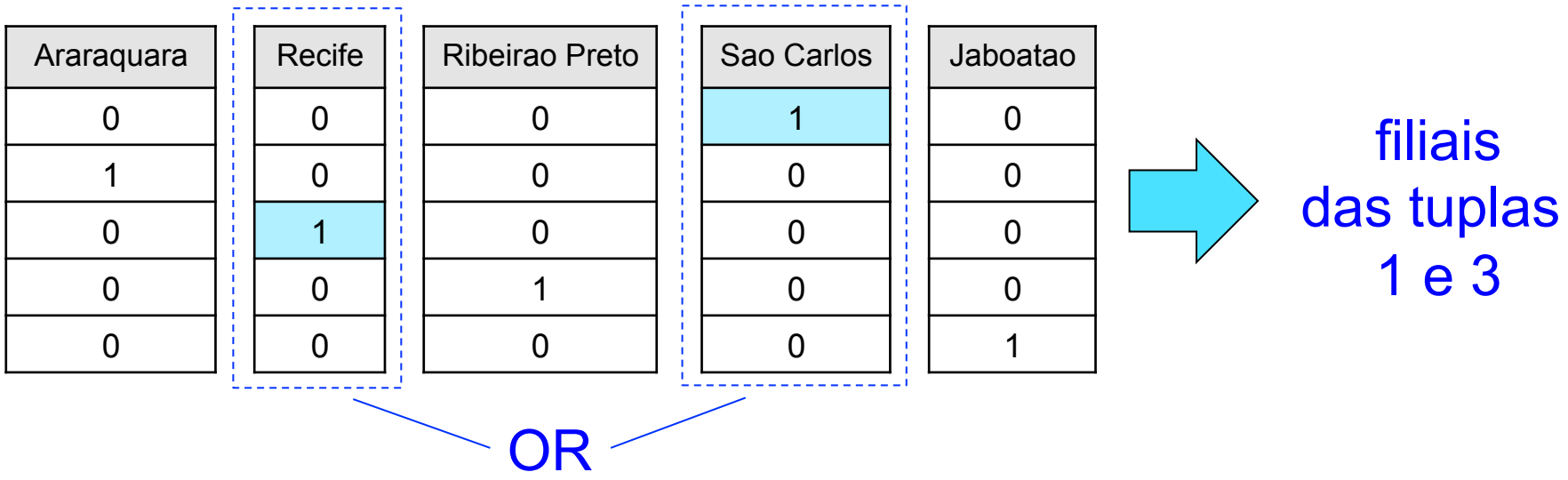
Vantagens e Desvantagens

- Vantagem
  - tempo de resposta reduzido, baseando no processamento de operações lógicas bit-a-bit OR, AND, XOR
- Desvantagem
  - requer grande espaço de armazenamento, especialmente para atributos com domínio ativo muito grande
  - atualização não é eficiente

