Projeto Físico de Bancos de Dados

Laboratório de Bancos de Dados

Prof. Dr. José P. Alcázar

Revisão

- Depois do projeto conceitual, refinamento do esquema e a definição de visões → teremos os esquemas lógico e externo do BD.
- Próximo passo é escolher índices, tomar decisões de agrupamentos e refinar os esquemas lógico e externo (se for necessário) para alcançar as metas de desempenho.
- Nos devemos começar pelo entendimento da carga de trabalho:
  - As consultas mais importantes e a sua freqüência
  - A atualizações mais importantes e a sua freqüência
  - O desempenho desejado para essas consultas e atualizações

Decisões a serem tomadas

- Quais índices deveríamos criar?
  - Quais relações deveriam ter índices? Que campos deveriam ser chave de busca? Deveríamos construir vários índices?
- Para cada índice, que tipo de índice deveria ser?
  - De agrupamento? hash/árvore?
- Deveríamos mudar o esquema lógico?
  - Considere esquemas normalizados alternativos (Lembre, existem várias escolhas de decomposição em BCNF, etc.)
  - Deveríamos desfazer algumas decomposições e passar para formas normais mais simples (desnormalizar).
  - Particionamento vertical, replicação, visões, ...
- Informação sobre a carga de trabalho no processo de projeto é difícil → tuning

Seleção de índices para junções

- Considerar quais índices criar, começamos com a lista de consultas (incluindo consultas que aparecem nas atualizações) → relações → atributos.
- Uma abordagem, considere as consultas mais importantes e para cada uma, determinar qual plano o otimizador escolheria dados os índices a serem criados. É possível um plano melhor adicionando mais índices? Consultas por faixas de dados → árvores B+; consultas exatas → hash. Clustering beneficia consultas por faixas e consultas exatas se vários dados contêm o mesmo valor da chave.
- Antes de ser definido um novo índice, deve-se ser avaliado o impacto sobre as atualizações: Índices podem fazer as consultas mais rápidas, atualizações mais lentas. Precisa mais espaço.

Diretrizes para seleção de índices

1. Não construa um índice a menos que alguma consulta se beneficie. Tente construir índices que beneficiem o máximo número de consultas.
2. Escolha de chave de busca. Atributos mencionados na cláusula WHERE são candidatos a índices.
3. Chaves de busca de multi-atributos.
   - Uma cláusula WHERE inclui condições sobre mais de um atributo de uma relação.
   - Elas facilitam estratégias de avaliação baseadas unicamente em índices para consultas importantes (acesso à relação pode ser evitado).
4. No máximo um índice pode ser clusterizado por relação e eles podem melhorar bastante o desempenho. Escolha este baseado sobre consultas importantes que se beneficiem.
   - Consultas de faixas são bastante beneficiadas. Se várias consultas de faixas são feitas numa relação. O que fazer?
   - Se for uma estrategia de avaliação baseada unicamente em índices, não é necessário clusterizar

Diretrizes para seleção de índices

1. Índices Hash vs. Árvores. Um índice em árvore B+ é preferível, porque suporta consulta de faixas e consultas de igualdade. Um índice hash é preferível:
   - Suporte de junções de laços aninhados indexados: a relação indexada é a interna, e a chave de busca inclui as colunas de junção.
   - Uma consulta de igualdade muito importante e nenhuma por faixa.
2. Balanceamento do custo de manutenção do índice.
   - Se a manutenção do índice diminui a eficiência de atualizações freqüentes → considere a remoção do índice
   - Entretanto, a criação de um índice pode melhorar a eficiência de uma atualização.

Exemplos de índices de agrupamento com uma só Tabela

```
SELECT E.dno
FROM Emp E
WHERE E.age > 40
```

- Índice em Árvore B+ sobre E.age pode ser usado para obter as tuplas
  - Que tão seletiva é a condição? Que fração do empregados é maior de 40?
  - É o índice agrupado?

