- 1º Semestre -

INTRODUÇÃO

Definição e Objectivos

Uma BD é por definição um conjunto organizado de dados disponível a todos os utilizadores ou processamentos da organização que deles tenham necessidade.

A tecnologia de BD tem dois objectivos:

- ✧ Dar corpo a uma forma mais natural de pensar sistemas de informação;
- ✧ Disponibilizar os meios de desenvolvimento de mais altos níveis que acelerem o processo de desenvolvimento de novos sistemas e facilitam a manutenção de sistemas construídos segundo esta tecnologia.

Dados *vs* Informação

Dados são apenas elementos ou valores discretos que isoladamente não tem qualquer valor. Só se transforma em informação quando são relacionados e interpretados. A informação é o resultado de alguma forma de processamento sobre os dados.

A informação é um dos recursos mais importantes de uma organização. Para que a informação seja utilizada de um modo eficaz pela organização tem que verificar simultaneamente algumas condições:

Actualidade O valor da informação depende de ela ser actual ou não. Devido ao dinamismo que se verifica no ambiente empresarial, o período de validade de informação é cada vez mais curto.

Relevância A informação deve ser filtrada para que apenas a informação relevante para cada situação seja considerada.

Correcção É necessário que a informação seja correcta e rigorosa para se poder decidir com mais confiança.

Disponibilidade A utilidade de uma informação depende também da rapidez com que ela é disponibilizada. Pois o processo de tomada de decisão tem que ser quase instantâneo. Informação disponibilizada tardiamente não é útil.

Legibilidade A informação só é informação se poder ser interpretada. A forma como a informação é interpretada também é muito importante.

A informação será actual e correcta se os dados forem precisos e actualizados; a relevância, disponibilidade e legibilidade da informação dependem dos meios utilizados para o processamento dos dados.

A tecnologia de BD contribui de forma significativa para viabilizar estes requisitos, contribuindo para a qualidade da informação fornecida e melhorando os serviços prestados pelas tecnologias de informação.

Sistemas de Gestão de Ficheiros

Nos SGF os dados são armazenados em ficheiros diferentes e alturas diferentes e recolhidos por diferentes aplicações. Deste modo, existem grandes probabilidades dos dados se contradizerem.

Uma das características principais destes sistemas é que os programas de aplicação e os ficheiros de dados se encontram ligados por interfaces físicos e para cada aplicação e cada ficheiro deve existir uma interface física específica, assim estes sistemas surgem como ilhas isoladas entre si.

É evidente que, deste modo, a partilha de dados num SGF apresenta problemas ao nível de manutenção dos próprios sistemas e mais grave do que isso, os problemas dos acesos aos dados partilhados tem que ser

resolvidos ao nível das aplicações o que põe em causa a fiabilidade dos sistemas; por fim como estas aplicações interagem com o SO para aceder aos ficheiros, é necessário trabalhar a baixo nível o que faz com que o esforço requerido para o desenvolvimento dos sistemas implique uma baixa de produtividade de trabalho.

Sistema de Bases de Dados

A abordagem pelos SBD tem uma característica fundamental, os dados estão organizados num único conjunto. O acesso aos dados passa por uma entidade chamada de **Sistemas de Gestão de Base de Dados** (SGBD) que centraliza em si o acesso físico em si à BD. As aplicações apenas têm uma interface que é lógico e não físico com o SGBD. A BD encontra-se armazenada nalgum tipo de memória de características não voláteis mas de forma transparente aos utilizadores e a todo o nível aplicacional. Deste modo o SGBD é a única entidade que manipula a BD.

Esta interface lógica é conseguida à custa do armazenamento dos dados e das suas descrições (metadados) numa entidade chamada dicionário de dados que actua como um filtro que permite ao SGBD interpretar os dados armazenados.

Desta forma, o mesmo ficheiro de dados pode ser interpretado de modo diferente, dependendo da aplicação. Esta é uma diferença para os SGF que têm uma interface físico onde estão armazenados os detalhes e a organização física dos dados.

O principal ganho desta tecnologia foi a separação entre os programas e os dados.

Modelos de Dados

Com a tecnologia da SGBD surgiram actividades importantes para o desenvolvimento destes sistemas como sendo a modelação de dados. A BD existente deve servir as aplicações presentes mas também aquelas que possam a vir a ser desenvolvidas no futuro. Por isso, é necessário que a identificação dos dados necessários para a BD dependa essencialmente dos requisitos do SI que a BD pretende suportar. A modelação dos dados pega em todos os requisitos do SI e tenta encontrar um modelo (modelo conceptual de dados) que traduza a estrutura lógica dos dados que satisfaçam esses requisitos. Este modelo será mais tarde traduzido para uma forma particular de BD, podendo esta ser hierárquica, em rede, relacional, orientada objecto, etc.

SISTEMA DE GESTÃO DE BASES DE DADOS

O SGBD é um conjunto de software complexo, destinado a gerir todo o armazenamento e manipulação dos dados do sistema, servindo de interface entre a BD e o nível aplicacional. O SGBD é responsável pela partilha, segurança e recuperação dos dados.

Mais do que esconder os detalhes do armazenamento dos dados, o SGBD fornece ao nível aplicacional um nível abstracto de tal modo elevado que certos modelos de BD trabalham já a um nível muito próximo do mundo-real.

Arquitectura ANSI/SPARC

Esta arquitectura tenta estabelecer um padrão para o desenvolvimento de tecnologias de BD. Esta arquitectura foi desenvolvida em 3 níveis diferentes, cada um deles descrevendo a BD num nível de abstracção diferente.

- Nível interno – Este nível está implicitamente relacionado com as estruturas de armazenamento dos dados, e a principal preocupação aqui é tentar proporcionar o melhor desempenho possível em todas as operações e àquelas que ocorram com mais frequência.
- Nível conceptual – A este nível esta representado o esquema da estrutura de dados, independente de qualquer utilizador ou aplicação particular, é esta camada que esconde do nível aplicacional os detalhes da implementação física do ficheiros onde os dados estão armazenados.
- Nível externo – Aqui, e para cada utilizador, é estabelecida uma janela sobre o esquema conceptual, que lhe permite trabalhar sobre a parte dos dados que lhe dizem respeito, isto acontece também com as aplicações.

Esta arquitectura obtém deste modo algumas característica importantes:

- Independência física – É possível fazer alterações relativas ao modelo interno, sem por isso que mudar o modelo conceptual (ex. organização de ficheiros). Alterações de nível interno não se repercutem no conceptual.
- Independência lógica – É possível, na maioria dos casos, fazer alterações conceptuais sem alterar o nível externo. As alterações no nível conceptual não afectam as vistas estabelecidas no nível externo.

O SGBD tem a função de ligar estes 3 níveis, mapeando cada nível e armazenado este mapeamento no dicionário de dados. Alem disso, as características de independência física/independência lógica, originam uma outra característica importante: a independência programas/dados, em que as alterações que envolvam a estrutura de dados não obrigam a alterações no nível aplicacional.

Transacção

Há situações em que várias operações sobre a BD só fazem sentido se executadas como um todo. Surge assim o conceito de transacção que é um conjunto de operações sobre a BD perfeitamente limitado.

Características das transacções:

- **Atomicidade** – as operações que constituem uma transacção forma um grupo indivisível. Ou todas são executadas com sucesso ou nenhuma é executada.
- **Integridade** – uma transacção deve transportar a BD de um estado de integridade para outro estado de integridade.
- **Isolamento** – as transacções executam concorrentemente mas o sistema deve dar a ilusão a cada transacção que é a única a executar. Deve garantir que estas não interfiram entre si.
- **Persistência** – os efeitos provocados por uma transacção bem sucedida devem se tornar persistentes na BD e visíveis a outras transacções. Deste modo os efeitos sobre a BD não podem ser desfeitos senão por outras transacções posteriores.

Tipos de transacção As transacções mais simples são chamadas *flat transactions* que consiste apenas em delimitar o conjunto de operações entre duas instruções:

```
BEGIN TRANSACTION
…
```

```
<Operações sobre BD>
…
END TRANSACTION
```

São *flat transactions* porque só existe um nível de controlo, ou seja, entre BEGIN TRANSACTION E END TRANSACTION ou todas as operações sucedem ou nenhuma sucede. Quando o corpo das operações for grande e ocorrer um erro perto do fim da transacção é necessário corrigir o trabalho efectuado até aí. Isso leva a que exista algum beneficio em utilizar transacções que evitassem o tudo ou nada. Existem três tipos mais elaborados de transacções:

save points – são transacções em que existem pontos intermédios que dividem a transacção em várias partes, permitindo o *rollback* ou *rollfoward* antes da transacção terminar.

```
BEGIN TRANSACTION
…
<Operações sobre BD>
…
SAVE POINT <x>
…
<Operações sobre BD>
…
END TRANSACTION
```

chained transactions – são similares às *save points*, mas nestas estes pontos correspondem a *commits*, formando uma transacção com vários *commits* intermédios. As *save points* são mais flexíveis sobre os processos de *rollback* e *rollfoward*. As *chained transactions* evitam perder toda a informação guardando o trabalho até ao último ponto do *commit*.

```
BEGIN TRANSACTION
…
<Operações sobre BD>
…
COMMIT POINT
…
<Operações sobre BD>
…
END TRANSACTION
```

nested transactions – é mais flexível e corresponde a uma árvore de transacções (subtransacções). Estas subtransacções ou são *flat transactions* ou são elas próprias *nested transactions*. Existem alguns pressupostos no controlo das *nested transactions*:

- Cada subtransacção pode desfazer-se ou finalizar mas o seu *commit* só é efectivo quando a sua transacção pai tiver finalizado.
- O *commit* de uma subtransacção apenas torna visíveis os seus efeitos à transacção pai.
- Quando uma subtransacção é desfeita todas as suas transacções são desfeitas.

Requisitos fundamentais de um SGBD

Segurança O objectivo das medidas de segurança é proteger os dados armazenados de acessos não autorizados, garantindo que apenas os

utilizadores autorizados acedem ao sistema. A segurança pode assumir duas perspectivas:

- Segurança física – este tipo de segurança implica que o sistema esteja fisicamente fora do alcance de pessoas não autorizadas. Actualmente devido à dispersão dos sistemas informáticos isto é praticamente impossível.
- Segurança lógica – os recursos do sistema são protegidos através de mecanismos lógicos de controlo de acessos não autorizados (*usernames*, passwords, …). Este controlo é limitado e deve ser mais sofisticado definindo não só quem tem acesso mas também a quê e como pode aceder.

Em relação aos utilizadores autorizados é preciso definir os limites das suas autorizações, ou seja, ao que é que têm acesso e o que podem fazer (perfil do utilizador). Este perfil define-se à custa de dois tipos de restrições:

Vistas – a que partes da BD pode aceder.

Regras de autorização – de que forma pode aceder à BD.

Outra medida de segurança é dissuadir os utilizadores de aceder a zonas proibidas ou efectuar operações não autorizadas, para isso o utilizador identifica-se perante o sistema e as operações que realiza podem ser registadas. Uma outra medida de segurança, mais radical, passa por encriptar os dados. O SGBD deve fornecer os mecanismos para codificar/descodificar os dados. O sistema fica deste modo mais pesado.

É impossível obter uma protecção absoluta contra acessos não autorizados. Deve-se estabelecer um nível de segurança equivalente ao valor dos dados a proteger de tal forma que o risco e os custos das tentativas de acessos não autorizados sejam superiores aos benefícios desses acessos.

Integridade Uma BD encontra-se num estado de integridade se contem apenas dados válidos que contradizem a realidade que esta representa. A manutenção da integridade pressupõe proteger a BD de acessos menos válidos que executem operações que ponham em risco a correcção dos dados armazenados.

Uma BD vai evoluindo ao longo de estados ditos de integridade sempre que uma transacção que envolva alterações ao conteúdo da BD seja finalizada com sucesso. Assim, apenas as operações de actualização podem por em risco a integridade da BD. As actualizações são governadas por um conjunto de regras de integridade que definem o que é e o que não é válido. Podem ser classificadas em dois grupos:

Implícitas – São próprias de cada modelo da BD e impedem actualizações que ponham em causa o funcionamento do modelo.

Explicitas – são restrições de integridade semântica que são próprias da realidade modelada e independentes do modelo de BD.

Restrições estáticas → estabelecem as condições de validade para os estados, ou seja, as condições que se devem verificar para que um estado seja considerado de integridade (“o salário de um funcionário deve ser >= ao salário mínimo”)

Restrições dinâmicas → definem as transições de estado permitidas, impedindo transições inválidas (“o salário de um funcionário nunca pode descer apenas aumentar”)

O levantamento das restrições de integridade é feito na fase de modelação de dados e faz parte do modelo conceptual. Assim, as restrições de integridade são mantidas e coordenadas pelo SGBD e partilhadas a todo o nível aplicacional. A maior parte dos SBD são pobres em termos de

tratamento de integridade mesmo ao nível das restrições implícitas. Relativamente às explícitas a maioria dos SGBD não os suporta.