indicado para DWs porque DWs são não voláteis

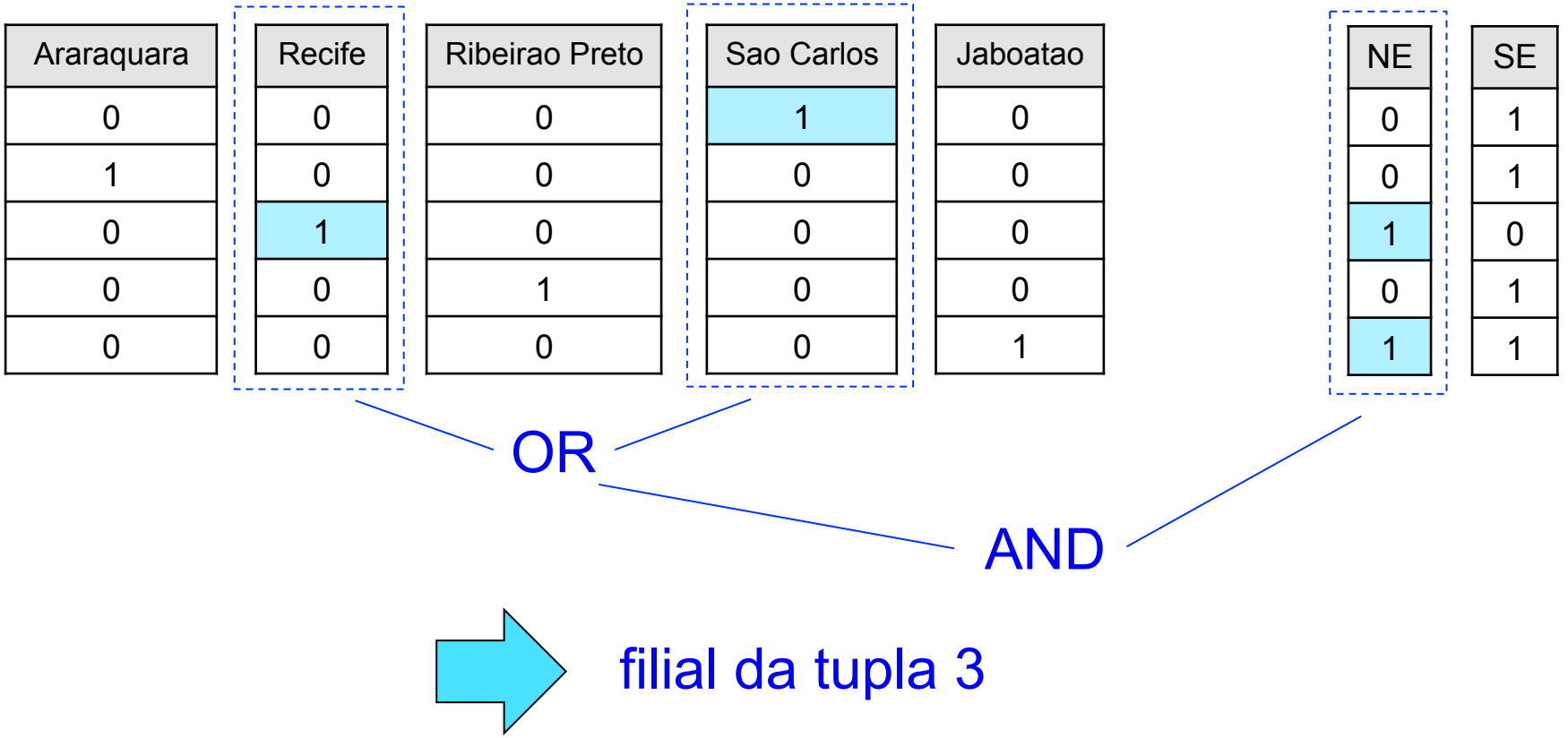
Exemplo: Consulta 1

- Quais as filiais localizadas em Sao Carlos ou em Recife?



Exemplo: Consulta 2

- Quais as filiais localizadas em Sao Carlos ou em Recife, e que sejam da região NE?



Índice Bitmap de Junção

- Característica
  - adapta índices bitmap para DW
  - usado para evitar a computação das junções requeridas pela junção-estrela
- Funcionamento
  - para cada atributo A_i de cada tabela de dimensão T_j de interesse
    - criar um índice bitmap que relaciona os valores de chave de A_i às tuplas da tabela de fatos

Exemplo (1/2)

tabela de dimensão: filial

chaveFilial	nomeFilial	cidade	estado	regiao	pais
1	Filial 1	Sao Carlos	SP	SE	Brasil
2	Filial 2	Araraquara	SP	SE	Brasil
3	Filial 3	Recife	PE	NE	Brasil
4	Filial 4	Ribeirao Preto	SP	SE	Brasil
5	Filial 5	Jaboatao	PE	NE	Brasil

tabela de dimensão: tempo

chaveTempo	dia	mes	trimestre	semestre	ano
1	1	janeiro	1	1	2014
2	2	janeiro	1	1	2014
3	1	janeiro	1	1	2015
4	2	janeiro	1	1	2015

tabela de dimensão: produto

chaveProduto	nomeProduto	marca	categoria	departamento
1	coca cola 600 ml	Coca Cola	refrigerante	1

Exemplo (2/2)

tabela de fatos: vendas

chave Filial	chave Produto	chave Tempo	lucro Dolar	unidades Vendidas
1	1	1	1	2
2	1	1	2	4
3	1	1	3	6
1	1	2	3.5	7
3	1	2	1	2
4	1	2	2	4
2	1	3	1.5	3
3	1	3	4	8
5	1	3	4.5	9
1	1	4	5	10
4	1	4	1	2
5	1	4	7.5	15

NE	SE
0	1
0	1
1	0
0	1
1	0
0	1
0	1
1	0
1	0
0	1
0	1
0	1

índice bitmap para o atributo região

2014	2015
1	0
1	0
1	0
1	0
1	0
1	0
0	1
0	1
0	1
0	1
0	1
0	1

índice bitmap para o atributo ano

Exemplo: Consulta 3

- Qual a quantidade de unidades vendidas para as filiais localizadas no NE ?