Exemplos de índices de agrupamento

```
SELECT    E.dno, COUNT(*)
FROM      Employee E
WHERE     E.age > 10
GROUP BY E.dno
```

- Considere a consulta GROUP BY.
  - Se várias tuplas têm E.age > 10, o uso do índice E.age e ordenação das tuplas recuperadas pode ser custosa.
  - Um índice de agrupamento para E.dno pode ser melhor

Exemplos de índices de agrupamento

SELECT E.dno
FROM Emp E
WHERE E.hobby = ‘Stamps’

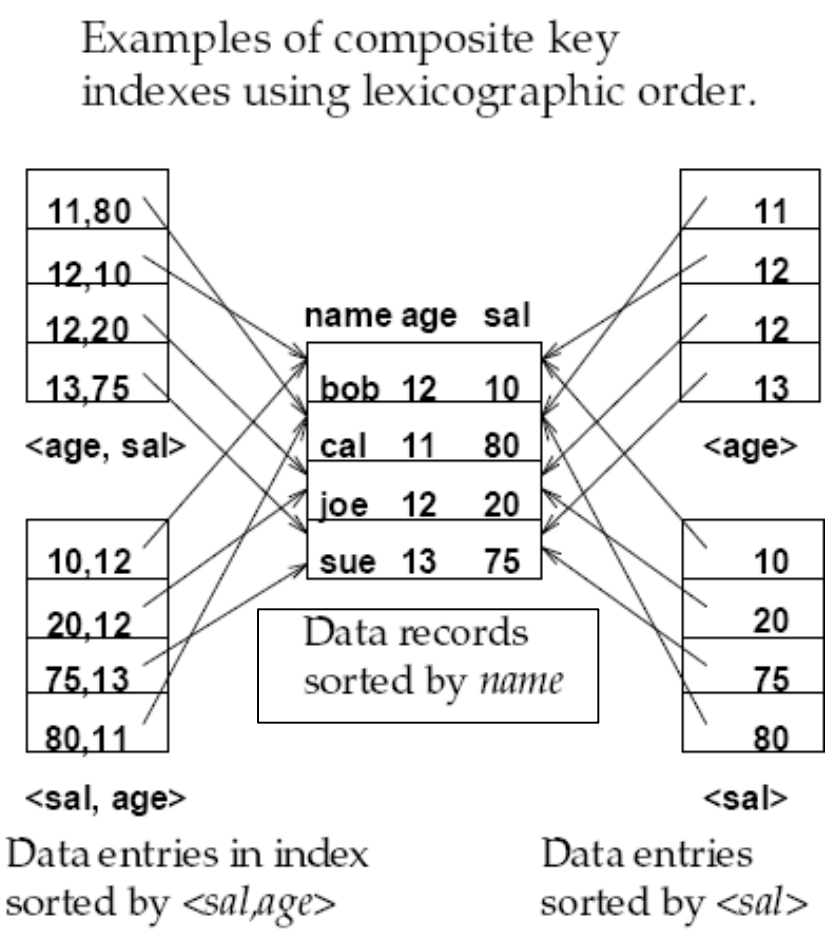
- Consultas de igualdade com duplicatas:
  - Agrupamento sobre E.hobby ajuda.

SELECT E.dno, COUNT(*)
FROM Emp E
GROUP BY E.dno

- Um plano para esta consulta é ordenar Emp por dno para calcular count. Entretanto, se existir um índice (B+) sobre dno, **pode ser respondida procurando somente o índice.** Nao precisa agrupamento.

Índices com atributos compostos

- Índice composto: busca por uma combinação de campos.
  - Consulta igualdade: todo campo é igual a uma cte. Índice <sal,age> Para age = 20 e sal = 10 m.
  - Consulta faixa: Algum campo não é uma cte. Ex: age = 20; ou age = 20 e sal > 10
- Entrada no índice ordenada pelo campo.



Índices com atributos compostos

- Para recuperar registros Emp com age=30 e sal=4000, um índice <age,sal> seria melhor que um índice sobre age ou um índice sobre sal.
  - Um índice hash
- Se condição é: 20<age<30 e 3000<sal<5000:
  - Índice em árvore agrupado sobre <age,sal> ou <sal,age> é melhor.
- Se condição é: age=30 e 3000<sal<5000:
  - Índice agrupado por <age,sal> bem melhor que <sal,age>
- Índices compostos são maiores e atualizados com mais freqüência.

Exemplos de projeto de índices compostos

```
SELECT E.eid
FROM Employee E
WHERE E.age BETWEEN 20 AND 30
                AND E.sal BETWEEN 3000 AND 5000
```

- Um índice composto sobre <age, sal> poderia ajudar se a clausula WHERE for seletiva. Hash ou B+? Agrupado ou não agrupado?
- Se as condições por age e sal são igualmente seletivas então um índice <age, sal> ou <sal,age> são igualmente efetivos.

Exemplos de projeto de índices compostos

```
SELECT E.eid
FROM Employee E
WHERE E.age = 25
                AND E.sal BETWEEN 3000 AND 5000
```

- Um índice B+ composto agrupado sobre <age, sal> dará um bom desempenho. Por outro lado, um índice B+ composto agrupado sobre <sal, age> não funcionarão também.