Controlo da concorrência relaciona-se com a coordenação da partilha dos dados armazenados pelas várias aplicação e utilizadores, tratando-se por isso de um problema específico dos SBD multiutilizador. Deve garantir que cada aplicação ou utilizador interaja com a BD como se fosse o único a utilizar os seus serviços. A unidade básica do controlo da concorrência é a transacção. As transacções podem ser executadas de duas formas:

Execução em série – as transacções são executadas sequencialmente, um só se inicializa quando a anterior tiver finalizado. A concorrência, neste caso, deixa de existir e há um nível baixo de utilização do sistema.

Execução concorrente – as execuções são combinadas e intercaladas segundo as suas operações de leitura e escrita no sentido de maximizar a utilização do sistema.

É necessário garantir que o resultado final da execução em concorrência seja o mesmo que se as transacções fossem executadas em série, assegurando, deste modo, a integridade da BD. A execução de duas transacções concorrentemente, sem coordenação, pode originar resultados incorrectos. É necessário, por isso, definir procedimentos e mecanismos que impeçam duas transacções concorrentes de acederem simultaneamente ao mesmo conjunto de dados, envolvendo a actualização dos mesmos.

Mecanismos de coordenação

Escalonamentos serializados

O escalonamento é a ordem pela qual, as operações leitura e escrita de um conjunto de transacções concorrentes, são executadas. Para o mesmo conjunto de transacções podem surgir diferentes escalonamentos e destes alguns conduzem a BD para estados de integridade sucessivos (escalonamentos concorrentes serializados). Para testar a serialização de um escalonamento utilizam-se grafos de precedências que são representações gráficas das dependências entre as transacções (dependências de I/O). O grafo de precedências é constituído por:

Nodos – representam as transacções

Arcos orientados – representam dependências de I/O

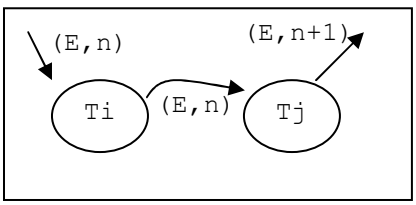
Cada elemento da BD evolui ao longo de várias versões, este é o conceito de versão de um elemento:

(E,1)→ quando o elemento é criado.

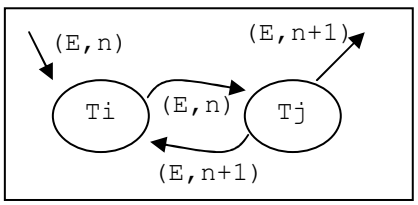
(E,n)→ quando esta num estádio da sua evolução.

Quando a BD muda de estado significa que pelo menos um dos seus elementos evoluiu para uma nova versão.

Se o grafo de precedências de um escalonamento não contiver ciclos significa que as transacções não interferem entre si e este é serializado.



- Grafo de precedências serializado



- Grafo de precedências não serializado

Existem três tipos de dependências de I/O:

A transacção Ti lê o elemento E da BD que mais tarde Tj actualiza.

A transacção Ti actualiza o elemento E da BD que mais tarde Tj lê.

A transacção Ti actualiza o elemento E que mais tarde Tj também actualiza.

Métodos de controlo da concorrência

Como o processo de serialização é muito completo não se testa sequer se um escalonamento é ou não serializado. Ao contrário de criar estes escalonamentos utilizam-se métodos que impeçam a ocorrência de escalonamentos não serializados. Existem três destes métodos:

Métodos utilizando mecanismos de locking – existem dois tipos de *locking*, de leitura e de escrita. Já que o mesmo dado pode ser lido por várias transacções *ao mesmo tempo* sem alterar o seu valor podem existir simultaneamente várias transacções com *locks* de leitura sobre o mesmo elemento. Os *locks* de escrita dão acesso exclusivo, não podem existir dois *locks* sobre um mesmo elemento se um deles for de escrita.

Métodos utilizando mecanismos de etiquetagem – a cada transacção é atribuído um valor único. Se houver um conflito a transacção com menor valor (mais antiga) tem prioridade e a outra é desfeita sendo-lhe atribuída uma nova etiqueta. O controlo da concorrência neste método existe ao longo de todo o período de vida de cada transacção.

Métodos optimistas – partem do pressuposto que as interferências são raras e o sistema não deve ser sobrecarregado com tarefas de gestão e sincronização de transacções. Apenas quando a tarefa termina é que se verifica se o seu *commit* traz problemas em relação às outras transacções. O sistema verifica se as operações de leitura da transacção (read-set) colidem com as operações de escrita das outras transacções, verifica também se as operações de escrita da transacção (write-set) colidem com as operações de leitura das outras transacções. Este método só é viável se realmente não existirem muitas interferências de outro modo a sobrecarga de trabalho torna o método impraticável. O controlo da concorrência neste método existe apenas no fim do período de vida de cada transacção.

No método de etiquetagem sempre que uma transacção acede a um elemento de dados marca-o com o seu número de etiqueta. Quando uma transacção tenta aceder a um elemento de dados cujo valor de etiqueta é superior ao seu é necessário desfazer e reiniciar a transacção com um novo valor de etiqueta. Se o valor de etiqueta for igual ou inferior ao valor de etiqueta da transacção o acesso é permitido.

Existem mecanismos mais elaborados de etiquetagem que usam duas etiquetas por cada elemento de dados: uma de leitura, que contem o valor da transacções mais recente que leu este valor; e outra de escrita, que contem o valor mais recente da transacção mais que actualizou esse elemento.

Os *lockings* são os mecanismos mais utilizados para controlo de concorrência. Estes mecanismos assumem que antes de aceder a um elemento da BD é necessário obter o *lock* desse elemento. Os *lock* mais elementares são os binários. Estes podem assumir dois valores: *locked(1)*, significando que o elemento não pode ser acedido; ou *unlocked(0)*, significa que qualquer transacção que necessite aceder ao dado deve fazer o seu *locking* (colocar a 1) e depois aceder ao dado. A grande desvantagem do *lockings* binários é que apenas uma transacção pode aceder de cada vez a um dado mesmo que existam outras transacções apenas de leitura. Para colmatar esta desvantagem existem tipos mais elaborados de *locking*. Os *locks* de leitura/escrita permitem que duas ou mais transacções acedam ao dado desde que sejam de leitura. Apenas os acessos de escrita são

exclusivos. Um elemento de uma BD pode estar em um de três estados: *locked para leitura*; *locked para escrita*; ou *unlocked*. Os *locks* de leitura e escrita vão diminuindo de valor até que chegam a 0, ficando *unlocked*. Outra abordagem corresponde a criar uma nova versão do elemento se há intenção de actualizá-lo. Os acessos de leitura actuam sobre a versão original, os acessos de escrita actuam sobre uma cópia do elemento. Assim, os acessos de leitura nunca são bloqueados e os de escrita apenas são bloqueados para transacções de escrita do mesmo elemento.

Os mecanismos de controlo de concorrência usando *lockings* podem criar alguns problemas graves: o *deadlock* acontece quando duas transacções se bloqueiam uma à outra para o mesmo elemento. O risco de *deadlock* é maior quanto maior for o número de transacções a executar. O risco de *deadlock* existe se permite que uma transacção bloqueada mantenha os seus *locks* enquanto espera por outros recursos. Existem duas maneiras de resolver este problema:

- Uma transacção só pode iniciar quando obtiver todos os *locks* que necessita, sendo estes *locks* realizados todos ao mesmo tempo. Ou a transacção realiza todos os *locks* ou não realiza nenhum.
- Deixar os *deadlocks* acontecer e quando a situação for detectada uma das transacções é abortada libertando os seus *locks* para tentar resolver o problema.

Se uma transacção necessitar de muitos *locks* pode nunca ser realizada ou pala segunda abordagem ser continuamente abortada (*livelock*). Como o *deadlock* ocorre raramente, é preferível deixar a situação acontecer e depois de detectada tentar resolver o problema.

Os *deadlocks* podem ser resolvidos com *timeouts* que correspondem ao tempo máximo que uma transacção pode ficar bloqueada, ao fim desse tempo a transacção é desfeita. O problema é que uma transacção grande pode ser confundida com uma transacção bloqueada sendo desfeita.

Os grafos de espera são outra situação de detectar *deadlocks* que tenta representar as transacções bloqueadas por outras transacções. Se existirem ciclos neste grafo existe um *deadlock*. Se existir algum ciclo o sistema selecciona uma transacção e faz o seu *rollback*.

Por fim, outro método de detectar *deadlocks* é o *two-phase locking* (2PL). Segundo este método o processamento de uma transacção faz-se em duas fases: a primeira corresponde à obtenção de *locks*, que termina quando acontece o primeiro *unlock*; e a segunda fase, onde se libertam todos os *locks*. Não é permitido a uma transacção 2PL libertar um *lock* e depois obter outro. Uma transacção é 2PL se todos os seus *locks* antecedem todos os seus *unlocks*. No controlo de concorrência segundo 2PL, e para que o desempenho seja maximizado, a obtenção dos *locks* é feita à medida que os dados vão sendo necessários e não todos de uma vez. A maioria dos SGBD apenas liberta os seus *locks* quando uma transacção faz o seu *commit*. Isto acontece para evitar o *rollback* em cascata em que se uma transacção é abortada isso pode implicar o *rollback* das transacções que lhe precedem.

Existem situações em que é aceitável que haja interferências entre transacções desde que o resultado final não seja afectado pelas outras transacções. Os SGBD tentam disponibilizar ao nível aplicacional vários níveis de isolamento dependendo das situações concretas e em função das necessidades de isolamento das transacções. Quanto mais baixo o nível de *locking* (isolamento) maior a concorrência permitida; quanto mais alto o nível de *locking* menor a concorrência permitida.

Recuperação/Tolerância a falhas é a actividade que tem por objectivo restaurar a BD para um estado de integridade garantido após a ocorrência de qualquer falha. O SGBD deve dispor de mecanismos que suportem a recuperação da BD de falhas graves como a perda de toda a BD assim como pela falha de uma transacções individual. Os mecanismos de recuperação de SGBD baseiam-se na utilização de formas de redundância que quase duplicam a BD. Existem dois destes mecanismos.

Backups – são cópias de segurança que abrangem toda a BD e tentam recuperar esta de falhas catastróficas. A grande desvantagem é que esta recuperação é limitada ao momento quando o *backup* foi feito perdendo-se deste modo as actualizações seguintes. Outra desvantagem é que este processo obriga à paralisação de todo o sistema. A sua periodicidade deve depender do valor dos dados e da frequência com que são alterados.
Transaction logging – vêm eliminar a principal fraqueza dos *backups* ao manter o registo das operações efectuadas à BD após o *backup*.
Enquanto o *backup* actua ao nível de toda a BD o *transaction log* actua ao nível das transacções.

Existe um tipo de falhas cuja solução implica não o refazer mas o desfazer da BD. Se uma transacção não terminar normalmente devem-se desfazer os seus efeitos intermédios sobre a BD (*rollback*). O *transaction log* deve armazenar os valores referentes a estas transacções (*before-images*). Estes dois mecanismos implementam duas características das transacções, atomicidade ao desfazer os efeitos de uma transacção não sucedida e persistência ao refazer os efeitos de todas as transacções bem sucedidas. Como os *transaction logs* são armazenados em memória central (volátil), em zonas chamadas *buffers*, em caso de falha do sistema esses *buffers* são perdidos, o que pode levar a perdas efectivas de dados. Para que isto não aconteça, aquando dos *commits* de uma transacção, as suas actualizações devem ser escritas na BD para esse efeito utiliza-se o *transaction log*. Em caso de falha o *transaction log* pode ser utilizado para refazer as transacções cujos efeitos ainda não tenham sido passados da memória não volátil. Esta abordagem designa-se por *write-ahead logging* que corresponde à escrita dos efeitos de um *commit* no *transaction log* antes da BD ser actualizada. Se acontecer uma falha neste momento apenas as transacções activas serão perdidas.

Dependendo do tipo de falhas existem diferentes métodos de recuperação:

- **Falha de disco**, é necessário carregar o último *backup* e actualizar a BD utilizando as *after-images* do *transaction log* (*rollfoward).*
- **Falha de sistema**, nestas situações a memória central está corrompida devendo voltar-se a um estado anterior válido. Com um *rollback* recuperam-se as *before-images* das transacções e através das suas *after-images* repõem-se a BD num estado válido anterior à falha. O problema é que o *rollback* deve recuar até um momento de sincronização do *transaction log* com a BD. Para isso o sistema executa *check points* que correspondem a marcas no *transaction log* em que os *buffers* em memória central foram escritos no disco. Os *check points* identificam quais as transacções a desfazer e quais as a refazer, limitando a amplitude dos *rollbacks* e *rollfowards*. Existem quatro situações na utilização dos *check points:*
  (1) As transacções iniciadas e finalizadas antes do último *check point*
  (2) As transacções iniciadas e finalizadas após o último *check point*

(3) As transacções iniciadas antes do último *check point* e finalizadas após o último *check point*
(4) As transacções iniciadas antes ou após o *check point* e nunca finalizadas.