NE
0
0
1
0
1
0
0
1
1
0
0
1

somar as unidades vendidas das tuplas 3, 5, 8, 9, 12 da tabela de fatos vendas

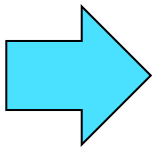
Exemplo: Consulta 4

- Qual a quantidade de unidades vendidas para as filiais localizadas no NE em 2014 ?

NE
0
0
1
0
1
0
0
1
1
0
0
1

AND

2014
1
1
1
1
1
1
0
0
0
0
0
0



somar as unidades vendidas
das tuplas 3, 5 da
tabela de fatos vendas

Visão Materializada

- Definição
  - especificação: intenção
  - dados: extensão
- Características
  - tabela simples que é derivada de outras tabelas
  - existe necessariamente em sua forma física
    - não é uma tabela virtual

Visão Materializada

- Utilidade em ambientes de DWing
  - replicação dos dados
  - armazenamento de dados agregados
  - aumento no desempenho de processamento de consultas
  - diminuição dos custos relacionados à atualização de outras visões materializadas

Visões Materializadas & Níveis de Agregação

- Nível inferior
  - conjunto de visões materializadas no qual as relações base residem nos provedores de informação
- Demais níveis
  - conjunto de visões materializadas no qual as relações base são as do nível imediatamente subjacente

Problemas Relacionados

- Três grandes linhas de pesquisa

  1. Identificação de quais visões devem ser materializadas

  2. Manutenção incremental das visões

  3. Reformulação transparente de consultas dos usuários de SSD usando visões materializadas

Identificando Visões

- Problema
  - requisito processamento de consultas eficiente é conflitante tanto com o tamanho do DW quanto com o tempo gasto para manter a consistência dos dados
- Trabalhos existentes
  - *entradas*: restrição de espaço, consultas frequentes dos usuários, uso de índices, custo de manutenção
  - *saída*: quais visões devem ser materializadas

Mantendo Visões

- Problema
  - visões materializadas tornam-se inconsistentes sempre que as relações base são alteradas
- Passos
  - detecção e propagação de alterações relevantes nos provedores
  - atualização incremental das visões materializadas tanto de nível inferior quanto dos demais níveis de agregação

Reformulando Visões

- Problema
  - a existência de diversas visões correlacionadas permite que uma mesma consulta seja respondida usando-se diferentes visões materializadas
- Trabalhos existentes
  - dado uma consulta Q e um conjunto de visões materializadas, encontrar uma reescrita de Q, chamada de Q', de forma que Q' seja equivalente a Q

Visão Materializada

```
CREATE MATERIALIZED VIEW nome_visão
[BUILD [DEFERRED | IMMEDIATE]]
[[REFRESH [COMPLETE | FAST | FORCE]]
        [ON COMMIT| ON DEMAND]]
        [[START WITH | NEXT] DATE ]]
[[ENABLE | DISABLE] QUERY REWRITE]
AS
<SELECT>
```

+ diversas outras opções

BUILD

- Quando a visão materializada é povoada
  - IMMEDIATE: imediatamente
  - DEFERRED: primeiro REFRESH

REFRESH

- Como é feita a atualização da visão
  - COMPLETE x FAST x FORCE
    - COMPLETE: atualiza completamente a visão, executando o comando SELECT
    - FAST: somente considera as alterações realizadas (atualização incremental)
    - FORCE: Oracle vai executar FAST sempre que possível, e COMPLETE caso contrário

REFRESH

- Como é feita a atualização da visão
  - ON COMMIT x ON DEMAND
    - ON COMMIT: atualiza a visão quando as operações realizadas na relação base forem finalizadas com sucesso
    - ON DEMAND: atualiza a visão somente quando um comando específico solicitar

REFRESH

- Como é feita a atualização da visão
  - START WITH x NEXT
    - START WITH: data na qual será realizada a primeira atualização automática
    - NEXT: intervalo de tempo entre duas atualizações automáticas consecutivas

QUERY REWRITE

- Se a visão materializada pode ser usada para reescrita de consultas
  - ENABLE: sim
  - DISABLE: não

Diversas outras opções encontram-se disponíveis, sendo que as funcionalidades dependem do SGBD