Exemplos de projeto de índices compostos

```
SELECT AVG (E.sal)
FROM Employee E
WHERE E.age = 25
            AND E.sal BETWEEN 3000 AND 5000
```

- Um índice B+ sobre <age,sal> permite responder a consulta buscando unicamente no índice. O índice <sal, age> também permite mas deve recuperar mais entradas no índice.

Exemplos de projeto de índices compostos

```
SELECT      E.dno, COUNT(*)
FROM        Employee E
WHERE       E.sal = 10.000
GROUP BY E.dno
```

- Um índice sobre dno não permite avaliação só no índice. Mas um índice composto, sim. Com um índice B+ por <sal,dno> é possível e mais eficiente, mas se a condição passa a ser “E.sal > 10.000”, já não é possível. Qual é a consulta mais freqüente?

Exemplos básicos de Seleção de Índices casos de mais de uma tabela - junção

SELECT E.ename, D.mgr

FROM Emp E, Dept D

WHERE D.dname = 'Toy' AND E,dno=D.dno

- Um índice hash por D.dname suporta a seleção ‘Toy’
  - Dado isto, um índice por D.dno não é necessário.
- Um índice hash por E.dno nos permite fazer o casamento de tuplas Emp (internas) para cada tupla Dept (externa) selecionada. A escolha dos índices foi guiada pelo plano de avaliação desejado, veja Fig.
- O que acontece se WHERE incluísse: “...AND E.age =25” ?
  - Poderíamos recuperar tuplas Emp usando o índice E.age, então juntamos com Tuplas Dept que satisfazem a seleção dname. Comparável à estratégia usada para o índice E.dno.
  - Se o índice E.age foi já criado, esta consulta oferece muito menos motivação para adicionar um índice E.dno.

Exemplos básicos de Seleção de Índices casos de mais de uma tabela - junção

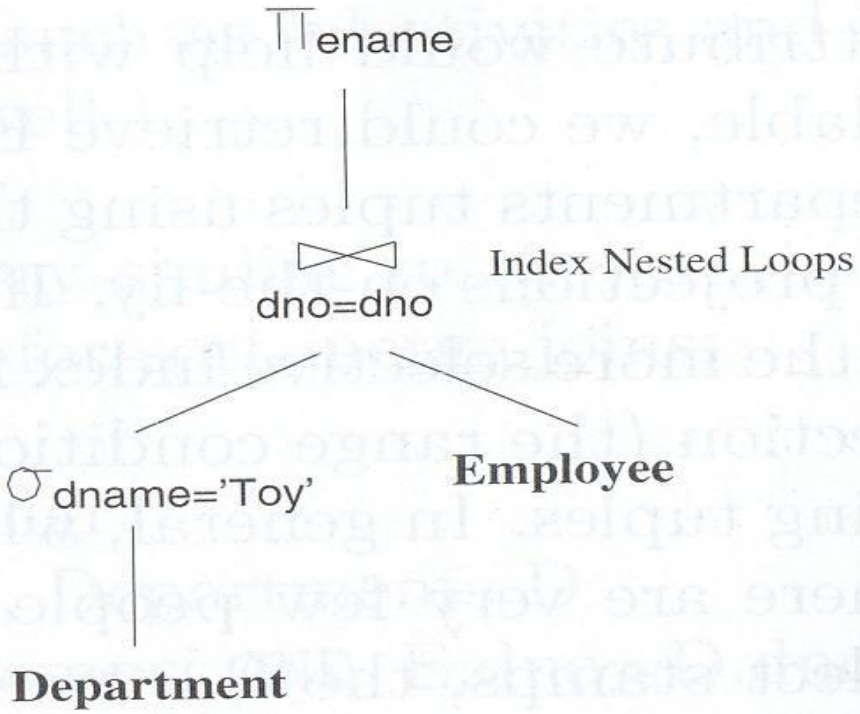


Figure 20.1 A Desirable Query Evaluation Plan

Exemplos básicos de Seleção de Índices casos de mais de uma tabela - junção

```
SELECT E.ename, D.dname
FROM Emp E, Dept D
WHERE E.sal BETWEEN 10000 AND 20000
        AND E.hobby = ‘Stamps' AND E.dno=D.dno
```

O uso de BETWEEN é recomendado

- Claramente, Emp deveria ser a relação externa.
  - Seria bom um índice hash sobre D.dno.
- Qual índice deveríamos construir para Emp?
  - Um ávore B+ sobre E.sal poderia ser usado, ou um índice sobre E.hobby poderia ser usado. Unicamente um desses é necessitado, e qual é o melhor depende da seletividade das condições.
    - Em geral, as seleções de igualdades são mais seletivas que as seleções por faixas.
- Como ambos exemplos indicam, a escolha de índices é guiada pelo(s) plano(s) que esperamos o otimizador considera para uma consulta. ***É importante entender o processo de otimização***.