Para (4) as transacções devem ser desfeitas; as do (1) estão salvas; as do (2) e (3) devem ser alvo de recuperação.

⌛ **Falha de transacção**, basta recorrer ao *transaction log* e recuperar as *before-images* da transacção.

Os mecanismos de recuperação trazem custos para a sua manutenção:
Acréscimo nos acessos a disco, porque é necessário constantemente actualizar os ficheiros de actualização, limitando o desempenho dos SGBD.
Acréscimo das necessidades de espaço, pois é necessário adicionar espaço para suportar os mecanismos de recuperação.
Sobrecarga de processos em termos de utilização de CPU (este aspecto não é muito significativo).

SBD vs SGF

As diferenças mais importantes entre as duas abordagens são:

Abstracção de dados Com a tecnologia de BD não é necessário a nível aplicacional conhecer os pormenores dos dados pois o utilizador/programador trabalha com um modelo abstracto de dados.
Independência programas/dados Nos SGF a definição de dados faz parte das aplicações. Nos SBD as aplicações são escritas independentemente dos ficheiros onde os dados estão armazenados.
Partilha de dados Nos SBD existe uma entidade coordenadora enquanto que nos SGF as aplicações controlam a partilha dos dados.
Diminuição da redundância Nos SBD a redundância pode ser controlada e mais facilmente mantida.
Controlo nos acessos aos dados Nos SBD são definidas vistas para os utilizadores dando-lhes autorizações de trabalho segundo as suas necessidades.
Desenvolvimento + Manutenção O SGBD esconde dos utilizadores os detalhes da implementação de BD. Não se perde tempo com os pormenores irrelevantes às soluções dos problemas melhorando a manutenção das aplicações existentes.
Integridade dos dados Cada vez mais o SGBD verifica a correcção dos dados submetidos no sistema. Nos SGF as aplicações têm esta responsabilidade por completo.
Mecanismos de recuperação a falhas os SBD têm mecanismos que fazem face a qualquer tipo de falha ao contrário dos SGF que são mais vulneráveis diminuindo a sua fiabilidade.
Questões *ad hoc* Os SGBD fornecem ferramentas em que é possível obter respostas a questões directamente da BD em *run-time*.
O SGBD é uma peça de software complexa e mais dispendiosa. Para além das acções de formação necessárias para a sua utilização.

Utilizadores de sistemas de BD

Existem duas classes de indivíduos perante o sistema:
☺ **O administrador da BD** (ADB) responsável máximo pelo funcionamento do sistema e cujas responsabilidades são: a especificação do esquema conceptual da BD, definir procedimentos de *backup* e recuperação, e controlar a segurança dando e retirando acessos aos utilizadores.
☺ **Utilizadores** podem basicamente ser de dois níveis:

- *Finais* podem consultar, alterar, ou remover dados. Os menos sofisticados limitam-se a correr aplicações sobre o sistema e interrogá-lo para obter respostas às suas perguntas.
- *Programadores* utilizam linguagens de alto nível e têm vistas sobre o esquema conceptual da BD para poderem desenvolver aplicações para os utilizadores finais.

Linguagens de BD

O SGBD deve proporcionar linguagens apropriadas para as necessidades de cada tipo de utilizadores. Assim, temos dois tipos de linguagens:

Linguagem de definição de dados Não é uma linguagem de programação pois não possui instruções específicas de processamento. É uma notação que descreve a estrutura dos dados (linguagem de definição dos metadados). O resultado da sua execução é armazenado no dicionário de dados.

Linguagem de manipulação de dados (LMD) Linguagem disponibilizada ao nível aplicacional para manipular os dados. As operações possíveis são: inserção, remoção, alteração e consulta dos dados.

ORGANIZAÇÃO E ADMINISTRAÇÃO DE DADOS

Hierarquia de memórias

Em termos de hardware um sistema de computação pode ser dividido em três tipo de componentes, unidade de processamento (CPU), memória e dispositivos de I/O (periféricos).

Existem dois tipos de memória, a memória central (RAM) é volátil e apenas mantém os seus dados enquanto for alimentada por energia; a memória secundária usualmente discos mantém o seus conteúdo por períodos de tempo longos. Características das memórias:

Capacidade Memória central está em situação de inferioridade em relação à memória secundária e tem limitações de crescimento relacionadas com o número de linhas de endereço e na maioria dos casos não é suficiente para conter todos os dados.

Custo Um bit de memória central custa mais que um bit de memória secundária e as necessidades de armazenamento de uma BD típica só tornam viável o armazenamento da BD em memória secundária.

Volatilidade A memória central é volátil.

Tempo de resposta A maior desvantagem da memória secundária é o seu tempo de resposta, sendo a memória secundária o principal "gargalo" de qualquer sistema.

Funcionalidade Os dados não podem ser processados directamente em memória secundária e devem ser enviados para a memória central. Esta característica torna a memória central insubstituível.

Por sua vez a memória pode ser classificada por três classes:

- **On-line**, são dispositivos, de memória secundária, com acesso aleatório, como por exemplo os discos (magnéticos, ópticos, …)
- **Near-line**, são relativamente recentes, dispõe de uma ou várias unidades de leitura em que braços mecânicos colocam e retiram discos de acordo com as solicitações que chegam (*disk jukeboxes*).
- **Off-line**, são dispositivos, de memória secundária, de acesso sequencial, como por exemplo as bandas magnéticas.

Podemos distinguir entre, memórias on-line e near-line utilizadas para armazenar a BD actualizada e consultada em tempo real; e memórias off-line utilizadas para suportar mecanismos de recuperação/tolerância a falhas como os backups.

Existe, portanto, no sistema uma hierarquia de memórias que vai desde os registos internos do processador até à memória off-line. O custo e a rapidez destas memórias aumentam com a profundidade.

Dos factores que distinguem a memória central da memória secundária é o tempo de resposta. O tempo de acesso à memória secundária é muito superior ao tempo de acesso à memória central e o objectivo aqui é minimizar o número de acessos à memória secundária.

A superfície dos discos esta organizada em pistas e estas em sectores. Qualquer acesso ao disco envolve a transferência de um conjunto de bytes (bloco de I/O). O tempo total de transferência de um bloco para a memória central depende um conjunto de factores, sendo o mais importante o tempo de médio de leitura/escrita. Este é constituído por 3 parcelas:

- **Tempo médio de posicionamento** que é a maior parcela de tempo e corresponde ao tempo que demora a colocar a cabeça do disco na pista pretendida.
- **Atraso rotacional médio** que corresponde ao tempo de que a cabeça tem de esperar pela rotação da pista.

- **Tempo de transferência de bloco** depende número de blocos a transferir.

Os blocos transferidos para a memória central ficam armazenados o mais tempo possível *buffers*. O conteúdo de um bloco de disco armazenado em *buffer* é designado *página*.

Sempre que o SGBD solicita um dado, se o dado se encontrar em buffer, não é necessário aceder à memória secundária. Através de uma gestão correcta do buffer, o SGBD pode assim melhor o seu desempenho.

Gestão de *buffers*

Algumas partes da BD estão constantemente a ser solicitadas, são bons candidatos a permanecer em *buffer*. Após uma página ter sido lida para *buffer* o objectivo é mantê-la em memória central durante o maior tempo possível. Mas como o espaço é escasso, ela terá de dar lugar a outra página proveniente do disco.

Quando uma página é substituída duas coisas podem acontecer, ou o seu conteúdo é exactamente igual ao bloco em disco (leitura) e a página é reescrita (*overwriting*); ou o conteúdo é diferente e a página deve ser substituída, antes deste processo, a página em buffer deve ser escrita em disco. *dirty-pages* são as páginas que dão origem a transferência de páginas entre memórias.

A gestão de *dirty*-pages pode ser feita segundo dói critérios:

LRU *Last Recently Used* em que o bloco menos utilizado recentemente é mais provável que venha a ser substituído.

MRU *Most Recently Used* em que o bloco mais utilizado recentemente é mais provável que venha a ser substituído.

Destes o mais eficaz parece ser o **LRU** já que segundo o princípio da localidade, as páginas mais utilizadas recentemente, devem voltar a ser utilizadas.

Métodos de acesso e organização de ficheiros

Os ficheiros são abstracções lógicas sobre os suportes de armazenamento que permite associar numa unidade lógica elementos relacionados entre si. Existem vários tipos de ficheiros, desde os não estruturados até aos estruturados que são constituídos por registos que por sua vez são constituídos por campos. Os campos têm um identificador e um tipo de dados. Cada ficheiro encontra-se armazenado ao longo vários blocos, e forma como estão organizados permite melhores ou piores desempenhos.

Também, a organização dos ficheiros determina o modo como são acedidos, existem dois métodos de acesso:

- **Sequencial** em que os registos são acedidos pela sua ordem em memória.
- **Aleatório** em que o acesso não entra em linha de conta com a ordem de armazenamento dos ficheiros.

Em relação à organização, existem três tipos de definições: ficheiros sequenciais, ficheiros indexados e ficheiros directos/hash.

Ficheiros sequenciais Os registos são inseridos em posições consecutivas dentro de cada bloco tornado o acesso sequencial muito eficaz. No caso de se pretender aceder ao ficheiro de modo aleatório isto pode tornar-se uma operação muito pesada. Para melhorar o desempenho neste caso podem-se utilizar diferentes tipos de pesquisas como por exemplo a pesquisa binária em que dado um espaço de procura acede-se ao bloco situado ao meio desse espaço, se o registo pretendido não for esse procura-se à direita ou à esquerda do bloco conforme o registo seja superior ou inferior a qualquer dos registos desse bloco e assim sucessivamente até encontrar o registo pretendido.

Genericamente, uma pesquisa binária encontra um registo de um ficheiro com **B** blocos no máximo em **Trunc($\log_2$(B)+1)** acessos ao disco.

Ficheiros indexados São ficheiros constituídos por dois campos, um deles é a chave (para pesquisa), outros é um apontador para o endereço em memória do dado (ficheiros de dados). A um mesmo ficheiro de dados podem estar associados vários ficheiros de índices. Embora um grande número de ficheiros de índices torne mais eficiente o acesso aleatório aos dados por outro lado torna as operações de actualização mais pesadas por implicarem a alteração de vários ficheiros de índice. Uma implementação mais eficiente de ficheiros indexados é estruturando os ficheiros de índices sobre a forma de árvore. Nas árvores B+ cada nodo ocupa uma página de memória e apenas o último nível contém todos os valores da chave de indexação. A profundidade de uma árvore B+ depende da dimensão do bloco de disco, da dimensão da chave e do número de registos ocupado pelo ficheiro de dados.

O número de níveis de uma árvore B+ é dado aproximadamente por **Trunc($\log_f$(N)+1)** em que **f** é o número de registos de índices por página do último nível e **N** o número de registos do ficheiro de dados. O número de testes a efectuar numa árvore B+ é sempre igual ao número de níveis da árvore.

Ficheiros directos/hash Segundo esta organização basta um acesso a disco para encontrar qualquer registo de dados segundo uma chave de pesquisa. Através de uma função de *hashing*, faz-se corresponder a cada valor da chave de pesquisa, um endereço de memória secundária. O espaço de endereçamento corresponde ao número de blocos reservados para os ficheiros de dados. Os resultados da função são endereços de bloco relativos. O endereço de armazenamento dos dados está directamente relacionado com o valor da chave de pesquisa. O problema acontece quando a diferentes valores da chave de pesquisa é dado o mesmo valor pela função de *hashing* basta que o bloco correspondente esteja lotado com registos com o mesmo valor (colisão). Estas colisões podem-se resolver por várias formas. As mais simples usam listas ligadas de blocos que partilham o mesmo endereço de *hashing* mas em endereços físicos diferentes. Estes blocos, são chamados de *blocos de overflow*, estão fora dos espaços de endereçamento da função de *hashing*. As colisões mesmo assim podem sempre acontecer e o resultado disso é o aumento de acessos a disco. Quanto maior o espaço de endereçamento menor número de colisões e mais desperdício de espaço, e vice-versa.

Análise do desempenho das várias organizações de ficheiros o acesso aleatório sem facilidades os mecanismos de procura torna-se limitador do desempenho de um sistema de BD. Para a mesma quantidade de dados o acesso sequencial é cerca de 10 vezes mais rápido que o aleatório.

Já que os acessos mais frequentes são aleatórios, nos ficheiros de BD devem utilizar organizações que facilitem este tipo de acessos. Existem dois tipos de acessos:

- **Exact match**, são pesquisas que satisfazem uma condição de igualdade (“**cod_func**=1531”);
- **Range search**, em que a pesquisa envolve uma gama de valores, por exmplo **cod_func**>5555 e **cod_func**<9999.

Uma organização de tipo *hashing* é útil apenas para acessos *exact match*. Os ficheiros indexados são mais flexíveis e adaptam-se a ambos os tipos de acesso. As estruturas preferidas para os ficheiros de índices são as árvores B+.

Desempenho *vs* flexibilidade

O SGBD deve para além de implementar um modelo de dados que representa um universo, satisfazer algumas condições especialmente de desempenho. Deve-se estabelecer um compromisso entre a flexibilidade e a viabilidade de utilização de um sistema. O tempo de resposta de um sistema depende dos seguintes factores: Tempo gasto em acessos ao disco, tempo gasto em comunicações de/para o sistema e o tempo de processamento (irrisório).