Exemplo

- Esquema
  - Campeonato de futebol
- Consulta base

```
SELECT cl.nomeClube, cl.apelidoClube, es.nomeest
       es.capacidadeest, eq.nomeeq, eq.nrotituloseq
FROM   clube cl, equipe eq, estadio es,
       clubepossuiest cles
WHERE  es.nomeest     = cles.nomeest AND
       cles.cnpjclube = cl.cnpjclube AND
       cl.cnpjclube   = eq.cnpjclube;
```

Visão Materializada 1 (1/5)

- Criar a visão materializada times que seja *povoada imediatamente* e *atualizada completamente* sempre que houver um *commit* nas tabelas base

Visão Materializada 1 (2/5)

- Criar a visão materializada times que seja *povoada imediatamente* e *atualizada completamente* sempre que houver um *commit* nas tabelas base

```
CREATE MATERIALIZED VIEW times
      ("Nome do Clube", "Apelido do Clube",
       "Nome do Estadio", "Capacidade do Estadio",
       "Nome da Equipe", "Numero de Titulos")
BUILD IMMEDIATE
REFRESH ON COMMIT
AS CONSULTA_BASE;
```

Visão Materializada 1 (3/5)

- Listar todos os dados de times

```
SELECT * FROM times;
```

- Inserir uma nova tupla

```
INSERT INTO EQUIPE (cnpjclube, nomeeq,
       nrojogadoreseq, nrotituloseq)
VALUES ('60.517.984/0001-04','MASTER',50,10);
```

Visão Materializada 1 (4/5)

- Listar todos os dados de times

```
SELECT * FROM times;
```

A visão estará atualizada? Por quê?

Visão Materializada 1 (5/5)

- Realizar *commit*

```
COMMIT;
```

- Listar todos os dados de times

```
SELECT * FROM times;
```

A visão estará atualizada? Por quê?

- Excluir times

```
DROP MATERIALIZED VIEW times;
```

Visão Materializada 2 (1/4)

- Criar a visão materializada times que seja *povoada* apenas quando o *usuário* solicitar.

Visão Materializada 2 (2/4)

- Criar a visão materializada times que seja *povoada* apenas quando o *usuário* solicitar.

```
CREATE MATERIALIZED VIEW times
      ("Nome do Clube", "Apelido do Clube",
       "Nome do Estadio", "Capacidade do Estadio",
       "Nome da Equipe", "Numero de Titulos")
BUILD DEFERRED
AS CONSULTA_BASE;
```

Visão Materializada 2 (3/4)

- Listar todos os dados de times

```
SELECT * FROM times;
```

A visão possui dados? Por quê?

Visão Materializada 2 (4/4)

- Povoar times

```
EXECUTE DBMS_MVIEW.REFRESH('times');
```

- Listar todos os dados de times

```
SELECT * FROM times;
```

A visão possui dados?
Por quê?

- Excluir times

```
DROP MATERIALIZED VIEW times;
```

Visão Materializada 3 (1/6)

- Criar a visão materializada times que *seja povoada imediatamente* e que *atualizada* apenas quando o *usuário* solicitar.

Visão Materializada 3 (2/6)

- Criar a visão materializada times que *seja povoada imediatamente* e que *atualizada* apenas quando o *usuário* solicitar.

```
CREATE MATERIALIZED VIEW times
       ("Nome do Clube", "Apelido do Clube",
        "Nome do Estadio", "Capacidade do Estadio",
        "Nome da Equipe", "Numero de Titulos")
BUILD IMMEDIATE
REFRESH ON DEMAND
AS CONSULTA_BASE;
```

Visão Materializada 3 (3/6)

- Listar todos os dados de times

```
SELECT * FROM times;
```

- Inserir uma nova tupla

```
INSERT INTO EQUIPE (cnpjclube, nomeeq,
       nrojogadoreseq, nrotituloseq)
VALUES ('60.517.984/0001-04','MASTER',50,10);
```

Visão Materializada 3 (4/6)

- Listar todos os dados de times

```
SELECT * FROM times;
```

A visão estará atualizada? Por quê?

Visão Materializada 3 (5/6)

- Realizar *commit*

```
COMMIT;
```

- Listar todos os dados de times

```
SELECT * FROM times;
```

A visão estará atualizada? Por quê?

Visão Materializada 3 (6/6)

- Solicitar a atualização de times

```
EXECUTE DBMS_MVIEW.REFRESH('times');
```

- Listar todos os dados de times

```
SELECT * FROM times;
```

A visão estará atualizada? Por quê?

- Excluir times

```
DROP MATERIALIZED VIEW times;
```