Agrupamento e Indexação

SELECT E.ename, D.mgr

FROM Emp E, Dept D

WHERE D.dname='Toy' AND E.dno=D.dno

- Agrupamento é especialmente importante quando acessamos tuplas internas numa junção de laço aninhado indexada
  - Deveríamos fazer um índice agrupado sobre E.dno
- Suponhamos que a cláusula WHERE é agora:

  WHERE E.hobby=Stamps AND E.dno=D.dno
  - Se muitos empregados colecionam selos, junção sort-merge pode ser válido considerar. Um índice agrupado sobre D.dno ajudaria.
- Resumo: Agrupamento é útil sempre que várias tuplas têm que ser recuperados. Veja Fig.

Agrupamento e Indexação

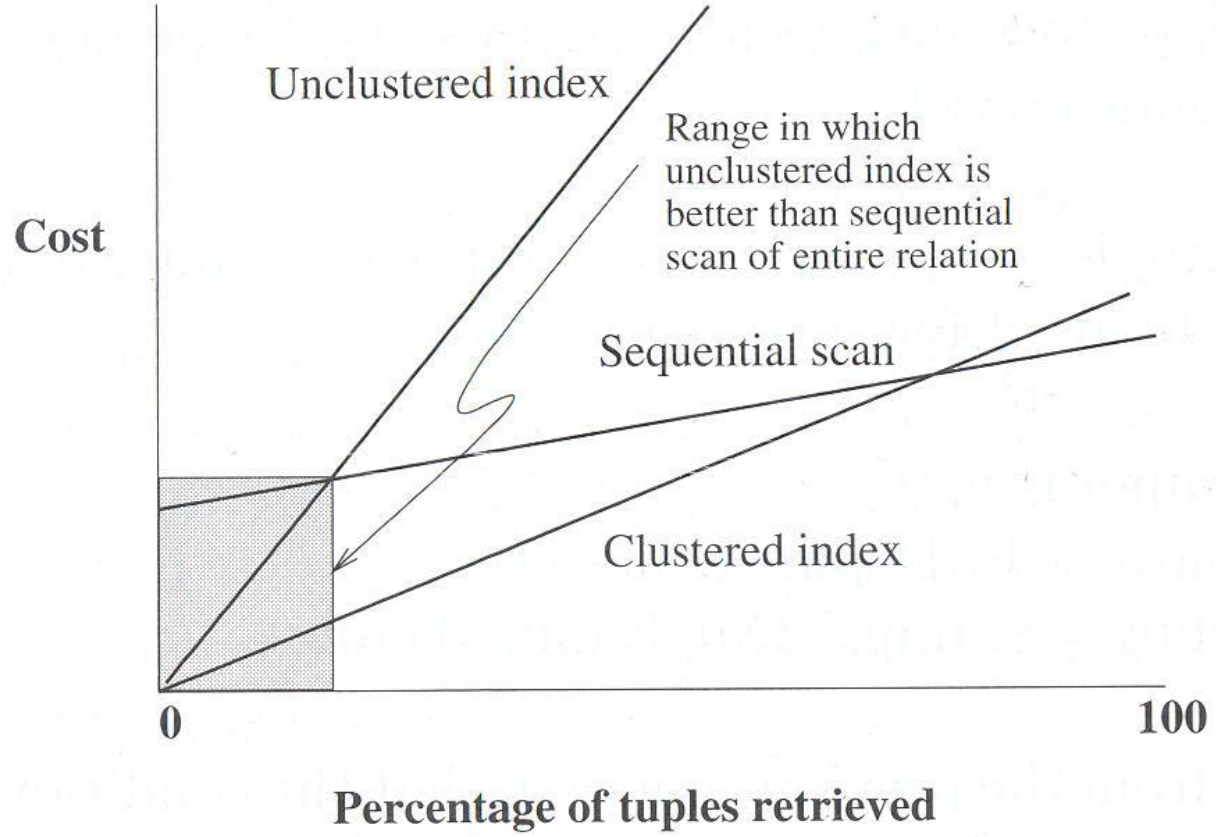


Figure 20.2 The Impact of Clustering

Ferramentas para apoiar a seleção de índices

- O número de índices possíveis a serem considerados é potencialmente muito grande.
- Grandes aplicações, sistemas ERP, criam dezenas de milhares de relações diferentes. Tuning manual é um trabalho gigantesco.
- Importante o desenvolvimento de ferramentas que ajudam aos administradores de BDs para selecionar índices apropriados para uma dada carga de trabalho.
- Primeira geração de ferramentas: index tuning wizards ou index advisors. Ferramentas separadas do motor de BD.
- A geração mais recente, integrada ao SGBD evitando a duplicação do modelo de custo do otimizador de consultas.

Revisão do Tuning de Banco de Dados

- Depois do projeto inicial do BD, o uso real do BD fornece uma fonte de informação de muito valor que pode ser usada para refinar o projeto inicial. Monitoração cuidadosa das consultas pode revelar problemas não esperados: ex. O otimizador pode não estar usando alguns índices necessários para gerar bons planos.
- Um tuning constante é importante para obter o melhor desempenho possível. Três tipos de tuning: tuning de índices, tuning do esquema e tuning de consultas.
- A seleção de índices se aplica ao tuning de índices.

Tuning de índices.

- A escolha inicial de índices pode ser refinada:
  - A carga de trabalho revela que algumas consultas e atualizações consideradas importantes inicialmente não são muito freqüentes.
  - Aparecem novas consultas que são importantes
  - A escolha inicial de índices tem que ser revisada com a obtenção de nova informação. Raciocínio igual ao usado no projeto inicial.
  - Podemos encontrar que o otimizador não esteja gerando os mesmos planos que a gente esperava.