Para diminuir o tempo de acesso ao disco podem-se utilizar duas soluções, utilizar unidades de disco mais rápidas – se resolver o problema deve ser utilizada; ou introdução de redundância nos ficheiros com maiores tempos de acesso. A introdução de redundância de ficheiros chama-se *desnormalização* e como pode trazer problemas de integridade deve ser utilizada apenas quando existem fortes razoes de desempenho.

Clustering vs *Declustering*

O *clustering* consiste em armazenar fisicamente próximo dados relacionados que frequentemente são acedidos em conjunto. Então, com um nico acesso a disco podem-se recuperar os dados necessários a um determinado processamento. Para que o *clustering* funcione é necessário que os registos possuam um campo em comum – *cluster key* – com base no qual os registos vão ser armazenados. O *clustering* pode assumir duas formas, *intra-file* que armazena no mesmo bloco ou blocos contíguos registos de um ficheiro com o mesmo valor de *cluster key*; ou *inter-file* que armazena no mesmo bloco registos de ficheiros diferentes. Dado que o *clustering* beneficia os processos mais frequentes os outros podem sofrer prejuízos.

A necessidade de obter taxas de transferências maiores entre as memórias é difícil conciliar com bons acessos à memória. De facto, os dispositivos de memória com maiores capacidades têm menires taxas de transferência. As maiores taxas de transferência conseguem-se actualmente com dispositivos de disco magnético.

Ao contrário do *clustering*, o *declustering* consiste em espalhar os ficheiros da BD por vários discos para que, acedendo paralelamente, seja possível transferir maiores quantidades de dados por unidades de tempo. O *declustering* consegue aumentar tanto a capacidade de armazenamento como a taxa de transferência entre memórias.

MODELOS DE BASES DE DADOS (HIERARQUICO E REDE)

Os modelos de dados suportados pelo SGDB não conseguem traduzir completamente a riqueza semântica do modelo conceptual desenvolvido. Nesta passagem alguma informação é perdida. Existem, portanto, grandes diferenças em relação à capacidade de modelação dos vários modelos existentes.

Existem três gerações de modelos de BDs: hierárquico e de rede, relacional e novos modelos. À medida que as novas gerações foram aparecendo, o nível de abstracção aumentou e a perda de informação diminuiu.

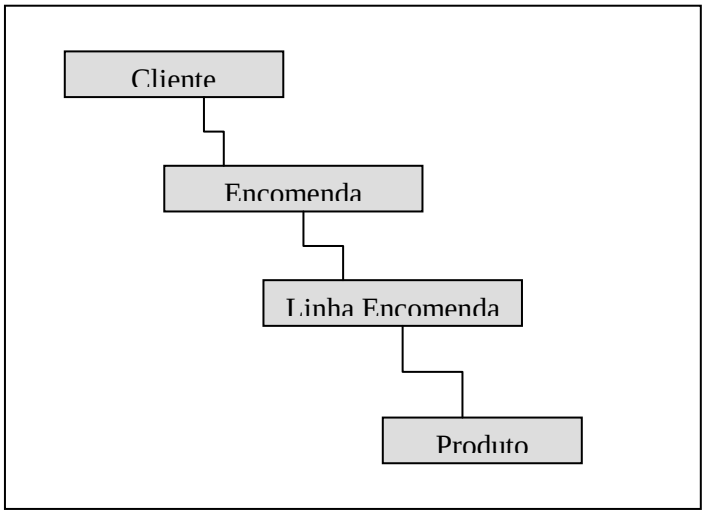
Modelo hierárquico

O modelo hierárquico resulta da evolução dos processos utilizados nos sistemas de gestão de ficheiros. É um modelo adequado para o processamento sequencial já que os dados se encontram estruturados de forma hierárquica. Este modelo tem um bom desempenho no suporte de grandes BDs mas apresenta algumas limitações relativamente ao processamento aleatório.

Conceitos

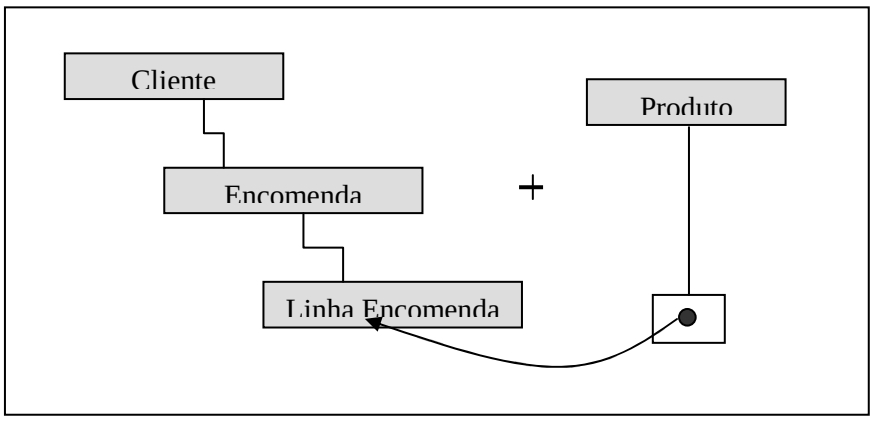
Uma BD hierárquica obedece à seguinte estrutura: *registo pai e registo filhos*. Um *segmento* designa os *nodos* das hierarquias que contêm os registos. A relação existente entre pais e filhos é do tipo 1:M em que um registo pode ser pai de vários registos mas apenas filho de um registo num segmento anterior.

Os dados armazenados podem ser visitados segundo uma sequência hierárquica do tipo *cima-para-baixo-esquerda-para-direita* (*depth-first-order*).



Esta estrutura hierárquica tem alguns problemas **graves**:

- Não é possível registar dados de um produto enquanto este não for encomendado por um cliente.
- Para aceder aos dados de um produto é necessário encontrar um cliente que o tenha encomendado.
- Sempre que um produto é encomendado os seus dados são registados (redundância). Isto traz grandes dificuldades de actualização de dados.
- A criação de várias hierarquias tenta resolver estes problemas de acessos a dados mas provoca redundância de parte da informação. Alguns sistemas para resolver este problema permitem a existência de segmentos virtuais, em cujos registos existem apontadores (*registos pai virtuais*).



Restrições de integridade

- Não pode ocorrer um registo filho que não esteja associado a um registo pai, ou seja:
  - Um registo filho só pode ser inserido se estiver ligado a uma registo pai.
  - A remoção de um registo pai remove todos os registos filho e os seus descendentes.
  - Um registo filho pode ser removido independentemente do seu registo pai.
- Se um registo filho tem dois ou mais registos pai em segmentos diferentes, apenas um dos registos será pai, os outros serão virtuais.
- Se um registo filho tem dois ou mais registos pai num mesmo segmento, o registo filho deve ser duplicado.

Manipulação de BD hierárquicas

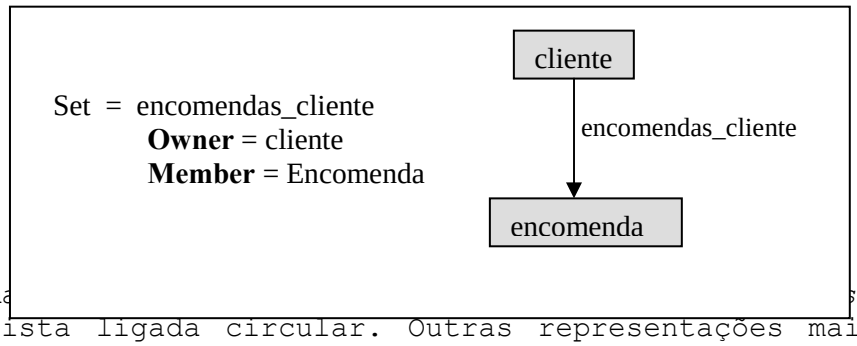
??

Modelo de rede

Este modelo corresponde a uma extensão do modelo hierárquico. Eliminando o conceito de hierarquia permite que um registo esteja envolvido em diferentes associações. No modelo em rede os registos estão organizados em grafos.

Conceitos

No modelo de rede existe um tipo de associação o *set* que define um relacionamento 1:M entre dois tipos de registos *owner* e *member*.



A forma ma *set* será, sem dúvida, a lista ligada circular. Outras representações mais elaboradas incluem a lista ligada circular duplamente ligada ou a colocação de um apontador em cada registo *member* para o seu *owner*.

Restrições de integridade

- Uma ocorrência de um *set* deverá ter um *owner* e zero ou mais *member*.
- No mesmo tipo de *set*, não é permitido que uma ocorrência, de um tipo de registo, apareça mais que uma vez, como *owner* ou como *member*. Uma ocorrência de um tipo de registo pode ser *owner* ou *member* de um ou mais *sets*, desde que de tipos diferentes.
- Nenhum tipo de registo pode ser, simultaneamente, *owner* e *member* do mesmo tipo de sets.

Manipulação de BD rede

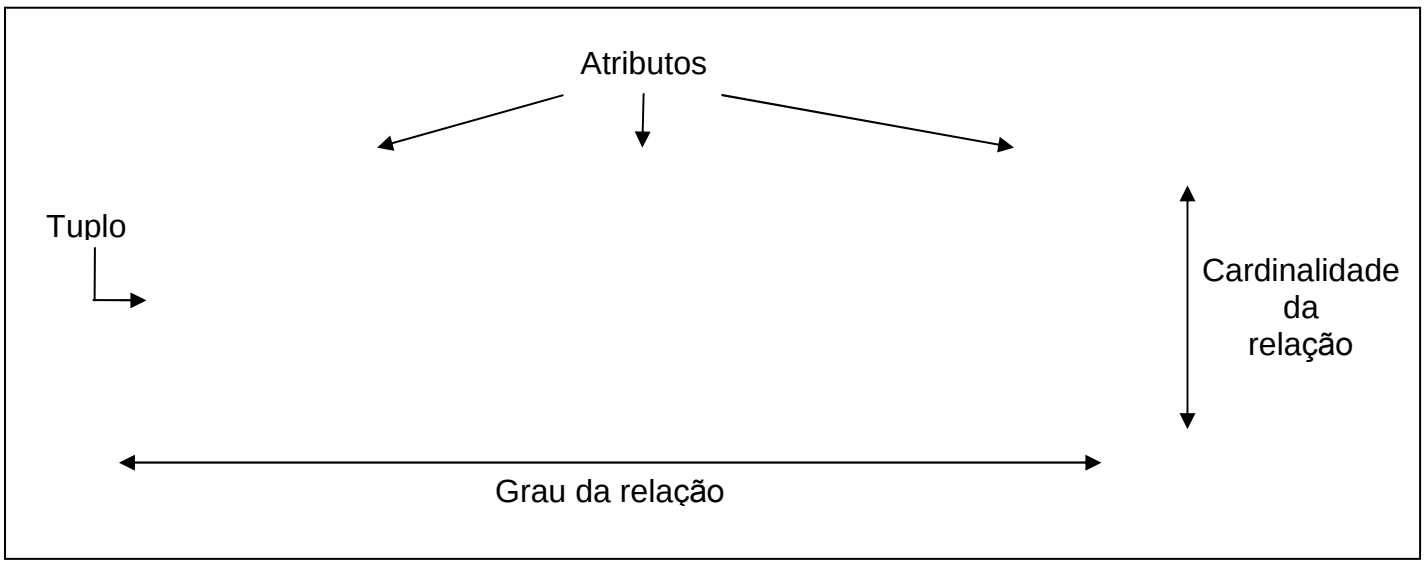
??

MODELOS DE BASES DE DADOS (RELACIONAL)

O modelo relacional foi fruto de um desenvolvimento teórico, tendo por base a teoria dos conjuntos. Neste momento, este modelo é considerado o melhor entre os convencionais mas a sua aceitação não foi fácil. Enquanto, os defensores argumentavam que ele é mais simples e flexível, os seus detractores contrapunham que o seu desempenho não permitiria a sua utilização comercial.

Conceitos

A estrutura do modelo relacional é a relação (tabela). O esquema de uma relação é constituído por atributos que definem os dados a armazenar. Um conjunto de atributos designa-se por tuplo.



Restrições de integridade

- *Integridade de domínio:* as restrições ao domínio dos atributos podem ser garantidas pelo SGBD
- *Integridade de entidade:* Deve existir uma chave primária em todas as relações, que permite identificar univocamente cada tuplo.
- *Integridade referencial:* O valore de uma chave estrangeira ou é null ou contem um valor que é chave primaria noutra relação.

Cod_Produto	Designação	Preço
1234	Monitor SVGA 15”	100
4321	Teclado português 102 teclas	35.5
2143	Impressora laser HP 4MPlus	520
3412	Rato 3 botões	5.5

Interfaces ao modelo relacional

A álgebra e o cálculo relacional foram propostos por Codd para fazer a manipulação do modelo relacional.