- Exemplo:

Tuning de índices.

SELECT D.mgr, E.eid

FROM Employees E, Departments D

WHERE D.dname = ‘Toy’ AND E.dno = D.dno

- Um bom plano seria usar um índice por dname para Departments e usar outro índice por dno de Employees. Este último não precisaria ser agrupado.
- Suponhamos que a consulta está gastando um tempo maior do esperado. Podemos pedir olhar o plano produzido pelo otimizador (Comando EXPLAIN do PostgreSQL)
- Percebemos que uma busca só no índice não é feito. Então?

Tuning de índices.

- Algumas outras limitações dos otimizadores são:
  - Eles não manipulam efetivamente as seleções que envolvem expressões de string, aritméticas ou valores nulos
  - Alguns índices precisam ser reorganizados. Índices estáticos como ISAM podem gerar longas cadeias de overflow. Mesmo índices B+ quando a implementação não mistura as páginas sob remoção → REINDEXING de PostgreSQL.
  - As estatísticas são atualizadas unicamente quando um programa utilitário especial é executado (ANALYZE)

Tuning do Esquema lógico

- Se percebemos que a escolha atual de esquema relacional não preenche os objetivos de desempenho para qualquer conjuntos de esquemas físicos → re-projetar o esquema lógico.
- Evolução do esquema, para melhorar o desempenho.

Tuning do Esquema lógico

- A escolha do esquema lógico deveria ser guiado pela carga de trabalho, além de aspectos de redundância:
  - Nós podemos optar pela 3FN antes da FNBC
  - A carga de trabalho pode influenciar a escolha que nos fazemos para decompor uma relação em 3FN ou FNBC.
  - Nós podemos decompor ainda mais um esquema na FNBC
  - Nos deveríamos des-normalizar (desfazer um passo de decomposição), ou nós deveríamos adicionar campos a uma relação.
  - Nós deveríamos considerar a decomposição horizontal.
- Se tais mudanças são feitas após de um BD esteja em uso, chamado evolução do esquema; nós deveríamos fazer transparente algumas destas mudanças às aplicações definindo *visões*.

Tuning de Consultas e Visões

- Ao perceber que uma consulta está executando muito mais lentamente do esperado, devemos examinar a consulta com cuidado → reescrita da consulta além de algum tuning de índices (solução). Deve-se verificar se o índice precisa de reconstrução, ou se as estatísticas são muito antigas.
- Não será discutido o tuning de visões separadamente.

Tuning de Consultas e Visões

- Algumas vezes, o SGBD pode não estar executando o plano que você tinha em mente. Áreas comuns de fraqueza dos otimizadores:
  - Seleções que envolvem valores nulos
  - Seleções que envolvem expressões aritméticas e cadeias de caracteres.
  - Seleções que envolvem condições OR
  - Falta de características de avaliação como estratégias ou certos métodos de junção ou pobre estimação de tamanhos.
- Verificar o plano que está sendo usado. Então ajustar a escolha de índices ou re-escrever a consulta/vista

Escolhas no Tuning do Esquema lógico

- Esquema Exemplo:

  Contracts(Cid, Sid, Jid, Did, Pid, Qty, Val)

  Depts(Did, Budget, Report)

  Suppliers(Sid, Address)

  Parts(Pid, Cost)

  Projects(Jid, Mgr)

Um departa-mento compra no máximo uma peça de qualquer fornecedor

Um projeto compra uma peça dada usando um único contrato

- Nós concentraremos em Contracts descrito como CSJDPQV. As seguintes restrições vigoram: JP → C, SD → P, C é a chave principal
  - Quais são as chaves candidatas de CSJDPQV?
  - Em qual forma normal está a relação?

Passando para uma forma normal mais fraca

- CSJDPQV pode ser decomposta em SDP e CSJDQV e ambas relações estão na FNBC. (Qual DF sugere que nós façamos isto?)
  - Decomposição sem perda, mas não preservamos as dependências.
  - Adicionando CJP faz esta decomposição preservar as dependências.
- Suponhamos que a seguinte consulta é muito importante:
  - Achar o número de cópias Q da peça P solicitadas no contrato C.
  - Precisa de uma junção sobre o esquema decomposto, mas pode ser respondido por uma busca na relação original
  - O custo adicional poderia persuadirmos a manter a relação em 3FN.

Desnormalização

- Suponhamos que a seguinte consulta é importante:
  - É o valor de um contrato menor que o orçamento do departamento?
- Para melhorar esta consulta, nós deveríamos adicionar um campo *budget* B a Contracts
  - Isto introduz a DF D → B
  - Então, Contracts já não está na 3FN.
- Nós deveríamos escolher esta mudança se a consulta é importante o suficiente, e não podemos melhorar o desempenho de outra forma (ex: adicionando índices ou escolhendo um esquema 3FN alternativo).

Escolha das decomposições

- Existem duas formas de decompor CSJDPQV em FNBC:
  - SDP e CSJDQV; sem perda de inf. mas não preserva deps.
  - SDP, CSJDQV e CJP; também preserva as dependências.
- A diferença entre estas é realmente o custo de reforçar a DF JP → C.
  - 2nda decomposição: índice sobre JP na relação CJP.
  - 1eira: CREATE ASSERTION CheckDep

```
CHECK (NOT EXISTS ( SELECT *
                    FROM PartInfo P, ContractInfo C
                    WHERE P.sid=C.sid AND P.did=C.did
                    GROUP BY C.jid, P.pid
                     HAVING COUNT(C.cid)>1))
```

Escolha das decomposições (Cont.)

- As seguintes restrições ICs vigoram:

  JP → C, SD → P, C é a chave primária.

- Suponha que, além, um dado fornecedor sempre coloca o mesmo preço para uma dada peça: SPQ → V.
- Se nós decidimos que desejamos decompor CSJDPQV na FNBC, nós agora temos uma terceira escolha:
  - Comece por decompor esta em SPQV e CSJDPQ
  - Então, decompor CSJDPQ (não 3FN) em SDP, CSJDQ
  - Isto nós da a decomposição sem perda: SPQV, SDP, CSJDQ.
  - Para preservar JP → C, nós podemos adicionar CJP, como antes.
- Escolha: {SPQV, SDP, CSJDQ} ou {SDP, CSJDQV} ?

Decomposição de uma relação na FNBC

- Suponhamos que escolhemos [SDP, CSJDQV}. Esta na FNBC, e não há nenhuma razão de decompor ainda mais.
- Entretanto, suponhamos que estas consultas são importantes:
  - Achar os contratos no nome do fornecedor F
  - Achar os contratos nos quais o departamento D está envolvido.
- Decompondo CSJDQV ainda mais em CS, CD e CJQV poderia aumentar a velocidade das consultas (Porque?)
- Por outro lado, a seguinte consulta é mais lenta:
  - Achar o valor total dos contratos no nome do fornecedor F. Qual é mais importante?

Relações menores e controle de concorrência

Decomposições horizontais

- Nossa definição de decomposição: Relação é trocada por uma coleção de relações que são projeções. Caso mais importante.
- Algumas vezes, poderia se desejar trocar uma relação por uma coleção de relações que são seleções.
  - Cada nova relação tem a mesma estrutura (esquema) que a original, mas com um subconjunto de tuplas.
  - Em conjunto, as novas relações contêm todas as tuplas da original. Tipicamente, as novas relações são disjuntas.