Dado que uma relação é um conjunto, todas as operações da teoria dos conjuntos podem ser aplicadas às relações. As operações específicas do modelo relacional incluem:

- União
- Intersecção
- Diferença

- Produto cartesiano
- Selecção
- Projecção
- Junção natural
- Divisão

Nem todas estas operações são primitivas, i.e., algumas podem ser definidas à custa de outras. Como por exemplo, a intersecção:

$$R1 \cap R2 = R1-(R1-R2)$$

Com cálculo relacional define-se o que se pretende e não como conseguir o que se pretende, ou seja, as questões são colocadas de uma forma declarativa. Dentro do calculo relacional pode ser dividido em:
- Cálculo relacional de *tuplos*
- Cálculo relacional de domínios

As 12 regras de Codd

Estas foram definidas para esclarecer o que é ou não relacional:

1. Todos os dados, incluindo o próprio dicionário, são representados de uma só forma em tabelas bidimensionais.
2. Cada elemento de dados é determinado pela combinação do nome da tabela, do valor da chave primária e do respectivo atributo.
3. Valores nulos representam informação não disponível, independentemente do domínio do atributo.
4. Metadados são apresentados e acedidos da mesma forma que os dados.
5. O SGBD relacional deve ter uma linguagem com as seguintes características:
    a. Manipulação de dados, com possibilidade de manipulação interactiva ou em programas de aplicação.
    b. Definição de dados
    c. Definição de *views*
    d. Definição de restrições de integridade
    e. Definições de acessos
    f. Manipulação de transacções
6. Numa *view* todos os dados que forem modificados devem ver essas modificações traduzidas nas tabelas base.
7. Capacidade de tratar uma tabela como se fosse um simples operando, tanto em operações de consulta como de modificação.
8. As alterações físicas dos ficheiros de dados ou nos métodos de acesso a estes ficheiros não devem afectar o nível conceptual.
9. as alterações do nível conceptual (esquema da BD)não devem afectar o nível externo.
10. as restrições de integridade devem ser especificadas numa linguagem relacional.
11. o facto de uma BD estar centralizada ou não, não deve repercutir-se ao nível da manipulação de dados.
12. se existir uma linguagem de mais baixo nível no SGBD ela não pode permitir que se ultrapassem as restrições de integridade.

Não existe ainda, nem parece provável que venha a existir um sistema que implemente as 12 regras.

A normalização

É um processo que pretende obter um esquema de BD relacional que suporte adequadamente os dados relevantes a um determinado universo.

Este é um processo sistemático que pretende solucionar algumas anormalidades que surgiam da utilização de um BD pouco normalizada. A normalização é constituída por decomposições sucessivas de relações maiores que permitam remover a redundância, dando origem a modelos de BD mais flexíveis e menos sujeitos a posteriores problemas na sua manipulação.

A normalização altera a estrutura das relações sem alterar o conteúdo, não havendo por isso perdas de informação.

Problemas de redundância

Estes problemas podem ser:

- Manutenção: A simples alteração ou remoção pode significar o acesso a várias partes da base de dados
- Espaço de armazenamento: É importante não ocupar o espaço de armazenamento com informação desnecessária.
- Desempenho: Se a houver muita redundância isso vai implicar acessos a disco para obter os mesmos dados. Mas a criação de novas tabelas também pode trazer os mesmos problemas, pois é necessário associar varias tabelas para obter os mesmos dados.

Dependências funcionais, multivalor e de junção

O processo de normalização

O processo de normalização pode ser iniciado a partir de uma única relação (*relação universal*), chegando a uma série de relações prontas a serem implementadas num BD relacional.

1ª Forma Normal: Eliminar os grupos de valores repetidos que possam existir numa estrutura não normalizada.

2ª Forma Normal: Estando uma estrutura na 1ªFN, todos os atributos devem depender da chave primária e não apenas de parte dela.

3ª Forma Normal: Estando uma relação na 2ªFN, não devem existir dependências entre os atributos de uma relação. Cada atributo pode apenas depender da chave primária da relação.

As consequências da normalização

A normalização não deve ser levada ao extremo pois a proliferação de relações tem implicações no desempenho da BD, e o processo de obtenção de informação pode tornar-se muito demorado. A normalização deve estabelecer um compromisso equilibrado entre a necessidade de integridade e um desempenho assinalável. Na maioria dos casos a normalização para entre a 3ªFN e a BCNF. Por vezes, para melhorar o desempenho é propositadamente introduzida a redundância em partes concretas do modelo.

Linguagens Relacionais

A linguagem SQL

A SQL como linguagem de manipulação de dados

Uma dais maiores virtudes da SQL é que integra facilidades que normalmente estão separadas, como controlo, definição e manipulação de dados. Ao nível da manipulação de dados, as instruções da SQL podem ser divididas em dois grupos:

- Interrogação da BD (instrução SELECT)
- Actualização da BD (instruções INSERT, DELETE e UPDATE)

A SQL como linguagem de definição de dados

No núcleo do SGDB está o dicionário de dados, assim como toda a informação necessária à gestão da BD. Estando o dicionário de dados armazenado em tabelas pode ser consultado da mesma forma que os dados, mas a sua actualização faz-se recorrendo a instruções específicas para a definição de dados (*LDD*).

As restrições de integridade impedem a ocorrência de actualizações sobre a BD que ponham em risco a sua integridade. Em relação às restrições de integridade referencial, ou o sistema impede a ocorrência de qualquer actualização que viole a integridade referencial, ou permite estas actualizações e automaticamente repõe as condições de integridade.

As limitações da SQL

Apesar de todas as virtudes da SQL, esta não é uma linguagem completa, não sendo possível resolver utilizando apenas SQL, todas as questões sobre um esquema relacional.

Relativamente ao desenvolvimento de aplicações, não dispõe de qualquer instrução de controlo de fluxo, não permite também o desenvolvimento de interfaces complexos que as aplicações apresentam. É necessário completar a SQL com os serviços de uma linguagem de programação.

Linguagens de programação e SQL

A SQL é uma linguagem que numa única operação pode manipular um número arbitrário de registos (*set-oriented*). As linguagens de programação não têm esta capacidade, tendo que recorrer a operações recursivas que permitam manipular esses registos (*record-oriented*). A SQL é uma linguagem declarativa e as linguagens de desenvolvimento de aplicação são não-declarativas. Assim, é necessário conhecer duas linguagens diferentes e o sistema deve fornecer uma interface que permita a troca de dados entre as duas. Para minimizar este problema tentou-se aproximar as duas linguagens, introduzindo instruções de controlo de fluxo (*if, then, else…*) junto da linguagem de manipulação de dados, assim a maior parte das aplicações pode ser escrita recorrendo à linguagem do SGBD.

Processamento e optimização de questões

O único motivo pelo qual os dados são armazenados e actualizados é porque muito provavelmente serão mais tarde consultados. As questões colocadas à BD podem implicar a manipulação de grandes quantidades de dados e vários acessos a disco.

A *optimização de questões* é o processo pelo qual se encontra a resolução óptima de uma questão, implicando menos custos de recursos do sistema e diminuindo o tempo de resposta do nível aplicacional.

O processamento de uma questão num sistema relacional envolve três fases:

- **Análise da questão**. Verificar se a questão pode ser executada, i.e., está de acordo com as regras da linguagem; verificar se quem colocou a questão tem privilégios para a sua execução; se tem acesso às tabelas e *views* envolvidas; etc.
- **Optimização da questão**. Avaliar os diferentes planos de execução e seleccionar o que implicar menos custos para o sistema (tempo de CPU, acessos a disco, …).
- **Execução da questão**. Aceder às tabelas e *views* envolvidas e seleccionar e executar os dados pretendidos.

O objectivo da optimização é produzir uma sequência de operações que represente a versão mais eficiente de responder à questão. Para isso, uma

questão de alto nível, é decomposta num conjunto de operações de baixo nível.

Teoricamente, é possível analisar todas as possibilidades de resolução de uma questão de alto nível recorrendo à álgebra relacional. Na prática, os sistemas resolvem as questões à custa de operações **semelhantes** às da álgebra relacional.

Uma solução possível seria fazer o produto cartesiano de todas as relações envolvidas e seleccionar os tuplos intermédios que responderem à questão. Esta solução pode produzir uma grande relação intermédia em que apenas parte dela esteja armazenada na memória central por isso será necessária a memória secundária e fazer acessos a disco para manipular esta relação, tornando esta solução pouco eficiente.

Na prática, não é viável que o sistema avalie todos os planos de solução, pois o tempo gasto nesta avaliação pode não ser compensado pelas vantagens obtidas pela execução do melhor plano encontrado.

A ideia é seleccionar o plano que não sendo óptimo é o melhor, segundo determinados critérios:

- Optimização baseada em *heurísticas*. Coloca uma questão ao sistema e este, baseado num conjunto de heurísticas, pesquisa no dicionário de dados os índices com interesse para aceder às tabelas envolvidas na questão. Surgem, assim, vários planos de execução. Estes planos são avaliados depois de acordo com as operações envolvidas, às quais estão associados diferentes pesos. O sistema selecciona então o plano de execução com menor peso (maior potencial de optimização).
- Optimização baseada em custos. O SGBD estima os custos envolvidos nos vários planos de execução através da utilização de dados estatísticos acerca das tabelas envolvidas armazenados no dicionário de dados, seleccionando aquele que potencialmente envolver menos custos. A estimativa dos custos depende de três factores:
    - Se o tipo de procura incide directamente sobre uma tabela ou se usa um ficheiro de índices.
    - Se a questão envolve a utilização de junções. Existem três tipos diferentes de junções, cada uma com custos diferentes.
    - A sequência da junção utilizada não é determinada pela ordem em que as tabelas aparecem na cláusula FROM de uma instrução SELECT. A junção de duas tabelas é mais eficiente quando a cardinalidade (número de linhas) da tabela exterior seja inferior à da tabela interior.

Comparação entre os modelos das 1ª e 2ª gerações

O modelo relacional é mais fácil de utilizar mesmo por indivíduos sem conhecimentos de informática, o que não acontece nos modelos de rede e hierárquico. Nos modelos r/h o utilizador é responsável por gerir o acesso aos dados enquanto que no modelo relacional, essa responsabilidade é do SGBD. No modelo relacional, ao contrário do r/h, as alterações na estrutura são fáceis. Os modelos r/h continuam a ser melhores em termos de desempenho mas perdem em termos de simplicidades e flexibilidade.

A diferença fundamental entre os modelos é a forma como se estabelecem as associações entre dados. No modelo r/h as associações à custa de apontadores. No segundo, as associações são implícitas (através de atributos comuns).

No relacional, os dados são organizados numa única estrutura (tabela, relação), estas relacionam-se através de atributos comuns. Diminui o desempenho dos sistemas. Relativamente à capacidade de modelação, o modelo hierárquico obriga à transformação do modelo conceptual, podendo deturpar o modelo original.

As BD de grande e médias dimensão continuam a ser suportados por modelos r/h.

A 3ª GERAÇÃO (NOVOS MODELOS)

A utilização de BD actualmente tem incidido principalmente na área do processamento de dados de gestão. Nos nossos dias, as novas tecnologias permitem-nos ter grande poder de processamento e armazenamento. Temos também à nossa disposição outras tecnologias, como imagem e som, que alargam o âmbito da utilização das tecnologias de informação com a necessidade de novos sistemas capazes de gerir e armazenar estes tipos de dados. Os sistemas de 1ª e 2ª geração não conseguem responder a estas novas funcionalidades, dando-se assim origem à 3ª geração de BDs.

Necessidade de novos modelos

As aplicações que necessitam utilizar estes novos tipos de dados não utilizam SGBDs, mas sim sistemas de ficheiros dedicados. Isto acontece porque os SGBDs convencionais não fornecem as funcionalidades necessárias. Desta forma, estas aplicações tornaram-se estanques a outras aplicações, isoladas do resto do sistema informático da organização. A utilização de SGBD resolvia o problema de isolamento e facilitaria o desenvolvimento e manutenção destas aplicações.

Áreas de SGBD convencionais:

- Necessidade de manipular grandes volumes de dados
- Dados simples, homogéneos com formatos fixos
- Modelos de dados simples e estáveis
- Transacções curtas

Novas solicitações aos SGBD:

- Suportar tipos de dados complexos
- Suportar modelos de dados mais complexos
- Representar formas mais elaboradas de conhecimento
- Transacções mais longas
- Manter a evolução dos objectos modelados

Uma das maiores dificuldades dos SGBD convencionais é ultrapassar o conjunto fixo de tipos de dados que não é adequado para representar a complexidade das novas aplicações. É forçoso, por isso, o desenvolvimento de novos modelos de BDs.

Os SGBDs de 3ªG devem manter ainda as características fundamentais dos SGBDs convencionais, quer na eficiência de manipulação, quer no controlo e segurança, etc.…

Extensões ao modelo relacional

O modelo relacional é um modelo estável, por isso tenta-se aproveitar as suas virtudes, adicionando novos conceitos para ultrapassar as suas fraquezas:

- Suportar tipos de dados complexos, assim como funções para a sua manipulação.
- Acesso procedimental aos dados através de linguagens de interrogação do tipo SQL.

Através de um esforço de extensão do modelo relacional isto pode ser conseguido. A versão 3 do SQL aumenta a Saúde funcionalidades relacionais e apresenta características *OO*.

Uma desvantagem do modelo relacional é o facto de dados terem de estar normalizados. Isto provoca que os dados estejam distribuídos por diferentes relações. Melhor seria que estes dados fossem reorganizados dando a origem a objectos representativos do mundo real.

Uma das soluções apresentadas é a inclusão de uma camada de software entre o SGBD e as aplicações que seria responsável por essa reorganização – o *gestor de objectos*. Que seria responsável por interpretar os dados para as aplicações *OO*.

Outra perspectiva seria a remodelação do próprio modelo relacional de modo a conseguir uma correspondência directa entre os *objectos* do mundo real e as suas versões armazenadas, permitindo a existência de relações não normalizadas ***nested relations***.

Ainda outra situação são as aplicações que necessitam de dados temporais, isto pode ser conseguido através da utilização de mais redundância nos modelos relacionais. É necessário um modelo que represente a temporalidade destes dados: ***modelo relacional temporal***.

Nested relations

Esta é uma extensão ao modelo relacional em que os atributos não necessitam de ser atómicos. O valor de um atributo num tuplo pode ser um conjunto de valores ou um conjunto de tuplos. Assim, podemos ter relações aninhadas dentro de relações, podendo um objecto complexo ser representado por um único tuplo dentro de uma *nested relation*. Existe, deste modo, uma correspondência directa entre os objectos do mundo real e os objectos de uma BD. O esquema de uma BD é assim definido por um conjunto de *nested relations*.

Modelo relacional temporal

Na maioria dos SGBDs actuais apenas é mantido o estado actual da BD, perdendo-se os estados anteriores não sendo possível observar a evolução da BD. Seria conveniente que o tratamento da dimensão temporal da BD fosse tarefa do SGBD. No modelo relacional temporal um tuplo transporta a evolução da realidade do objecto que representa. Por isso, uma BD neste modelo deve ter duas partes lógicas:

Parte histórica – estão armazenados os estados passados da BD.

Parte activa – está armazenado o estado actual da BD.

Estes sistemas implicam utilização de muito espaço de armazenamento, que tem sido o maior obstáculo ao desenvolvimento destes SGBDs.

Os eventos ocorridos no mundo real e a sua representação na BD podem não coincidir no tempo. Por isso, existem duas medidas temporais:

Tempo real, momento em que se verificam os eventos na vida real.

Tempo do sistema, momento em que as alterações provocadas por esses eventos são registadas na BD.

É importante, por isso, distinguir entre as actualizações ocorridas devido a alterações no mundo real (evolutivas/progressivas) e as actualizações que são originadas por situações em que a BD não reflecte o mundo real e que, por isso, tem que ser corrigida (correctivas).

As BDs temporais registam o tempo real e o de sistema, o que permite disponibilizar o estado da BD em qualquer momento passado. Deste modo podem-se colocar diferentes tipos de questões à BD:

- **Correntes**: sem qualquer tipo de referencias temporais.
- **Normais**: actuam também sobre os dados passados, levando em conta eventuais correcções, até ao momento actual.
- ***Rollback***: actuam sobre momentos passados da BD.

Para implementar o modelo temporal é necessário adicionar uma terceira dimensão, **tempo**, criando assim uma visão cúbica da realidade. A adição da dimensão temporal designa-se por *etiquetagem temporal*:

- Intervalos de tempo, para situações em que os dados se mantêm estáveis, durante algum tempo.

- Instantes de tempo, para situações em que os dados variam continuamente e só são validos no momento em que são recolhidos.

Existem duas formas de implementar estas abordagens:

Etiquetagem temporal de tuplos A cada relação é adicionado pelo sistema um par de atributos (FROM/TO) temporais que indicam o intervalo de tempo em que esse tuplo é ou foi válido.

Etiquetagem temporal de atributos O par de atributos temporais é adicionado a cada atributo da relação. O atributo passa a ser constituído pelo seu valor e o intervalo durante o qual esse valor é ou foi válido. Deste modo, num tuplo onde apenas alguns atributos foram alterados, apenas esses passam para a parte histórica.

A etiquetagem temporal de tuplos é mais simples de implementar mas consome mais espaço de armazenamento.

A etiquetagem temporal de atributos é mais económica e faz mais sentido, pois uma alteração tipicamente envolve apenas alguns atributos. Por outro lado, as transacções são mais difíceis de resolver, porque para compor um tuplo valido é preciso aceder à parte histórica e atender aos intervalos de tempo de cada atributo. Aos atributos que não variem no tempo não lhes são atribuídos pelo sistema os atributos temporais e estes deixam de fazer parte do segmento histórico da BD (p.ex. a chave, id).

A chave de uma relação no modelo relacional temporal não deve mudar ao longo do tempo nem se devem repetir no tempo. As restrições de integridade resumem-se a que uma relação não pode conter tuplos, cujos valores dos atributos visíveis, sejam iguais em períodos de tempo que se sobrepõe ou são consecutivos e, um tuplo não pode ter um intervalo de tempo nulo.

Para poder modelar uma BD num modelo temporal é necessário que os esquemas passados numa BD estejam disponíveis para consulta. O próprio esquema da BD é uma componente variável no tempo, ou seja, os objectos representados na BD estão associados a uma intervalo de vida que corresponde ao tempo em que o objecto está operacional na B D.

O modelo Lógico/dedutivo

Além dos dados de uma organização o SGBD também deveria poder gerir outras formas de conhecimento, como por exemplo conceitos. Numa organização o mesmo conceito pode ser interpretado de formas diferentes por agentes diferentes. Isto pode originar incongruências de comportamento. Se o SGBD também gerisse este tipo de conhecimento este problema não existiria.

O que se pretende é que este conhecimento (conhecimento procedimental) seja tratado e armazenado como se tratasse de conhecimento factual.

Uma das formas de conhecimento procedimental são as restrições de integridade explícitas, as quais o modelo lógico/dedutivo tenta abordar. Por um lado, armazenar dados, por outro, armazenar regras que permitam deduzir factos adicionais.

Características da programação em lógica

Num sistema desenvolvido em lógica apenas é necessário definir o que se pretende e não como se pode obter. Um programa em lógica é constituído por factos que traduzem os dados e as regras que implementam a parte dedutiva (conhecimento procedimental).

Lógica e BDs

Um programa lógico pode ser visto como uma BD dedutiva. Numa BD dedutiva definem-se dois tipos de conhecimento: *conhecimento*

extensional que corresponde aos factos e o *conhecimento intencional* que corresponde ao conhecimento procedimental.

Deste modo, o conhecimento intencional permite criar novos factos em *run time*, evitando os volumes enormes de informação factual. A vantagem de definir regras sente-se no desenvolvimento e manutenção das aplicações, tornando desnecessário trabalhar em código de programação e permitindo ao utilizador o que quer e como pode obter. Este conhecimento procedimental passa a estar concentrado no SGBDDedutivo e não espalhado pelas aplicações.

Os sistemas de programação em lógica não permitem tratar grandes volumes de informação mas a utilização de BDs para armazenar estes dados e a sua manipulação resolvem este problema. Nada impede que os tuplos de uma relação armazenem factos em *Prolog* e a combinação das facilidades destes dois sistemas é a solução mais directa para o desenvolvimento de BD dedutivas.

Existem duas abordagens para o desenvolvimento de BD dedutivas:

Homogénea. A parte intencional (regras) e a parte extensional (factos) estão juntas o que tem a vantagem de uniformidade e transparência na linguagem utilizada (uma BDD em *Prolog*).

Heterogénea. Regras e factos estão separados e quando uma questão é colocada o SGBD decide a que parte vai colocar a questão.

Tipos:

- *Acoplamento fraco*. Existe cooperação entre os dois mas são autónomos.
- *Acoplamento forte*. As componentes são diferentes mas fortemente ligadas.

O desafio é conciliar as características das duas tecnologias para aproveitar melhor esta associação. Para isso a interface entre o sistema relacional e lógico pode ser analisada a três níveis:

Nível lógico. Considera a interface entre as aplicações e a BD, tratando o problema do acoplamento entre as características lógicas e as características relacionais para a manipulação dos dados.

Nível funcional. Considera o acoplamento entre os mecanismos de inferência das linguagens lógicas e as funções de gestão da BD.

Nível Físico. Trata da ligação entre a organização física dos tuplos do nível relacional e as regras de dedução e o modo como estes cooperam.

O Modelo OO

Bases de dados OO Este modelo surge na sequência dos progressos ocorridos com as linguagens de programação OO. Dispondo de grandes capacidades de modulação, adequa-se aos requisitos das áreas de aplicação mais complexas. Estas novas áreas necessitam manipular dados complexos com formatos variáveis, difíceis de implementar com BD relacionais. Fortemente inspiradas nas linguagens de programação OO as BDOO fornecem facilidades para representar estes tipos de dados complexos tornando-os adequados para suportar os requisitos nas novas áreas de aplicação.

Um dos objectivos da BDOO é conseguir uma correspondência directa entre os objectos do mundo real e as suas representações nas BD, por outro lado, o tipo de operações passam a ser específicas dos objectos definidos nas respectivas classes. A ideia é permitir que os utilizadores trabalhem com conceitos mais próximos do mundo real

Conceitos

Objectos – No domínio OO, um objecto é uma de muitas coisas que constitui o problema. Um objecto pode ter um nome, um conjunto de atributos (estados), e um conjunto de operações (alteram os estados) ou serviços. Um objecto pode ser único ou pertencer a uma classe de objectos similares.

Classes – Define um grupo de objectos com atributos semelhantes, operações comuns, uma relação comum com outros objectos na classe e uma semântica comum. A classe tem um nome, um conjunto de atributos e um conjunto de operações. Um objecto pode ser visto como uma instanciação de uma classe.

Encapsulamento – Permite a um objecto agregar num todo, todos os atributos que o caracterizam bem como toada as operações que podem alterar esses atributos. A única forma de alterar esses atributos é através de operações internas. Desta forma os objectos podem ser tratados e re-utilizados como componentes modulares de um programa.

Polimorfismo – Métodos com o mesmo nome podem realizar operações diferentes por pertencerem a objectos diferentes.

Bases de dados orientadas a objectos (BDOO) vs. Bases de dados relacionais (BDR)

As BDOO guardam todo o objecto incluindo o código e os métodos, ao contrário do que acontece nas BDR em que o modelo de dados é disperso por tabelas.

Actualmente, os sistemas distribuídos, interfaces gráficos, dados que evoluem ao longo do tempo são um ambiente propício aos objectos e para armazenar esta informação os SGBDR não são suficientes eficientes. Um SGBDOO permite reutilizar os objectos nele contidos para construir objectos complexos composto por outros objectos, e num ambiente distribuído permite a heterogeneidade de servidores de objectos e movimentar objectos de um servidor para outro de modo transparente quer para o utilizador quer para os outros objectos da BD.

O tempo de desenvolvimento da SGBDOO é mais curto e a sua manutenção mais baixa porque cada problema pode ser resolvido separadamente, ou seja, trabalha-se no objecto a ser alterado e quando o problema estiver resolvido substitui-se o objecto sem que o sistema pare de funcionar.

Quanto à tolerância a faltas, as transacções nos modelos convencionais caracterizam-se pela sua curta duração e pelo facto de serem atómicas (ou todos os seus passos sucedem ou nenhum sucede); nas novas áreas de aplicação, estas características alteram-se por exemplo no caso de uma aplicação CAD as transacções podem ter uma duração arbitrária (minutos, hora, ou dias). Não é tolerável portanto, que devido a uma falha circunstancial, se percam horas ou dias de trabalho.

Praticamente, nenhum sistema actual possui todas as funcionalidades, consideradas fundamentais de um verdadeiro SGBDOO, sendo os problemas de controlo da concorrência e recuperação/tolerância a falhas uns dos problemas de mais difícil solução.

Outra desvantagem dos SGBDOO decorre das dificuldades existentes ao nível do interface do utilizador. Enquanto, que num mundo relacional existe a facilidade de execução de questões *ad hoc* por parte dos utilizadores finais, num mundo OO o facto de todas as alterações já estarem pré-definidas e todas as referências entre objectos se fazerem por meio de apontadores (Object IDentifier - OIDs) vem trazer algumas dificuldades a este nível.

Outra grande diferença entre BDOO e BDR é o modo de identificar univocamente os dados, enquanto que as BDOO usam o OID que se mantém

constante após execuções de aplicações que acedem e alteram a BD, nas BDR os tuplos (instância do esquema de uma relação), são identificados por chave (um valor que é guardado no próprio tuplo). A utilização de OIDs permite o controlo mais fácil sobre o armazenamento de objectos. Quando um objecto é removido da BD, todas as referências ao seu OID devem desaparecer de modo que a integridade da BD se mantenha (não hajam referência erradas noutros objectos).