Decomposições Horizontais (Cont.)

- Suponhamos que os contratos com val > 10000 são sujeitos a diferentes regras. Isto significa que consultas sobre Contracts com freqüência conterão a condição val>10000.
- Uma maneira de tratar com isto é construir um índice em árvore B+ agrupado sobre o campo val de Contracts.
- Um segundo enfoque é trocar Contracts por duas novas relações: LargeContracts e SmallContracts, com os mesmos atributos (CSJDPQV).
  - Oferece os benefícios do índice sem o “overhead” da manutenção
  - Podemos construir índices agrupados sobre outros atributos. Se outras consultas precisarem.

Fazendo transparente as mudanças no esquema lógico

```
CREATE VIEW Contracts(cid, sid, jid, did, pid, qty, val)
        AS SELECT *
        FROM LargeContracts
        UNION
        SELECT *
        FROM SmallContracts
```

- A troca de Contracts por LargeContracts e SmallContracts pode ser feita transparente pela vista.
- Entretanto, as consultas com a condição val > 10000 devem ser feitas sobre LargeContracts para a execução eficiente: assim, os usuários relacionados com o desempenho têm que estar consciente da mudança.

Tuning de Consultas e Vistas – Reescrita de Consultas SQL

- Complicado para a interação de:
  - NULLs, dijunções, duplicatas, agregação, ou, sub-consultas.
- Diretriz: Usar unicamente um “bloco de consulta”, se possível.

```
SELECT DISTINCT *
FROM Sailors S
WHERE S.ename IN
        (SELECT Y.sname
         FROM YoungSailors Y)
```

——

```
SELECT DISTINCT S.*
FROM Sailors  S,
        YoungSailors Y
WHERE S.ename = Y.sname
```

Tuning de Consultas e Vistas – Reescrita de Consultas SQL

- SELECT E.dno

  FROM Empregado E

  WHERE E.hobby=`Selos´ OR E.idade=10
- Se nos temos índices sobre os dois atributos. Um otimizador falharia reconhecer esta oportunidade. O que fazer?

Mais diretrizes sobre Tuning de Consultas

- Minimizar o uso de DISTINCT: não é necessário se as duplicatas são aceitáveis, ou se a resposta contem uma chave
- Minimizar o uso de GROUP BY e HAVING:

```
SELECT MIN(E.age)
FROM Employees E
GROUP BY E.dno
HAVING E.dno = 102
```

```
SELECT MIN (E.age)
 FROM Employees E
 WHERE E.dno = 102
```

Considere SGBD usa de índices quando escreve expressões aritméticas: E.age = 2*D.age beneficiará do índice E.age, mas não pode se beneficiar do índice D.age!

Diretrizes sobre Tuning de Consultas

- Evite usando relações intermediárias:

```
SELECT * INTO Temp
FROM Emp E, Dept D
WHERE E.dno = D.dno
      AND D.mgrname = “Joe”
```

```
SELECT T.dno, AVG(T.sal)
FROM Temp T
GROUP BY T.dno
```

Vs.

```
SELECT E.dno, AVG(E.sal)
FROM Empl E, Dept D
WHERE E.dno=D.dno AND D.mgrname = “Joe”
GROUP BY E.dno
```

- Não materializa a relação intermediária Temp.
- Se existir um índice árvore B+ sobre <dno,sal>, um plano somente de índices pode ser usado para recuperar tuplas Emp na segunda consulta.

Impacto da Concorrência.

- Num sistema com vários usuários concorrentes, vários pontos adicionais devem ser considerados. O uso de locks.diminue o desempenho das aplicações
  - Redução da duração dos locks
  - Redução de hot spots

Reduzir a duração dos locks

- Retardar as solicitudes de lock. Protelar as mudanças até o final das transações.
- Fazer as transações mais rápidas. Cuidadoso particionamento das tuplas numa relação e seus índices associados a través de uma coleção de discos pode melhorar o acesso concorrente.
- Mudar transações extensas por curtas. Reescreva uma transação como duas ou mais. Problema de atomicidade.
- Construir um depósito.
- Considerar um nível mais baixo de isolamento.