As BDR podem ser utilizadas em casos em que a representação dos dados pode ser feita através de tabelas, em que estes dados não são complexos e sejam de tamanho pré-definido; por outro lado as BDOO servem melhor os propósitos de um conjunto de dados complexos, evolutivos, com modelos de muitas relações e informação de tamanho indeterminado com som e imagem.

A necessidade de standards OO

Ainda não existe um standard para as BDOO (embora este esteja a ser estabelecido por um consorcio de construtores), e cabe ao construtor seguir ou não esse standard. As BDR são mais estáveis neste aspecto por já existirem há mais tempo e a sua posição no mercado bem como nos métodos de análise utilizados pelos profissionais de BD está bem estabelecida.

Sendo assim, a decisão final cabe sempre ao cliente, tendo em conta as necessidades do seu projecto e os pontos fortes ou não dos SGBD em relação a esse projecto.

Unificação relacional-OO

Existe grande competição entre os modelos relacional e OO devido à competição entre os produtores de ambas as partes. A unificação dos modelos pretende que exista uma evolução no sentido OO sem perder a bem conhecida tecnologia relacional. Estes sistemas incluem as melhores características de cada um dos dois modelos. As justificações para estes modelos são fortes: manter os investimentos feitos no modelo relacional e evoluir para novas tecnologias OO.

BASES DE DADOS DISTRIBUÍDAS

Os SBD distribuídos podem ser classificados em homogéneos, em que todos os nodos (sistemas de computação da rede) usam o mesmo SGBD e em heterogéneos, em que existem SGBDs diferentes nos vários nodos.

Conceitos

Basicamente uma rede de computadores é um conjunto de vários computadores interligados por meios de comunicação com o objectivo de partilhar recursos.

LAN constituída por recursos próximos uns dos outros ligados por cabo directo.

WAN recursos mais distantes ligados por redes de comunicação próprios ou alugados.

As LAN utilizam exclusivamente tecnologia de comunicação digital, as WAN utilizam tecnologia analógica. Daí, as taxas de transmissão são inferiores às da LAN.

Em termos de tratamento informático, um ambiente de BD distribuída envolve 2 elementos: dados e processamento.

Sistemas centralizados

É a configuração mais simples de BDs distribuídas. Aqui, os dados e o processamento localizam-se na mesma máquina, o SGBD e as aplicações correm na mesma máquina. Isto acontece por motivos históricos. Estes são sistemas proprietários de difícil manutenção e são sistemas cuja expansão é muito cara.

Arquitectura cliente/servidor

Surgiu com a possibilidade de ligar computadores pessoais. Segundo esta arquitectura, há um cliente que faz pedidos a um servidor que os atende. Uma máquina mais dotada de capacidade de armazenamento permitiria armazenar grandes quantidades de dados que seriam partilhados pelos clientes. Sendo a unidade de transferência entre cliente/servidor o ficheiro, surgem dois problemas: tráfego na rede e partilha de dados.

Os primeiros sistemas limitavam-se a permitir a partilha de ficheiros não tendo qualquer intervenção no processamento das aplicações, dai a evolução da arquitectura cliente/servidor ter evoluído para um modelo de servidor de BDs. Este servidor responde às solicitações dos clientes apenas com os dados necessários, diminuindo o tráfego e aumentado o nível de concorrência.

O objectivo é separar as aplicações das funções de gestão de BDs.

A característica fundamental de um sistema BD cliente/servidor é a divisão do processamento global pelas máquinas clientes e as máquinas servidor serem responsáveis pela manipulação e gestão da BDs.

Uma nova configuração pressupõe a existência de três níveis: organizacional, com um servidor de BDs; departamental, com um servidor de aplicações; e o nível local com os diferentes clientes.

Sistemas distribuídos

Os sistemas distribuídos de BD pretendem realizar a distribuição dos dados pelos diferentes centros de processamento, e a partilha concorrente desses dados pelos centros. Surge a visão de uma BD lógica, distribuída fisicamente por vários nodos que cooperam entre si. A abordagem distribuída de uma BD é interessante por dois motivos:

- Reflectir o estado distribuído de uma organização, a própria organização está localizada em diferentes locais.

- Proporciona facilidades de crescimento ás organizações, através da simples adição de mais máquinas.

Como é que se processa então a partilha dos dados pelos diferentes nodos?

As soluções que os SGBDs utilizam para a distribuição dos dados são normalmente híbridas, passando pela replicação em conjunto com a fragmentação.

Replicação de dados

Uma alternativa é replicar os dados mais frequentemente solicitados nos nodos que os solicitam; também devido à tolerância a falhas que passam a ocorrer no nodo e não incluem a BD. A grande dificuldade disto é que se torna necessário manter a coerência global da BD, e logo as cópias dos dados sincronizadas. E para garantir isso, é necessário que qualquer actualização entre em vigor quando a todas as réplicas ficarem sincronizadas. Em caso de falha todas as réplicas devem falhar. Isto é de difícil implementação (falhas de comunicação, falhas no próprios nodos…).

Para facilitar a replicação a inconsistência da BD pode ser permitida em alguns casos. Sendo a sincronização feita logo que a transacção termine ou então actualizar quando várias translações terminem.

Existem vários tipos de replicação:

- **Primary state replication:** em que apenas um dos nodos pode fazer a actualização.
- **Dynamic ownership:** em que a autorização para actualizar passa por todos os nodos.
- **Replicação simétrica:** qualquer réplica pode ser actualizada a qualquer momento. Isto levanta problemas de conflito entre sincronizações, quando duas transacções tentam actualizar a BD simultaneamente em duas ou mais réplicas.

Fragmentação de dados

O conceito de fragmentação está associado à utilização mais frequente por parte de alguns nodos de dados específicos. A fragmentação é a forma como uma relação pode ser dividida em tabelas menores, sendo estas distribuídas pelos nodos. A tabela principal é uma tabela lógica e os seus fragmentos são tabelas com existência física. Dois tipos de fragmentação:

- *Fragmentação vertical*. A tabela divide-se em subconjuntos da tabela lógica. (resultados colunas) Projecção da tabela original.
- *Fragmentação horizontal*. A tabela divide-se em subconjuntos da tabela lógica. (resultados linhas) Selecção da tabela original.

Características de uma BD distribuída

Um sistema de BD distribuídas deve-se apresentar ao nível aplicacional como se de um só sistema se tratasse. Isto combina dois conceitos divergentes: distribuição física dos dados e integração lógica dos dados.

Os utilizadores e as aplicações não necessitam de ter conhecimento da localização dos dados, tornando-se o aspecto da transparência muito importante. Isto facilita os aspectos desenvolvimento e de manutenção da rede. Os utilizadores e as aplicações não são afectados por qualquer alteração na localização dos dados porque é o SGBD que resolve os detalhes de navegação na rede.

Disponibilidade permanente. Alterações na arquitectura da rede não devem impedir que os restantes nodos continuem as suas operações.

Fragmentação Transparente. Os utilizadores devem ver uma tabela fragmentada como se uma tabela só se tratasse.

Duplicação transparente. O SGBD é o responsável por se manter a coerência dos dados replicados.

Processamento de questões distribuído. A resposta de uma questão pode necessitar da cooperação de diferentes nodos.
Independência entre hardware, SO e tecnologia de rede. Ainda que existam diferentes SO, SGBDs e hardware os utilizadores devem poder aceder aos dados através do mesmo interface.

Concepção de BDs distribuídas

É a forma como a BD global vai ser distribuída pelos nodos. Isto vai influenciar o comportamento do sistema global.

- *Top-down*. Uma BD existente é armazenada em diferentes nodos, resulta num ambiente distribuído homogéneo.
- *Bottom-up*. Integrar várias BDs existentes numa BD global distribuída por diversas máquinas.

O conceito *top-down* pretende que as tabelas que suportam o modelo conceptual da BD sejam distribuídas pelos diferentes nodos. A ideia é maximizar o processamento local, os dados devem ficar mais próximos de quem os necessita. O problema é decidir a que nodos correspondem as tabelas, quais as fragmentar e quais os dados a duplicar.

A concepção de BDs distribuídas implica dois problemas que são a configuração da BD e a sua distribuição. Estas duas fases podem ser divididas. A configuração da BD pode ser resolvida através da utilização de heurísticas, enquanto que a distribuição da BD pode, posteriormente ser resolvida, definindo os esquemas locais da BD.

Existem duas aproximações aos problemas da distribuição dos dados. Uma, em que se analisam as transacções para decidir que nodos é que são utilizados para armazenar os dados; e outra, tendo em conta que o sistema se altera ao longo do tempo, continuamente se faça a redistribuição dos dados.

Processamento e optimização de questões

A diferença entre o processamento e optimização de questões nos sistemas centralizados e distribuídos está na utilização de meios de comunicação e no aproveitamento do paralelismos dos sistemas distribuídos. O objectivo da optimização nos sistemas distribuídos é ter atenção às transacções que envolvem dados localizados em diferentes nodos. Por um lado, tentar minimizar o tempo de resposta, por outro, minimizar a utilização de recursos de processamento.

Sempre que uma questão não pode ser resolvida localmente é necessário dividir o processamento pelos nodos.

O processamento de uma questão consiste na sua decomposição em sub-questões, de acordo com os dados distribuídos pelos nodos. A solução de uma questão corresponde, portanto à execução de sub-questões provavelmente em paralelo, por fim reunidas para obter a resposta final.

A optimização das questões dá-se em dois níveis. Primeiro a optimização da questão global, dividindo em sub-questões e decidindo que nodos respondem a essas sub-questões. Segundo, como nos sistemas centralizados, a optimização de cada sub-questão no nodo em que é executado.

BDs distribuídas heterogéneas (*bottom-up*)

A heterogeneidade das organizações tem levado a que estas disponham de recursos distribuídos geograficamente. Do mesmo modo os sistemas informáticos que correspondem a esta distribuição são necessariamente diferentes. Deste modo surgiu a necessidade de integrar os diferentes sistemas informáticos de uma organização conciliando-os. Isto pode ser feito através de uma abordagem *bottom-up*, criando assim mecanismos que conciliem os diferentes sistemas da organização. Os sistemas heterogéneos não devem interferir com as BDs locais existentes. Tendo em conta que as

BDs locais não forma desenvolvidas com a intenção de cooperarem umas com as outras, par alem dos problemas de ordem técnica, surgem dificuldades de compatibilização entre elas.

Para alem da necessidade de transparência entre as aplicações é necessária também uma transparência de localização dos dados. Existem três abordagens para uma BD heterogénea que se explicam em seguida.

Utilização de *Database Gateways/Middleware*

Implicam a existência de SGBDs diferentes mas que são baseados num mesmo modelo de dados.

A função dos *gateways* é traduzir os diferentes formatos de dados e as suas implementações de modo a que cada SGBD veja os outros da mesma forma.

Os *gateways* têm algumas limitações porque não têm suporte para transacções não locais. Ou seja, não há controlo de concorrência nem recuperação a falhas para transacções que envolvam dois sistemas.

Outra solução idêntica consiste em esconder do nível aplicacional as especificações de cada SGBD colocando entre os dois uma camada (middleware) que uniformiza o interface entre clientes e servidores.

Existem duas propostas de estandardização para o *middleware* a ODBC (Open Database Connectivity) da Microsoft para aplicações Windows e SQL, outra é o IDAPI (Integrated Database Application Programming Interface) que permite o acesso tanto a servidores SQL como a servidores Xbase.

A grande vantagem do *middleware* é que permite a portabilidade e simplifica as soluções obtidas. Destes dois o que está mais implementado é o ODBC.

Integração no modelo global

Os SGBDs são baseados modelos diferentes portanto é necessário utilizar um modelo global onde estes modelos locais estão mapeados. A ideia é criar uma camada sobre os diferentes sistemas que dê a ideia de um sistema homogéneo.

Foram definidos dois grupos de sistemas:

- *Esquema unificado*. Utiliza um modelo rico que integra nele os modelos das BDs existentes num único modelo global.
- *Multibase dados*. Não se procura um esquema global, cada BDs funciona separadamente mas coopera com as outras.

A tecnologia OO com as suas capacidades de modelação superiores oferece maiores possibilidades para a concepção de um modelo unificador. Assim, não seria necessário substituir sistemas antigos mas sim integra-los através de um modelo de dados OO.

As vantagens do *multibase* são que, além de manter a independência dos diversos sistemas, permite a coordenação entre eles. Mais do que isso, coordena a execução das transacções globais pelos diferentes sistemas, mantendo a integridade do sistema global. Cada SGBD local mantêm o controlo dos dados locais e determina o que é partilhado e como. São as BDs locais que mantêm o seu esquema de BD.

Objectos distribuídos

Este modelo adequa-se aos ambientes distribuídos em que os objectos estão geograficamente dispersos comunicando entre si através de mensagens (modelo OO).

O desenvolvimento de sistemas no futuro é chamado por Desenvolvimento por junção de componentes em que se desenvolvem apenas os objectos (componentes) que sejam necessários para alguma funcionalidade do sistema.

Devido às diferentes proveniências dos objectos são necessários mecanismos que permitam que estes comuniquem entre si sem que tenham

de residir numa mesma máquina, dando origem a uma ambiente de objectos distribuídos por várias máquinas ligados por redes de comunicação.