Reduzir hot spots

- Um “hot spot” é um objeto de BD que é acessado e modificado freqüentemente, e causa muitos retardos de bloqueio.
- Técnicas:
  - Retardar as operações sobre hot spots: Retardar pedidos de lock.
  - Otimizar padrões de acesso:
  - Operações de partição sobre hot spots
  - Escolha de índices

Resumo

- O projeto de BDs consiste de várias tarefas:

  análise de requisitos, projeto conceitual, projeto lógico, projeto físico e tuning.
  - Em geral, temos que ir para frente e para tras com essas tarefas para refinar um projeto de BD, e as decis{oes numa tarefa podem influenciar as escolhas em outra tarefa.
- O entendimento da natureza da carga de trabalho da aplicação, e as metas de desempenho, é essencial para desenvolver um bom projeto.
  - Quais são as consultas e atualizações mais importantes? Quais atributos/relações estão envolvidos?

Resumo

- O esquema conceitual deveria ser refinado considerando critérios de desempenho e carga de trabalho:
  - Pode escolher 3NF ou mais baixa forma normal sobre a FNBC.
  - Pode escolher entre decomposições alternativas em FNBC (ou 3FN) baseado na carga de trabalho.
  - Pode desnormalizar, ou desfazer algumas decomposições
  - Pode decompor uma relação em FNBC ainda mais.
  - Importância da preservação da dependência baseada na depêndencia a ser preservada, e o custo de verificação da IC,
    - Podemos adicionar uma relação para manter a preservação de dependência; ou senão, pode checar a dependência usando uma junção.

Resumo

- No tempo, índices tem que ser bem tuned (eliminados, criados, re-feitos) por desempenho.
  - Deveria determinar o plano usado pelo sistema, e ajustar a escolha de índices apropriadamente.
- Os sistemas ainda não podem achar um bom plano:
  - Somente planos profundos à esquerda são considerados
  - O otimizador pode se confundir com valores nulos, condições aritméticas, expressões de string, o uso de ORs, etc.
- Então, outra opção é reescrever a consulta&vista:
  - Evitar consultas aninhadas, relações temporárias, condições complexas, e operações como DISTINCT e GROUP BY.

Quando é necessário DISTINCT

- O conjunto de valores ou registros retornados não deveriam conter duplicatas.
- Os campos retornados não contêm (como um subconjunto) uma chave da relação criada pelas cláusulas FROM e WHERE.
- Se os campos retornados constituem uma chave de uma tabela T e todos as outras tabelas fazem um equijoin com T pelas suas chaves, então os valores retornados não conterão duplicatas, e DISTINCT não será necessário.

Generalização

- Uma tabela T é ***privilegiada*** se os campos retornados pelo select contêm uma chave de T.
- Seja R uma tabela não ***privilegiada. Suponnha que R é juntada por igualdade pelo seu campo chave a alguma tabela S, então R alcança S. A propriedade alcançar é transitiva.***
- ***Portanto, não existirão duplicatas entre os registros retornados, mesmo na ausência de DISTINCT, se:***
  - ***Toda tabela mencionada no select é privilegiada***
  - ***Toda tabela não privilegiada alcança no mínimo uma privilegiada.***

Exemplo Prático

- Empregado(ssnum, nome, gerente, dept, salário, numamigos)
  - Índice de agrupamento sobre ssnum
  - Índice não agrupado (i) sobre nome e (ii) sobre dept
  - Ssnum determina todos os outros atributos
- Estudante(ssnum, nome, grau_estudado, ano)
  - Índice de agrupamento sobre ssnum
  - Índice não agrupado sobre nome
  - Ssnum determina todos os outros atributos
- Tech(dept, gerente, local)
  - Índice de agrupamento sobre dept; dept é chave.

Alcance: Exemplo 1

- SELECT ssnum
  FROM empregado, tech
  WHERE empregado.gerente = tech. gerente

- Um mesmo registro empregado pode casar vários registros tech (porque gerente não é uma chave de tech), sendo assim o ssnum desse registro de empregado pode aparecr várias vezes.
- Tech não alcança a relação privilegiada empregado.

Alcance: Exemplo 2

- SELECT ssnum, tech.dept
  FROM empregado, tech
  WHERE empregado.gerente = tech.gerente

- Cada repetição de um dado valor ssnum seria acompanhado por um novo tech.dept já que tech.dept é uma chave de tech
- As duas relações são privilegiadas.

Alcance: Exemplo 3

- SELECT estudante.ssnum
  FROM estudante, empregado, tech
  WHERE estudante.nome = empregado.nome
  AND empregado.dept = tech.dept;

- Estudante é privilegiada?
- Empregado alcança Estudante?
- Precisa de DISTINCT?