A integração entre objectos é basicamente uma relação cliente/servidor. Se um objecto responde a uma mensagem está a comportar-se como um servidor.

O OMG (Object Managment Group) fornece uma base para a partilha de objectos independentemente da linguagem em que foram desenvolvidos, permitindo que um objecto cliente aceda aos serviços de um objecto servidor sem ter que conhecer os seus pormenores (CORBA).

A arquitectura CORBA é constituída por quatro componentes:

1. **ORB.** Gere a interacção entre os objectos, mantendo um registo da localização de todos os objectos no sistema, que serviços esses objectos disponiblizam e como podem ser invocados.
2. **Object Services.** Funcionalidades (transacções, controlo de concorrência, segurança…) que enriquecem o ORB para que este suporte um ambiente de objectos distribuídos.
3. **Common Facilities.** Permitem a definição de interfaces para apoiar a execução das aplicações.
4. **Application Objects.** Conjunto de objectos específicos que traduzem funcionalidades das aplicações.

O ORB permite a transparência entre o objecto servidor e o objecto cliente, parecendo que ambos estão na mesma máquina.

Através da definição de interfaces (IDL), cada objecto pode comunicar com os outros, definindo os serviços que oferece.

Através do IDL, é possível modificar a implementação de um objecto sem afectar os seus objectos clientes.

Com o IDL, é possível integrar sistemas pré-existentes sem alterar o seu código fonte integrando-os com os novos sistemas.

A associação entre ORBs e SBDs permite associar as facilidades de armazenamento dos SBDs com as facilidades de comunicação dos ORBs. Ao contrário de deslocar objectos entre máquinas, o cliente envia mensagens ao servidor e espera pelas respostas. Com as vantagens de dividir o processamento pelas máquinas e diminuir o tráfego na rede.

Assim, é possível desenvolver objectos de um nível mais elevado que melhor representem e se comportem como os objectos correspondentes do mundo real.

Gestão de BD distribuídas (SGBDD)

As transacções de um SGBDD são mais complexas porque envolvem cooperação entre diferentes nodos. Continua a ser necessário manter as características das transacções (atomicidade, isolamento, persistência, integridade). Por isso, é necessário uma coordenação entre as sub-transacções locais que constituem uma transacção global.

Cada sub-transacção deve ser atómica (ou todas as suas operações acontecem ou nenhuma acontece).

A transacção global deve ser atómica (todas as suas sub-transacções acontecem ou nenhuma acontece.

A finalização de uma transacção distribuída tem que ser coordenada não apenas verificando o estado final de cada sub-transacção mas aguardando que uma instrução *commit* lhe seja enviada por parte de uma entidade coordenadora da transacção global antes de ser finalizada.

A abordagem mais utilizada é o protocolo 2PC (two phase commit) em que um nodo funciona como coordenador e a transacção funciona do seguinte modo:

1ª fase – o coordenador envia uma mensagem de preparação *commit* a todos nodos participantes e espera por uma resposta positiva ou negativa.

2ª fase – se todas as respostas forem afirmativas envia uma mensagem de *commit* a cada nodo. Basta que um dos nodos tenha respondido negativamente para que ele envie uma mensagem de *abort* e peça um *rollback* das respectivas sub-transacções locais.

O 2PC é limitativo porque basta que o nodo principal falhe para que todos os outros nodos não saibam se devem fazer *commit* ou *abort.*

Outro problema grave é o *deadlock*. São mais difíceis de resolver nos SGBDD. Se ele acontece localmente pode ser resolvido no nodo local. Se o *deadlock* envolver recursos de ouros nodos, os nodos locais podem não se aperceber disso. Para isso pode-se construir um grafo global a todos os nodos com informações sobre estado de *locks* no sistema. O grande inconveniente é que aumenta o tráfego e se isto falhar todo o sistema pára. A melhor solução é a utilização de grafos locais juntamente com mecanismos de *timeout* para localizar ou detectar *deadlocks* distribuídos.

Vantagens, limitações e perspectivas futuras

Um SGBDD reflecte de uma forma transparente a estrutura da organização.

Permite acompanhar a evolução da organização, o seu crescimento é virtualmente ilimitado, bastando para isso a adição de novos nodos na rede.

Tem uma maior resistência a falhas porque se um nodo falhar não significa que todo o sistema pare.

O desempenho melhora tornando-se mais rápido o processamento das transacções porque este pode ocorrer em paralelo em diferentes nodos.

As técnicas de controlo da concorrência e recuperação a falhas, etc. partem do princípio que são poucos os nodos que cooperam entre si. Se com o crescimento das organizações é normal que alguns nodos estejam inoperacionais, diminuindo o desempenho da organização.

No futuro as organizações têm tendência a crescer pelo que o desenvolvimento de um sistema distribuído venha facilitar esse crescimento.

DESEMPENHO E ESCALABILIDADE

Neste momento, as dimensões das BDs medem-se em *terabytes*. Isto agravado com o aumento de utilizadores, tem vindo a agravar o desempenho destes sistemas de BD.

Máquinas de BD

Os computadores convencionais não se mostram adequados ao processamento não numérico característico dos sistemas de BDs que trabalha com linguagens de alto nível. Sendo necessário desenvolver mecanismos capazes de transformar essas representações de alto nível em representações de baixo nível que possam ser entendidas pelo hardware.

O buraco entre estes níveis tem sido preenchido com software, aumentando ainda mais o problema do desempenho pelo aumento da distância entre a máquina e o SGBD.

Conceitos

Uma forma de aliviar a máquina principal (host) é separar algumas funções mais pesadas para outra máquina (backend). Sempre que uma aplicação precisar de serviços de BD passa esse pedido à outra máquina que processa e devolve o resultado.

A máquina host é libertada das funções de SGBD e a máquina backend não lida com o nível aplicacional. Para isto é simplesmente necessário reorganizar a carga de trabalho do sistema mas continua a existir um buraco entre o nível aplicacional e o nível de hardware.

Hoje já é possível tratar deste problema diminuindo a distância entre o nível da máquina e o nível das aplicações através do desenvolvimento de hardware especializado. Este deve ser capaz de executar as operações básicas de GBD.

Limitações

Estes sistemas estão em permanente desenvolvimento, assim ou as máquinas de BD se destacam ou é difícil justificar a mudança (porque mais tarde ou mais cedo aparece um modelo melhor).

As máquinas de BD são diferentes das convencionais pelo que é necessário converter os sistemas instalados. Esta opção é cara e nem todas as organizações podem faze-lo.

A tecnologia das máquinas de BDs implementa as funções básicas do modelo relacional pelo que organizações com outros modelos instalados, devido à incompatibilidade, não adoptam este sistema.

Sistemas paralelos de BD

Devido à evolução na tecnologia dos processadores surgirem os computadores multi-processadores. Se estas máquinas não resolverem o problema do congestionamento I/O não solucionam o problema do desempenho.

Para resolver este problema existe a solução de armazenar o mais perto possível os dados que são frequentemente processados em conjunto (clustering). Outra solução é distribuir os dados pelas memórias secundárias para tirar partido do paralelismo I/O (declustering).

Arquitecturas multi-processador

Existem duas abordagens no desenho de sistemas multi-processador:

SMP (Symetric MultiProcessing): shared memory

MPP (Massively Parallel Processing): shared disks ou shared nothing

No SMP os proessadores encontram-se ligados por um barramento, usando a mesma memória central e os mesmos discos.

A desvantagem é que existe disputa pelos dados em disco, aumentando os períodos de espera de cada processador e diminuindo o desempenho do sistema.

O MPP distingue entre shared disks em que associado a cada processador se encontra uma memória central mas todos os processadores acedem a todos os discos; o shared nothing cada processador tem a sua memória central privada e o seu sistema de discos.

A maior vantagem destes dois sistemas é que pode crescer continuamente, bastando acrescentar novos módulos.

Com o aumento dos módulos a gestão do sistema complica-se, é necessário garantir que as memórias centrais de cada processador estão sincronizadas. Na abordagem shared nothing a escalabilidade é máxima porque cada processador depende dos seus próprios recursos e a rede de ligação entre os processadores é usada apenas para troca de mensagens (dados de coordenação, resultados…).

Com os sistemas paralelos dá-se um retorno aparente aos sistemas proprietários, mas na verdade os grandes sistemas paralelos poderão ter como solução a associação de pequenas máquinas interligadas numa solução do tipo shared nothing.

Suporte de SGBD paralelos

O SMP tem a facilidade de ser uma abordagem simples parecida com os sistemas mono-processador e ser uma tecnologia estabilizada. A principal, desvantagem é a sua menor escalabilidade e difícil adaptação a ambientes mais exigentes.

Numa tecnologia shared nothing a BD encontra-se distribuída pelos diferentes discos dos módulos e cada módulo comporta-se como uma máquina separada, comunicando com os outros módulos através da rede. a vantagem é que cada modulo possui uma parte da BD que pode manipular não sendo preciso coordenar a partilha dos dados. Por isso, é menos flexível pois um nodo muito solicitado pode ficar muito sobrecarregado. Para evitar que a falha de um módulo afecte toda a BD deve-se recorrer à replicação dos dados. A integridade dos dados deve ser mantida e por isso é necessário recorrer a um protocolo que coordene a sua actualização como o 2PC.

Numa tecnologia shared disk existe a vantagem da flexibilidade na utilização dos recursos pois o sistema decide dinamicamente que processador utilizar para executar uma operação. O inconveniente é a complicação pelo acesso à memória, sendo que é necessário garantir que os processadores não actualizam simultaneamente os mesmos dados (utilizar um gestor de *locks*).

As duas arquitecturas de BD paralelas podem suportadas por qualquer máquina MPP. Se uma máquina shared nothing desviar os acessos a dados para o módulo que os possui pode suportar uma máquina shared disks. Do mesmo modo se uma máquina shared disks tratar cada disco como acessível apenas por um módulo pode suportar uma máquina shared nothing.

Formas de paralelismo

O benefício do paralelismo é significativo na perspectiva de resolução de questões. Os sistemas relacionais são favoráveis à execução paralela. O paralelismo *interquery* é a forma mais simples e fácil de implementar o paralelismo. Utiliza os algoritmos já desenvolvidos para SGBDs mas não aproveita todos os recursos da máquina em todas as situações. Esta forma limita-se a atribuir questões diferentes a cada processador.

Outra solução é o paralelismo *intraquery* em que as sub-operações que constituem uma operação global podem ser escalonadas pelo sistema para que algumas sejam executadas em paralelo. Aumentando o grau de fragmentação das relações aumenta-se as possibilidades de processamento em paralelo. Assim, existem três configurações de fragmentação:

Fragmentação por gamas. Os tuplos são separados segundo as suas características (alunos do mesmo curso; professores do mesmo curso…). A desvantagem é que o resultado pode ser bastante desequilibrado.

Fragmentação round-robin. Os valores dos atributos são irrelevantes, a distribuição é feita pelos discos da seguinte forma (exemplo 3 discos): 1º tuplo-1º disco; 2º tuplo-2º disco; 3º tuplo-3º disco; 4º tuplo-1º disco…

Fragmentação por hashing. Os tuplos são armazenados num determinado disco dado por uma função de hashing.

A fragmentação deve ser adequada ao tipo da situação da BD.

Além do paralelismo das operações sobre uma relação existe paralelismo (intraquery) sobre relações distintas.

- Paralelismo entre as operações sobre relações distintas
- Para cada relação:

- Paralelismo na sequencia das várias operações
- Paralelismo na execução de cada operação recorrendo à fragmentação

Vantagens de SGBD paralelos

Com os SGBDs paralelos aumenta o desempenho e a possibilidade de escalabilidade dos SBDs.

A sua utilização torna possível obter respostas a questões complexas em tempo útil. Isto não seria possível com os convencionais.

O aproveitamento do paralelismo em situações de sistemas de BDs relacionais. Assim, é possível transitar para as máquinas multi-processador sem alterar o código das aplicações antigas. Só é preciso que o SGBD implemente paralelismo.

BD em memória central

Para minorar os efeitos da disparidade existente entre velocidades de processamento e a velocidade dos acessos à memória central surgiu uma solução mais radical que seria o armazenamento da BD em memória central. Quantos mais dados existirem na memória central menos acessos haveria a disco o que melhorava o desempenho. O preço das memórias centrais está a diminuir o que faz sentido em trazer mais dados para memória central.

As BDs em memória central têm características diferentes: a memória central é volátil, os tempos de acesso são mais rápidos e o tempo de acesso aos dados é igual independentemente do acesso ser sequencial ou aleatório.

A optimização do processo de questões não se põe porque não existem acessos a discos. Da mesma maneira o mesmo se passa com o controlo de concorrência pois as transacções são finalizadas mais rapidamente.

Continua a ser necessária manter uma cópia de segurança da BD e um ficheiro de log em que fiquem registados os registos afectados por uma transacção. Este ficheiro deve residir em memória secundária. Uma BD típica nunca poderá ser armazenada por completo em memória central mas a sua utilização pode melhorar o desempenho global de um sistema.